Diseño de una arquitectura para el control telemático de sistemas ciber físicos basada en microservicios y con tolerancia a fallos.

Cobo Jaramillo, Nikolás Mateo

Departamento de Eléctrica, Electrónica y Telecomunicaciones

Carrera de Ingeniería en Electrónica y Telecomunicaciones

Trabajo de titulación, previo a la obtención del título de Ingeniero en Electrónica y Telecomunicaciones

Ing. Alulema Flores, Darwin Omar, PhD.

18 de agosto del 2021

Curiginal

Document Information

Analyzed document	Cobo_Nikolas_Trabajo_Titulacion_Capitulos.pdf (D111483483)
Submitted	8/20/2021 2:44:00 AM
Submitted by	Alulema Flores Darwin Omar
Submitter email	doalulema@espe.edu.ec
Similarity	0%
Analysis address	doalulema.espe@analysis.urkund.com

Firmado electrónicamente por:
DARWIN OMAR ALULEMA FLORES

Sources included in the report

W	URL: https://www.colibri.udelar.edu.uy/jspui/bitstream/20.500.12008/25418/1/VR20.pdf Fetched: 8/20/2021 2:45:00 AM	3
W	URL: https://docplayer.es/78256758-Proyecto-fin-de-carrera-ingenieria-de-telecomunicacion.html Fetched: 12/11/2019 8:06:43 PM	1
W	URL: https://www.riunet.upv.es/bitstream/handle/10251/59657/Julio%20Torres%20Bataller%20-%20TFM%202014.pdf?sequence=1&isAllowed=y Fetched: 2/15/2021 4:25:27 AM	1
W	URL: http://www.math.utep.edu/Faculty/cmmundy/Math%202301/Solution_Manual.pdf Fetched: 8/20/2021 2:45:00 AM	1

DEPARTAMENTO DE ELECTRICA, ELECTRÓNICA Y TELECOMUNICACIONES

CARRERA DE INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES

CERTIFICACIÓN

Certifico que el trabajo de titulación, "**Diseño de una arquitectura para el control telemático de sistemas ciber físicos basada en microservicios y con tolerancia a fallos**" fue realizado por el señor **Cobo Jaramillo Nikolás Mateo,** el cual ha sido revisado y analizado en su totalidad por la herramienta de verificación de similitud de contenido; por lo tanto cumple con los requisitos legales, teóricos, científicos, técnicos y metodológicos establecidos por la Universidad de las Fuerzas Armadas ESPE, razón por la cual me permito acreditar y autorizar para que lo sustente públicamente.

Sangolquí, 18 de agosto del 2021

Firma:

Firmado electrónicamente por:
DARWIN OMAR ALULEMA FLORES

.....

Dr. Alulema Flores, Darwin Omar

C. C: 1002493334

DEPARTAMENTO DE ELECTRICA, ELECTRÓNICA Y TELECOMUNICACIONES

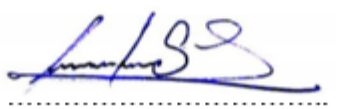
CARRERA DE INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES

RESPONSABILIDAD DE AUTORÍA

Yo, **Cobo Jaramillo, Nikolás Mateo** con cédula de ciudadanía N°1723503379, declaro que el contenido, ideas y criterios del trabajo de titulación**: "Diseño de una arquitectura para el control telemático de sistemas ciber físicos basada en microservicios y con tolerancia a fallos",** es de mí autoría y responsabilidad, cumpliendo con los requisitos legales, teóricos, científicos, técnicos, y metodológicos establecidos por la Universidad de las Fuerzas Armadas ESPE, respetando los derechos intelectuales de terceros y referenciando las citas bibliográficas.

Sangolquí, 18 de agosto del 2021

Firma:



Cobo Jaramillo, Nikolás Mateo

C.C.: 1723503379

DEPARTAMENTO DE ELECTRICA, ELECTRÓNICA Y TELECOMUNICACIONES

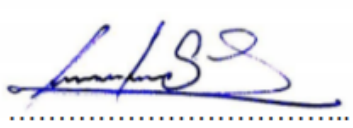
CARRERA DE INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES

AUTORIZACIÓN DE PUBLICACIÓN

Yo, **Cobo Jaramillo, Nikolás Mateo** con cédula de ciudadanía N°1723503379, autorizo a la Universidad de las Fuerzas Armadas ESPE publicar el trabajo de titulación: "**Diseño de una arquitectura para el control telemático de sistemas ciber físicos basada en microservicios y con tolerancia a fallos",** en el Repositorio Institucional, cuyo contenido, ideas y criterios son de mi responsabilidad.

Sangolquí, 18 de agosto del 2021

Firma:



Cobo Jaramillo, Nikolás Mateo

C.C.: 1723503379

Dedicatoria

En primer lugar, este trabajo se lo dedico a Dios, la luz de mi vida, sin su presencia e infinito amor, esto no sería posible.

A mi madre Luisa, por su paciencia, su sabiduría, su amor incondicional, y por ser mi mayor apoyo siempre.

A mi padre Fernando, por su ejemplo de sacrificio y fortaleza frente a las adversidades.

A mi hermana Camila, por su alegría, sus bromas y su compañía que llenan mis días de felicidad.

A mis abuelitos por siempre apoyarme, motivarme y acompañarme desde cualquier lugar y en todo momento.

Al resto de mi familia y a mis amigos que siempre han creído en mí.

Cobo Jaramillo, Nikolás Mateo.

Agradecimiento

Cuando escogí seguir esta carrera, nunca imagine la gran cantidad de dificultades y obstáculos que tendría que sobrepasar, pero tampoco imagine la cantidad de alegrías y buenos momentos que viviría y que me acompañaran el resto de mi vida. En el recorrido por este camino he aprendido demasiado, pero sobre todo he conocido personas increíbles que sin duda han marcado mi vida.

En primer lugar, quiero agradecer a Dios, las palabras nunca me alcanzarán para agradecer por la fuerza y determinación que me ha dado, solo él conoce lo que me ha costado llegar hasta aquí, por eso con palabras simples solo me queda decir "*Gracias Dios, por tanto, Gracias por absolutamente todo".*

Gracias a mi madre Luisa y a mi hermana Camila por tantos años de amor, aprendizaje y apoyo que me han ayudado a enfrentar cualquier adversidad y nunca darme por vencido.

Gracias a mi padre Fernando que siempre me ha motivado con sus palabras y me ha demostrado que en la vida a pesar de las circunstancias siempre hay que levantarse y seguir luchando.

Gracias a mis abuelitos por su inmenso amor y apoyo, sin ellos esto no sería posible.

Gracias al resto de mi familia que han estado siempre presentes.

Gracias a todos mis amigos que han hecho de este camino mucho más divertido y llevadero, sin las experiencias vividas y su apoyo sincero, este trabajo no sería lo mismo.

Finalmente, un agradecimiento especial a la Universidad de las Fuerzas Armadas- ESPE y a sus docentes, pero sobre todo a mi tutor, el Ing. Darwin Alulema que, con sus conocimientos, enseñanzas y su guía, ha sido el principal gestor en el desarrollo del presente proyecto y ha permitido la culminación exitosa del mismo.

Cobo Jaramillo, Nikolás Mateo.

Índice General

Índice de Tablas

Índice de Figuras

Resumen

El desarrollo tecnológico en los últimos años ha crecido de manera exponencial, enfocándose en la Web de las cosas (WoT), convirtiendo a los servicios web en un pilar fundamental de la conectividad de todas las aplicaciones actuales. Esto ha generado la necesidad de brindar servicios, robustos, escalables, pero sobre todo tolerantes a fallos, que estén disponibles todo el tiempo a pesar de presentarse un error. Además, los nuevos retos tecnológicos requieren la convergencia del entorno físico, virtual y los protocolos de comunicación para cubrir las necesidades emergentes de la sociedad en muchos campos de la industria y la ciencia. Con esta motivación, se propone una arquitectura basada en la WoT y en microservicios con tolerancia a fallos para el control telemático de un robot. Esta arquitectura consta de tres capas principales, la primera es el frontend, una aplicación de control desarrollada en HTML, CSS y JavaScript que puede ser utilizada en cualquier dispositivo, la segunda es el backend, una arquitectura orientada a microservicios y tolerante a fallos desarrollada con JAVA, Spring, librerías de Netflix y que ocupa bases de datos SQL, y la tercera y última capa es un sistema ciber físico (CPS) y más precisamente un robot que tiene una Raspberry Pi como controlador, en donde se encuentra desplegado un servidor desarrollado en Python y Flask. El despliegue para cumplir con el control telemático se lo realiza en internet con la ayuda de un VPS y posteriormente se evalúa funcionamiento, rendimiento y usabilidad, obteniendo muy buenos resultados.

PALABRAS CLAVE:

- **ARUITECTURA ORIENTADA A SERVICIOS.**
- **SISTEMAS CIBERFISICOS**
- **CONTROL TELEMÁTICO**
- **TOLERANCIA A FALLOS**
- **RESTful**

Abstract

Technological development in recent years has grown exponentially, focusing on the Web of Things (WoT), making web services a fundamental pillar of the connectivity of all current applications. This has generated the need to provide robust, scalable, but above all fault-tolerant services that are available at all times despite the occurrence of an error. In addition, new technological challenges require the convergence of the physical and virtual environment and communication protocols to meet the emerging needs of society in many fields of industry and science. With this motivation, an architecture based on WoT and fault-tolerant microservices is proposed for the telematic control of a robot. This architecture consists of three main layers, the first is the frontend, a control application developed in HTML, CSS and JavaScript that can be used on any device, the second is the backend, a microservices oriented and fault tolerant architecture developed with JAVA, Spring, Netflix libraries and occupying SQL databases, and the third and final layer is a cyber physical system (CPS) and more precisely a robot that has a Raspberry Pi as controller, where a server developed in Python and Flask is deployed. The deployment to meet the telematic control is done on the Internet with the help of a VPS and then evaluated operation, performance and usability, obtaining very good results.

KEYWORDS:

- **SERVICE-ORIENTED ARCHITECTURE.**
- **CYBER-PHYSICAL SYSTEMS**
- **TELEMATIC CONTROL**
- **FAULT TOLERANCE**
- **RESTful**

Capítulo I

Marco Metodológico

Antecedentes

El creciente desarrollo tecnológico en el área de las Telecomunicaciones y las Tecnologías de la Información (TIC), ha hecho que se adopten sistemas y arquitecturas avanzadas orientadas hacia el Internet de las Cosas (IoT, del inglés Internet of Things), que en los últimos años ha mejorado, colocando a los servicios y las aplicaciones en la nube en los extremos de una red (Taherizadeh, Stankovski, & Grobelnik, 2018). IoT además se basa en la interconexión de diversos elementos que, con la utilización de sensores, actuadores y protocolos de comunicación adecuados, permiten al usuario interactuar con el entorno.

Debido a la complejidad, rápida evolución, y gran volumen de los actuales servicios y aplicaciones en la nube, el desarrollo de software y los sistemas monolíticos comenzaron a presentar limitaciones al momento de realizar cambios o actualizaciones en su capa de aplicación, siendo demasiado lento e ineficiente. Como respuesta a estos problemas surgió la arquitectura orientada a servicios (SOA, Service Oriented Architecture) y más específicamente la arquitectura de microservicios, la cual permite la creación de sistemas altamente escalables, brindando una forma bien definida de exposición e invocación de servicios (Krivic, Skocir, Kusek, & Jezic, 2018). Otra de las ventajas de la arquitectura de microservicios es que en sus frameworks de desarrollo es posible utilizar librerías que mejoran compensaciones específicas de calidad de servicio (QoS) que deben tenerse en cuenta, especialmente en situaciones en las que las cargas de trabajo varían con el tiempo o cuando los dispositivos de IoT están cambiando dinámicamente su posición geográfica u operación (Taherizadeh, Stankovski, & Grobelnik, 2018; Santana, Alencar, & Prazeres, 2018). Entre estas se

encuentra *Hystrix*, una librería desarrollada y ofrecida por Netflix, diseñada para aislar puntos de acceso a sistemas remotos, servicios y librerías de terceros, implementando el patrón *Circuit Breaker* que permite gestionar diversos aspectos tales como: timeouts, estadísticas de éxito y fallo, semáforos, lógica de gestión de error, lógica de latencia y de esto modo deteniendo fallos en cascada, mejorando la resiliencia en sistemas complejos distribuidos donde la probabilidad de fallo es inevitable (Rodríguez, 2015; Crespo, 2016).

En la implementación de un servicio web (WS, del inglés Web Service) existen varias opciones en cuanto a su estilo y tecnologías de desarrollo dependiendo de las necesidades, como por ejemplo los “Big” WS (WS- *), que son flexibles y aportan una calidad de servicio avanzada, requisitos que ocurren comúnmente en la informática empresarial, o los RESTful WS, más adecuados para escenarios de integración no tan complejos que pueden ser utilizados para una solución específica, simplificando su interoperabilidad con otros servicios (Pautasso, Zimmermann, & Leymann, 2008). RESTful WS o RESTful API son programas basados en REST, usado para describir una interfaz para conectar varios sistemas basados en el protocolo HTTP (uno de los protocolos más antiguos) y nos sirve para obtener y generar datos y operaciones, devolviendo esos datos en formatos muy específicos, como XML y JSON, siendo este último el más utilizado en la actualidad. (Redacción TicJob, 2019; Rosa Moncayo, 2018).

La interconexión del “cosas inteligentes” del mundo físico (p. Ej., robots autónomos, RFID, redes de sensores) con el mundo virtual, es el objetivo de muchas investigaciones actualmente, trayendo consigo propuestas como Web of Things (WoT) que se basa en principios RESTful y busca convertir a los datos y funciones del mundo real en una parte integral de la Web, contribuyendo a la escalabilidad y modularidad de

la Web tradicional. Como resultado las cosas inteligentes se vuelven más fáciles de construir. Los lenguajes web populares (p. Ej., HTML, Python, JavaScript, PHP) se pueden utilizar para crear fácilmente aplicaciones que involucran cosas inteligentes y los usuarios pueden aprovechar mecanismos web bien conocidos (p. Ej., Navegación, búsqueda, marcadores, almacenamiento en caché, enlaces) para interactuar y compartir estos dispositivos (Guinard, Wilde, & Trifa, 2010).

Con la implementación de arquitecturas orientadas a Web Of Things (WoT) y RESTful WS, facilitando el intercambio de información entre dispositivos y aplicaciones que forman una pequeña parte del Internet y la Web, se han desarrollado sistemas que son capaces de interactuar, comunicarse, maximizar la seguridad, la protección, la comodidad y el ahorro de energía. Además, con la utilización de un microcontrolador como Arduino o un miniordenador como Raspberry PI, se puede integrar múltiples sensores y actuadores dentro de un nodo fijo o un robot móvil, lo que facilita el monitoreo de un entorno y garantiza la confiabilidad de los datos obtenidos (Vujović & Maksimović, 2015).

Otro concepto que actualmente ha tomado realce, es el de los Sistemas Ciber físicos (CPS), que son la integración del software con sus capacidades de computación, almacenamiento y comunicación junto con el hardware y sus capacidades de seguimiento y/o control de objetos en el mundo físico. El objetivo de los Sistemas Ciber físicos es percibir y comprender los cambios en el entorno físico que los rodea, analizar estos datos y tomar decisiones rápidas e inteligentes para responder a dichos cambios de forma autónoma. Este mecanismo junto a los conceptos de WoT y SOA, permite actuar y tomar decisiones en tiempo real (Shi, Wan, Yan, & Suo, 2009).

El presente proyecto se enfocó en desarrollar una arquitectura para el control telemático de sistemas ciber físicos e implementar un caso de prueba para el control de

robots móviles. La propuesta se basa en microservicios RESTful bajo una arquitectura SOA, la cual abarcará Front-end y Back-end. En el Front-end se desarrolló una interfaz híbrida para aplicaciones web, y móviles, donde el usuario pueda realizar diversas acciones de control mediante la planificación de rutas para un robot móvil, estas acciones son procesadas en el Back-end en formato JSON, donde se coordina la interacción del robot móvil que dispone de múltiples sensores y actuadores que están conectados a un controlador, el cual a su vez está conectado a internet utilizando el protocolo de comunicación HTTP y por medio de los métodos (GET, POST, PUT, DELETE) se envían las alertas al servicio web a través de mensajes JSON, donde de presentarse un error, se activan los mecanismos de control establecidos con Hystrix y el patrón Circuit Breaker inicia un proceso de “fallback”, el cual permite notificar al usuario en la interfaz que algo inesperado está sucediendo, permitiendo tomar una decisión y solucionar el problema, consiguiendo así un sistema tolerante a fallos.

Justificación e Importancia

Los proyectos e iniciativas desarrollados bajo el término general “Internet de las Cosas (IoT)” se han centrado principalmente en establecer conectividad en una variedad de entornos de red, haciendo que dispositivos inteligentes funcionen de manera desatendida. Sin embargo, un paso prometedor es construir modelos de interacción escalables sobre esta conectividad de red básica y, por lo tanto, centrarse en la capa de aplicación, a esto se le conoce como “Web of Things (WoT)”, bajo este concepto, las cosas inteligentes y sus servicios están completamente integrados en la Web mediante la reutilización y adaptación de tecnologías y patrones comúnmente utilizados para el contenido Web tradicional, es decir una capa de arquitectura Web facilita el intercambio de información entre dispositivos y sistemas, generando “Las

Cosas Web” (Guinard, Wilde, & Trifa, 2010). La utilización de Web of Things (WoT) y las prácticas basadas en los principios RESTful y SOA, han contribuido al éxito popular, la escalabilidad, la capacidad de evolución de la Web, y ha ayudado al desarrollo de varios prototipos y aplicaciones, por ejemplo, la conexión de nodos de sensores ambientales, sistemas de monitoreo de energía, objetos etiquetados con RFID, y robots inteligentes a la Web; también ha ayudado a la generación de aplicaciones para hubs domésticos y plataformas en la nube, donde algunas se ejecutan en segundo plano sin una interfaz hombre-máquina, como un registrador que almacena los datos de un sensor en una tarjeta de memoria y los carga en la nube, u otras aplicaciones que se ejecutan en primer plano y requieren una interfaz hombre-máquina; y finalmente ha demostrado que las cosas inteligentes habilitadas para la Web se pueden usar en aplicaciones ligeras y de propósito específico, llamadas "Mashups físicos". El creciente desarrollo de WoT y la creciente demanda mundial por la interconexión de elementos tecnológicos de la vida diaria, ha hecho que se discutan y traten de solventar los desafíos restantes hacia lo que se conocería como “World Wide Web of Things” global (Raggett, 2016).

La proliferación de sistemas de cómputo y sistemas Web, ha hecho que sean usados cada vez más en diversos ámbitos de la industria, sin embargo, no se ha podido generalizar su uso debido a problemas como la latencia, perdida de paquetes, entre otros. No son adecuados para aplicaciones críticas. Existen aplicaciones y sistemas que, si presentan un error en su funcionamiento resultaría catastrófico y podría causar importantes perjuicios. Ejemplos de estas aplicaciones son los procesos industriales delicados controlados por computador, como una central nuclear, los computadores que gobiernan aviones, satélites artificiales y naves espaciales, o sistemas como los de un banco, en todos estos casos si se pierde la información, se cae un servidor o existe algún problema que no permita enviar los datos adecuadamente, generaría un gran

problema, ya que se pueden perder equipos de millones de dólares, provocar una catástrofe ambiental y atentar contra la vida o bienestar de muchas personas (Wikipedia, 2020).

Como se mencionó, en la actualidad existen aplicaciones que resultan lo suficientemente críticas como para protegerlas contra potenciales fallo y una posible solución es usar técnicas para conseguir que los sistemas continúen funcionando correctamente a pesar de fallos en su hardware o errores de software, este tipo de sistemas se denominan sistemas tolerantes a fallos. Existen grandes empresas como Netflix, cuyo servicio principal es la distribución de contenido audiovisual a través de una plataforma en línea, utilizando como ejemplo el servicio que ofrecen, se puede afirmar que los sistemas tolerantes a fallos, son de vital importancia, ya que si una única dependencia de la API falla en un gran volumen con mayor latencia, puede saturar rápidamente todos los subprocesos de solicitud disponibles y eliminar todo el proceso, desde este paradigma es que Netflix puso como premisa que “La tolerancia a fallas es un requisito, no una característica” (Christensen, 2012; Cobos Domínguez, 2016). Bajo este concepto Netflix desarrollo Hystrix, una herramienta que utiliza el patrón de Circuit Breaker y es encargada de mejorar la fiabilidad global del sistema, ya que, aísla los puntos de acceso de los microservicios, impidiendo así los fallos en cascada a través de los diferentes componentes de la aplicación, proporcionando alternativas de “fallback”, gestionando timeouts, pools de hilos. Hystrix encapsula las peticiones a sistemas “externos” para gestionar dichos aspectos. Así por ejemplo si una petición a un servicio alcanza un cierto límite (20 fallos en 5 segundos por defecto), Hystrix abrirá el circuito de forma que no se realizarán más peticiones a dicho servicio, lo que impedirá la propagación de los errores en cascada (Rodríguez, 2015).

La Arquitectura Orientada a Servicios (SOA) no es un concepto nuevo, define la utilización de servicios para dar soporte a ciertos requisitos del negocio, se ocupa del diseño, desarrollo de sistemas distribuidos y es un potente aliado a la hora de llevar a cabo la gestión de datos en la nube (Eassa, 2018). Pero se debe cambiar la manera en la que el software es percibido, concebido y diseñado siendo lo nuevo en este caso pasar del paradigma de una arquitectura monolítica a una arquitectura basada en microservicios, para lo cual REST facilita los procesos de interoperabilidad, robustez y escalabilidad, lo cual es empleado en los sistemas actuales como Amazon AWS, Google Cloud o Netflix (Dragoni, Lanese, & Mazzara, 2018).

La idea del software como un servicio se planteó para que una entidad de software interactúe con otras a través de comunicaciones de paso de mensajes utilizando formatos y protocolos de datos estándar (por ejemplo, XML, SOAP y HTTP) e interfaces bien definidas. Los microservicios son un paso más en este camino, enfatizando el uso de “pequeños servicios”, y redireccionando las técnicas orientadas a servicios, al diseño, desarrollo e implementación de sistemas (Dragoni, Lanese, & Mazzara, 2018). Al momento de invocar los microservicios, existen dos maneras: coreografía y orquestación. La coreografía es cuando cada servicio Web implicado "conoce" exactamente cuándo ejecutar sus operaciones y con quién debe interactuar. La orquestación tiene un proceso central (que puede ser otro servicio Web) y lleva el control de los servicios Web implicados en la realización de una tarea y coordina su ejecución para cada operación. La orquestación es la utilizada en sistemas tolerantes a fallos, ya que los microservicios no necesitan conocer que están implicados en un proceso de composición y que forman parte de un proceso de nivel más alto. Solamente el coordinador central de la orquestación es "consciente" de la meta a conseguir, por lo que se centraliza mediante definiciones explícitas de operaciones y del orden en el que

se deben invocar los servicios web (Departamento de la Ciencia de Computación e Inteligencia Artficial., 2014; Wikipedia, 2020).

De los protocolos de datos estándar, REST se apoya en HTTP (HyperText Transfer Protocol). un sencillo protocolo cliente-servidor que articula los intercambios de información entre los clientes Web y los servidores HTTP, se basa en sencillas operaciones de solicitud/respuesta. Un cliente establece una conexión con un servidor y envía un mensaje con los datos de la solicitud. El servidor responde con un mensaje similar, que contiene el estado de la operación y su posible resultado (NEO, 2014). Para el intercambio de mensajes y de datos existen formatos como XML o JSON, pero este último resulta ser el más eficiente, ya qué es más rápido en cualquier navegador, más fácil de leer, más ligero (bytes) en las transmisiones y se parsea más rápido, además se emplea habitualmente en entornos donde el tamaño del flujo de datos entre cliente y servidor es de vital importancia, haciéndolo la opción más viable en un sistema tolerante a fallos que no maneja grandes cantidades de datos (Wikipedia, 2020).

Existe una serie de puntos que caracterizan al WoT, entre ellos está la interconectividad que permite el acceso a la información y la comunicación mundial, y de la mano viene la escalabilidad un concepto que tiene que ver con el número de dispositivos interconectados para generar una red de información (Wikipedia, 2020). La utilización de microcontroladores o microordenadores facilitan la implementación de un servicio Web como parte de WoT, por lo qué utilizando dispositivos como Raspberry Pi: una pequeña computadora económica, totalmente personalizable y programable con soporte para una gran cantidad de periféricos y comunicación de red, se puede mejorar el monitoreo y determinación del entorno en la que se encuentre un pequeño robot móvil, creando y desarrollando así un sistema completo, partiendo de cero y

apoyándose en las ventajas que ofrece Raspberry Pi: flexibilidad y amplia posibilidad de uso (Vujović & Maksimović, 2015).

En este sentido, con este estudio se desarrolla una arquitectura para controlar un sistema ciber físico, la cual es basada en microservicios y tolerante a fallos, en la cual convergen todas las tecnologías expuestas y permite la interoperabilidad de aplicaciones. En este caso los elementos de una red WoT que lo conforman, como el caso del Front-End con su interfaz de usuario y del Back-End con sus elementos como el servicio web implementado junto a la herramienta Netflix Hystrix y con un proceso de orquestación de microservicios, y finalmente el robot móvil conectado al Back-end mediante un controlador, que dispondrá de diversos sensores y actuadores, que permiten la interacción del mismo con el entorno en el que se encuentre.

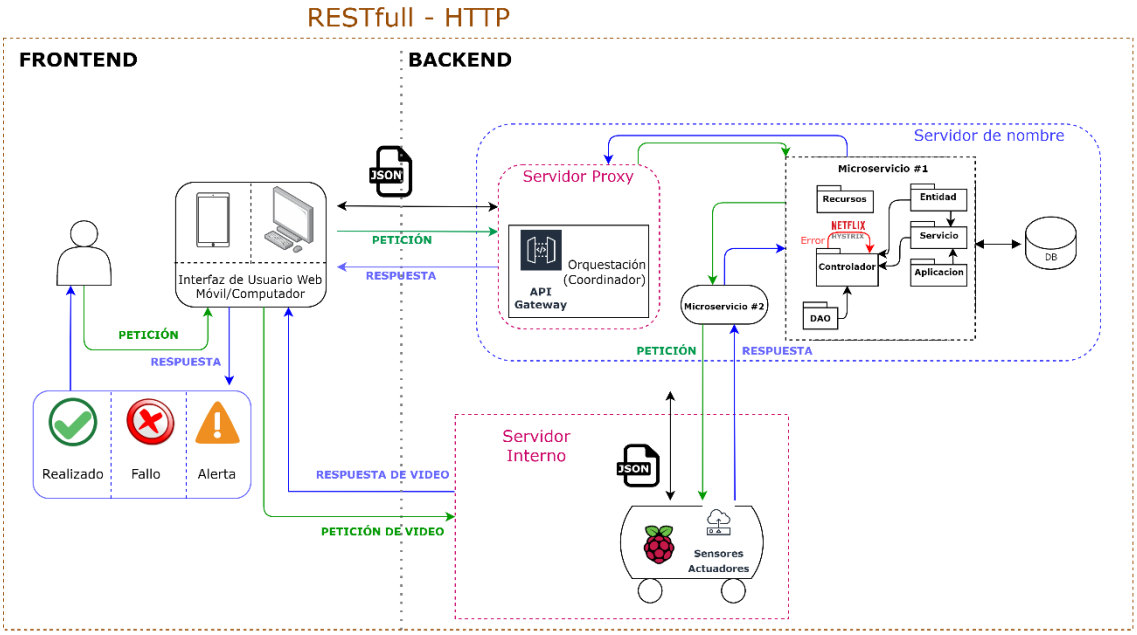
Alcance del Proyecto

La arquitectura desarrollada se puede apreciar a breves rasgos en la figura 1, es RESTful en su totalidad, usando REST que se apoya en el protocolo de comunicación HTTP, en su Front-End cuenta con una interfaz de usuario adaptable para aplicaciones móviles y aplicaciones web, donde el usuario puede ingresar comandos de control de movimiento para el robot móvil, estos comandos son enviados al Back-End en formato de mensaje JSON, en el Back-End se cuenta con una arquitectura de microservicios con un enfoque de orquestación, es decir, hay un proceso central que es el encargado de coordinar el resto de microservicios con la finalidad de obtener un objetivo común, cada microservicio representa un comando de movimiento ingresado en la interfaz de usuario, por lo que el orquestador los va invocando en el orden que sean ingresados para posteriormente enviarlos al robot móvil en formato de mensaje JSON, el robot móvil dispone de múltiples sensores que monitorean el entorno y detectan si existe

algún obstáculo en el camino, también dispone de motores (actuadores) en cada uno de sus ejes lo que facilita su movimiento en múltiples direcciones; tanto los sensores como los actuadores estarán conectados a un controlador, que gestiona su funcionamiento y está conectada al servicio web mediante internet y envía al mismo los mensajes de éxito como de fallo en formato JSON.

Figura 1

Diagrama del sistema implementado

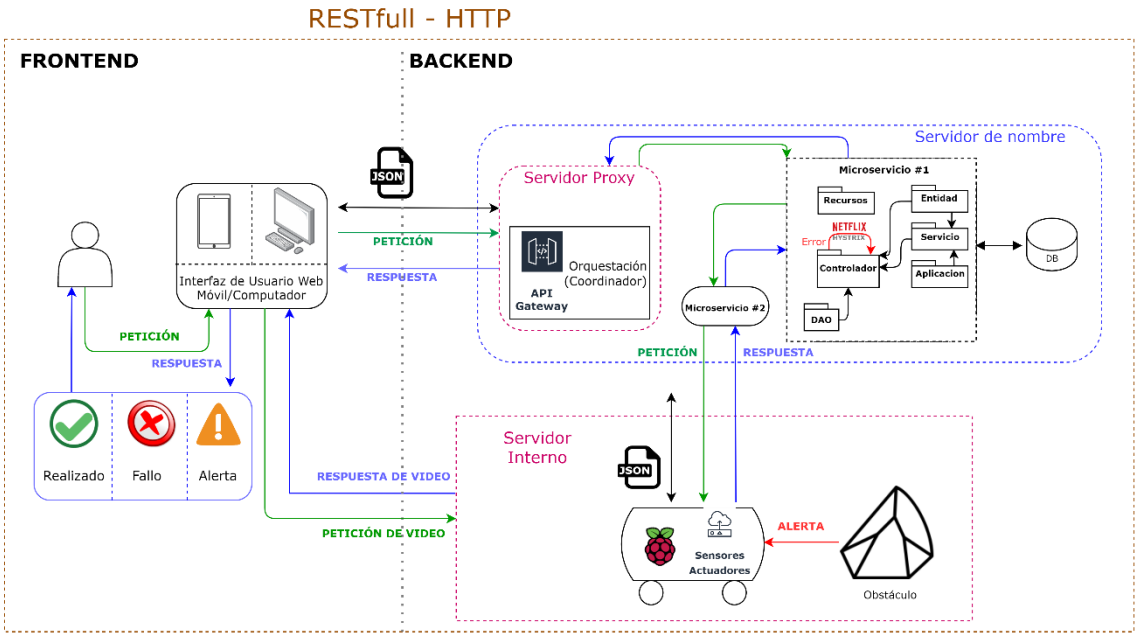


La arquitectura propuesta y descrita anteriormente es tolerante a fallos, es decir de presentarse un obstáculo en el camino del robot móvil como se puede apreciar en la figura 2, el sistema entra en un proceso para solventar este problema. En el caso de qué cualquiera de los sensores detecte algún obstáculo en el camino, el controlador envía una alerta al servicio web, entonces la librería de Netflix Hystrix con su patrón Circuit Breaker, abre el circuito para prevenir más errores, evitando que se invoque nuevamente el microservicio al que está relacionado el comando ingresado por el

usuario y entra en un proceso de “fallback”, el cual consiste en enviar una notificación de error a la interfaz de usuario donde se puede apreciar que el último comando ingresado presenta un problema y entonces el usuario puede decidir un comando diferente para solventar el problema, si el comando seleccionado ya no presenta problemas, entonces se envía nuevamente este mensaje al servicio web y posteriormente al robot móvil para que pueda continuar con su ruta.

Figura 2

Escenario del sistema donde se encuentre un obstáculo



Objetivos

General

Diseñar e implementar una arquitectura para el control telemático de sistemas ciber físicos bajo una arquitectura SOA de microservicios y tolerante a fallos, e implementar un caso de prueba para el control de robots móviles.

Específicos

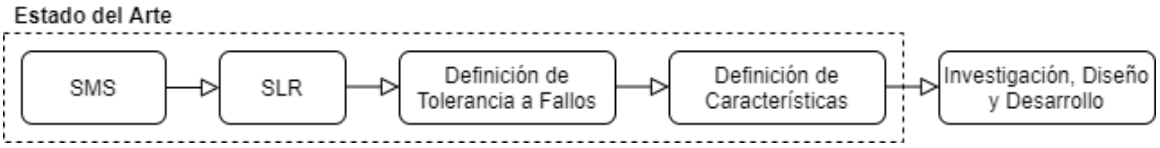
- Realizar un estudio del arte y análisis de las plataformas de servicios en la nube que estén basados en REST y una arquitectura de microservicios.
- Diseñar e implementar una arquitectura orientada a microservicios tolerante a fallos usando la herramienta Netflix Hystrix.
- Implementar una serie de servicios web RESTful para interconectar todos los elementos mediante el protocolo de comunicación HTTP.
- Diseñar e implementar una interfaz adaptable a aplicaciones móviles y aplicaciones web que permita la intercomunicación con el usuario y notifique los posibles errores de la interacción del sistema ciber físico.
- Realizar pruebas, análisis y validación del sistema desarrollado, de acuerdo a la interacción con el caso de un robot móvil con el entorno y las decisiones del usuario para solucionar un problema en caso de haberlo.

Estado del Arte

Esta sección está dividida en dos subsecciones. En la primera se define la metodología utilizada para la búsqueda del estado del arte, y para sintetizar los resultados se utilizó un *Mapeo Sistemático de la Literatura* (SMS, Systematic Mapping Study) y Revisión Sistemático de la Literatura (SLR, Systematic Literature Review). Por otro lado, la segunda sección propone el concepto de tolerancia a fallos para este tipo de sistemas y define las características necesarias que debe tener una arquitectura de microservicios, junto con un sistema ciber físico para que el producto final sea interoperable, robusto y escalable según lo obtenido en la búsqueda del estado del arte. Finalmente, se expone las conclusiones que se pudieron haber extraído de este capítulo. La figura 3 muestra el proceso resumido que se seguirá para desarrollar el Estado del Arte.

Figura 3

Proceso para el desarrollo del Estado del Arte



Nota: Inspirado en (Moguel Márquez, 2018)

Síntesis de evidencia disponible

Mapeo Sistemático de la Literatura

En el presente documento ya se ha justificado el uso de arquitecturas basada en microservicios y de la tolerancia a fallos, además de su notable crecimiento y su utilización en la mayoría de software actuales. El sector del desarrollo de aplicaciones ha visto un crecimiento exponencial y continúan apareciendo empresas especializadas en esto como su núcleo de negocio. Sin embargo, estas empresas se enfocan en su gran mayoría en el diseño de software, y dejan un poco de lado el hardware, sin tener en cuenta que los sistemas ciber físicos en los siguientes años tendrán un impacto importante en entornos de WoT.

En el desarrollo de software para el control telemático de sistemas ciber físicos, existen algunas empresas e investigadores que están desarrollando sus propios productos, sin embargo, son orientados a problemas específicos y por lo tanto estas soluciones son poco replicables. Al realizar un análisis y una revisión de los principales estudios en cuanto al desarrollo de aplicaciones para el control telemático de sistemas ciber físicos, se podría decir que se están abordando las soluciones de una manera dispersa, sin seguir un diseño estándar y orientadas a problemas muy concretos.

Por esto, y para tener un mayor y mejor conocimiento de lo que se está desarrollando en cuanto a aplicaciones para controlar telemáticamente sistemas ciber físicos y que sean tolerantes a fallos, se realizó un Mapeo Sistemático de la Literatura (Kitchenham & Charters, 2007) (SMS en adelante), en especial en las tecnologías, herramientas de desarrollo y los complementos necesarios (frameworks, base de datos, host, etc.).

En este punto el principal objetivo del SMS fue proporcionar un conocimiento amplio y sustentado, con el que se pudo desarrollar el trabajo de una manera óptima y adecuada, así como también aporta a futuras actividades de investigación.

Objetivos. Existen estudios y aplicaciones basadas en una arquitectura de microservicios, pero están más orientadas a la computación en la nube o a intenciones de sistemas IoT, por lo que sus soluciones no pueden definirse como replicables, ya que abordan problemas concretos (Guinard, Wilde, & Trifa, 2010; Krivic, Skocir, Kusek, & Jezic, 2018; Dragoni, Lanese, & Mazzara, 2018). Es verdad que, gracias a estos estudios e innovaciones, las arquitecturas de este tipo han logrado ser mucho más utilizadas, pero la mayoría de sistemas, aún no pueden definirse como ciber físicos y lo más importante, tolerantes a fallos, ya que pueden presentarse problemas que ocasionarían fallos en cascada y harían colapsar todo un servicio sin la opción de que el usuario pueda evitar dicho inconveniente.

La importancia del SMS presentado es que sirvió como un cimiento claro para poder conocer las líneas de estudio e investigación en las que actualmente se está trabajando a nivel mundial, saber si dichos trabajos pueden ser replicables e utilizados en la actualidad sobre todo en un entorno WoT con un sistema ciber físico como el propuesto. También sirvió para saber si existen posibles innovaciones y nichos sobre

los qué no se esté trabajando, pero que sean importantes para el desarrollo de un sistema interoperable, robusto y escalable.

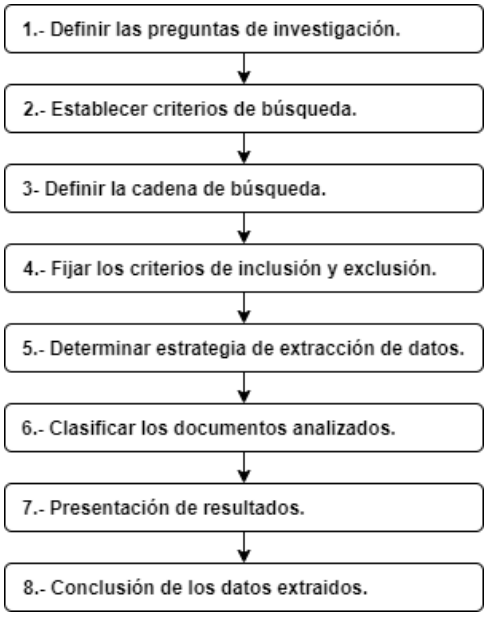
Una ventaja del SMS es la amplitud de las preguntas que pueden ser utilizadas y que son necesarias para conocer las publicaciones de vanguardia en el sector de arquitecturas de microservicios en entornos WoT con sistemas ciber físicos. Los resultados que brindo este estudio fueron el punto de partida para las revisiones sistemáticas posteriores.

Método de investigación. La metodología definida por un SMS está enfocada en clasificar, categorizar y analizar temas amplios a fin de obtener una perspectiva del conocimiento existente en estas áreas (Kitchenham & Charters, 2007; Petersen, Feldt, Mujtaba, & Mattsson, 2008). A partir de responder las preguntas que se plantean para la investigación, se puede categorizar la información generada, ahorrando esfuerzo y tiempo en la investigación. Para lograr estos objetivos, el SMS debe ser riguroso, completo y de calidad. La importancia de usar este instrumento es para identificar temas con suficientes estudios primarios que llevan a revisiones sistemáticas y a nuevos estudios (Kitchenham, Budgen, & Brereton, 2011)

El SMS fue adaptado y se aplicó herramientas y técnicas que se debía tener en cuenta para el desarrollo del proyecto, ya que involucra tanto desarrollo de software como utilización de hardware específico. Debido a que es un proceso exhaustivo y cada uno de los pasos está bien definido, en la figura 4 se presenta el proceso que se siguió para el SMS, teniendo en cuenta algunas recomendaciones. (Bailey, y otros, 2007)

Figura 4

Proceso seguido para el SMS



Nota: Inspirado en (Moguel Márquez, 2018).

Definir las preguntas de investigación. El propósito y motivación del presente SMS es examinar, las tecnologías y herramientas que se usan actualmente en el desarrollo de arquitecturas orientadas a microservicios y en especial en entornos WoT, donde estén involucrados sistemas ciber físicos que deben ser tolerantes a fallos, para evitar grandes problemas y sobre todo la inconformidad del usuario, para esto, se plantearon las siguientes preguntas:

Tabla 1

Preguntas para investigación del SMS

Pregunta de investigación	Motivación
Q1: ¿Qué técnicas y/o tecnologías existen para el desarrollo de arquitecturas orientadas a microservicios?	Conocer qué tecnologías, frameworks de desarrollo y pluggins adicionales existen, y en qué estado de madurez se encuentran.
Q2: ¿Qué técnicas y/o tecnologías existen para el desarrollo de aplicaciones tanto para el back-end como para el front-end?	Conocer los entornos de desarrollo más utilizados para las distintas plataformas que existen.
Q3: ¿Qué herramientas y/o dispositivos se utilizan para la implementación de sistemas ciber físicos orientados a entornos WoT?	Conocer el hardware existente que puede ser utilizado para un entorno WoT y que pueda ser controlado tanto en un servidor local o un servidor en la nube.
Q4: ¿Qué técnicas y/o tecnologías existen para la adquisición, procesamiento y almacenamiento de datos de sistemas ciber físicos en entornos WoT?	Conocer las tecnologías existentes para la adquisición, captura y procesamiento de dichos datos, ya sea en un servidor local o en un servidor en la nube.

Establecer los criterios de búsqueda primarios y secundarios. Los motores de búsqueda y bibliotecas digitales seleccionadas para realizar el mapeo de documentos, fueron IEEE Xplore, Science Direct, Springer Link y MDPI. La importancia de estas bases de datos es su amplia cobertura de publicaciones en ámbitos como la Ingeniería Electrónica, las Ciencias de la computación, entre otras, y además tienen indexados algunos catálogos de temáticas más específicas.

Debido a que se trata de un reconocimiento inicial de las técnicas, tecnologías y herramientas utilizadas actualmente, no se descarta ningún tipo de artículo, ya que el propósito es conocer el estado de desarrollo y de utilización en el que se encuentran las tecnologías existentes, independientemente de su origen.

Definir las cadenas de búsqueda. En un principio se desconoce el volumen de información con el que se va a tratar, por lo que es importante realizar una búsqueda exploratoria para sondear la cantidad de recursos disponibles. Se definieron algunas cadenas de búsqueda que sirvieron en la exploración, y a través de eso se estableció su relevancia para ser consideradas como fundamentales para el resultado final de esta tesis.

Para las cadenas de búsqueda de este SMS, se identificaron los términos fundamentales que se reflejan en las preguntas de investigación que se definieron anteriormente, y se presentan en la tabla 2, es importante mencionar que la búsqueda se realizó en el idioma inglés, ya qué es el estándar utilizado por las bibliotecas digitales.

Tabla 2

Términos definidos para las cadenas de búsqueda del SMS.

Término principal	Términos alternativos
Microservices oriented architectures. [Q1]	"Service oriented architecture" **OR** SOA **OR** "Microservice architecture".
App development. [Q2]	"Software development" **OR** "Mobile app development" **OR** "Web development" **OR** "Cross-Platform app development".
Cyber-physical system. [Q3]	CPS.
WoT [Q3, Q4]	"Web of Things" **OR** IoT.
Data acquisition and processing. [Q4]	DAQ **OR** "measurement".

A partir de estos términos y sus variantes, la cadena de búsqueda principal, fue la siguiente:

```
("Microservices oriented architectures" OR "Service oriented
architecture" OR SOA OR "Microservice architecture".)
AND
```

```
("App development" OR "Software development" OR "Mobile app
development" OR "Web development" OR "Cross-Platform app
development".)
AND
("Cyber-physical system" OR CPS)
AND
("Web of things" OR WoT OR IoT)
AND
("Data acquisition and processing" OR DAQ OR Measurement)
```

Para esta cadena de búsqueda se obtuvieron los siguientes resultados en las diferentes bibliotecas digitales:

- **IEEE Xplore:** 1 resultado.
- **Springer Link:** 6 resultados.
- **ScienceDirect:** 1 resultado.
- **MDPI:** 1 resultado.

Como se puede ver, los resultados obtenidos son insuficientes para realizar un mapeo y un estudio con garantías, que es lo que se busca con este SMS. Por lo que se decidió realizar tres búsquedas en paralelo con distintas cadenas de búsqueda:

1. Cadena de búsqueda 1:

```
("Microservices oriented architectures" OR "Service oriented
architecture" OR "SOA" OR "Microservice architecture".)
AND
("App development" OR "Software development" OR "Mobile app
development" OR "Web development" OR "Cross-Platform app
development".)
```

AND

(“Cyber-physical system” OR CPS).

2. Cadena de búsqueda 2:

(“Microservices oriented architectures” OR “Service oriented architecture” OR “SOA” OR “Microservice architecture”.)

AND

(“App development” OR “Software development” OR “Mobile app development” OR “Web development” OR “Cross-Platform app development”.)

AND

(“Web of things” OR WoT OR IoT).

3. Cadena de búsqueda 3:

(“Microservices oriented architectures” OR “Service oriented architecture” OR “SOA” OR “Microservice architecture”.)

AND

(“App development” OR “Software development” OR “Mobile app development” OR “Web development” OR “Cross-Platform app development”.)

AND

(“Data acquisition and processing” OR DAQ OR Measurement).

Fijar criterios de inclusión y exclusión. Para esta búsqueda se tuvieron en cuenta todos los artículos publicados hasta diciembre de 2020, y los posteriores a enero de 2016, además se debe tener en cuenta que los resultados obtenidos de este SMS, llevaron a una revisión sistemática de la literatura.

Debido a la cantidad de documentos encontrados y para saber si es o no pertinente incluirlos en este documento, se decidió que para que las publicaciones sean incluidas deben cumplir con las siguientes condiciones:

- Artículos completos
- Pertenecientes a las ramas de “Ciencias de la computación / Computer Science” o “Ingeniería / Engineering”, de esta última, siempre y cuando tenga que ver con el área de desarrollo tecnológico.
- Cualquier tipo de publicación, ya sea artículos, conferencias, revistas, tesis, etc.
- Estudios que presenten métodos o técnicas que ayuden a la implementación estructural adecuada tanto de los sistemas ciber físicos como de las arquitecturas de microservicios.
- Publicaciones desde enero de 2016 hasta diciembre de 2020.

Por otro lado, se excluyeron los estudios que cumplieron con al menos una de estas condiciones:

- Los artículos relacionados con campos de ciencia que no involucren el desarrollo tecnológico o electrónico.
- Los artículos que traten de arquitecturas monolíticas y no de microservicios, o que solo abarquen una comparación entre ambas.

- Los escritos que no traten específicamente sobre arquitecturas de microservicios, sistemas ciber físicos o Web of Things, y que no sean de interés para el avance del trabajo o que simplemente se mencione alguna de estas temáticas, pero que no sean su foco principal de estudio.
- Artículos que solo traten a la parte de hardware como su enfoque, y que únicamente los mencionen como avance para alguna aplicación, sin abordar a profundidad su estructura y diseño.
- Los documentos que no se encuentren detallados con claridad y que únicamente se trate de una herramienta introductoria a lo que se va a estudiar.
- Artículos que no tengan una estructura adecuada, como presentaciones de PowerPoint o similares.
- Artículos duplicados o que provengan de la misma fuente.

Determinar la estrategia de extracción de datos. Es importante extraer y sintetizar los datos, debido a su volumen, esto se lo realizo con la ayuda de una ficha para extraer información de cada uno de los artículos que revisó, donde se pudo ir registrando hallazgos importantes y la bibliografía correspondiente. También se ve la relevancia del artículo, dependiendo de los campos de aplicación a los que está enfocado, con una breve justificación. Finalmente se resumió las fichas realizadas, eliminando artículos duplicados, poco relevantes y/o de poco interés para el presente trabajo.

La ficha utilizada para este SMS, es la siguiente:

Tabla 3

Ficha para extracción de información del SMS

General Information	
Title	
Launch	
Source	
Relevance	
Is this article relevant to Microservice Architecture Field?	☐ Highly Relevant ☐ Relevant ☐ Irrelevant
Is this article relevant to CPS Field?	☐ Highly Relevant ☐ Relevant ☐ Irrelevant
Application domain?	☐ Surveillance and Security ☐ Environmental Control ☐ Location and Telematic control. ☐ Commerce and entertainment ☐ Digital services (medical, economic, industrial, etc.)
Data storage	☐ SQL ☐ NoSQL
Study	
Architecture	☐ Service oriented ☐ Microservice ☐ Monolithic
Focus of the study	☐ **Software Engineering** ☐ Development (implementation, language, etc.). ☐ App development. ☐ Simulators development. ☐ Testing. ☐ **Algorithms and data structures** ☐ Fault Tolerance (Collision avoidance, Cascading failure). ☐ **Computer Architecture** ☐ Hardware (processing, digital signal, sensors, PC, microcontrollers, etc.) ☐ Software ☐ **Communication** ☐ Communication Network. ☐ Communication Protocols (MQTT, HTTP, etc.) ☐ Telematic control. (Internet, WiFi, Bluetooth).

Clasificación de los artículos. El proceso de estudio y extracción de la información, se la realizo en algunas fases:

1. En la primera fase la búsqueda fue realizada con las cadenas expuestas en el numeral 1.5.1.5, los filtros utilizados para este primer paso fueron: **Title + Keywords + Abstract,** obteniendo un total de 2608 resultados entre las distintas bibliotecas digitales, de la siguiente manera:
    - **IEEE Xplore:** 415 resultados.
    - **Springer Link:** 244 resultados.
    - **ScienceDirect:** 89 resultados.
    - **MDPI:** 1860 resultados.
2. En la segunda fase se realizó un filtrado de los artículos encontrados en la primera fase, pero tomando en cuenta los criterios expuestos en el numeral 2.1.1.6 y al igual que en el paso anterior los filtros utilizados fueron: **Title + Keywords + Abstract,** obteniendo un total de 370 resultados entre las distintas bibliotecas digitales, de la siguiente manera:
    - **IEEE Xplore:** 146 resultados.
    - **Springer Link:** 94 resultados.
    - **ScienceDirect:** 70 resultados
    - **MDPI:** 64 resultados.
3. En la tercera y última fase, se ha aplicado un filtro mucho más exhaustivo en el que a partir de los artículos obtenidos en la segunda fase, solo se consideran los que son verdaderamente de interés y que abordan muy detalladamente los métodos y técnicas que pueden ser utilizados para el presente trabajo. Los filtros utilizados fueron: **All Metadata,** es decir, **Title + Keywords + Abstract + Full Article,**

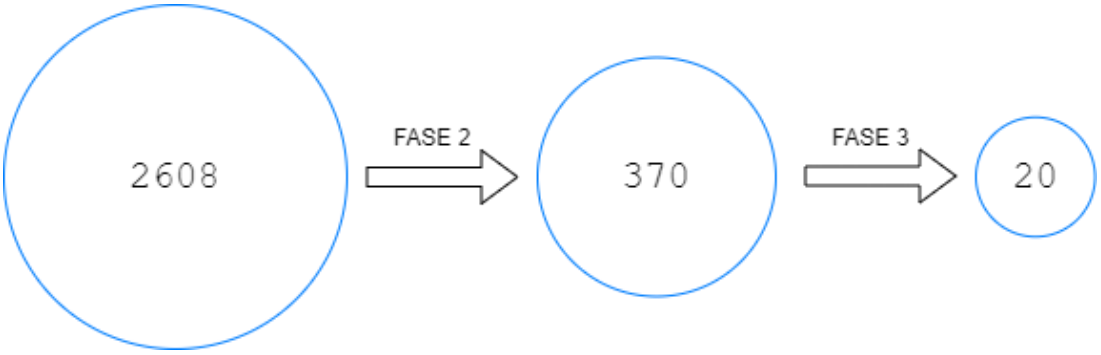
obteniendo un total de 20 resultados entre las distintas bibliotecas digitales, de la siguiente manera:

- **IEEE Xplore:** 12 resultados.
- **Springer Link:** 3 resultados.
- **ScienceDirect:** 3 resultados
- **MDPI:** 2 resultados.

En la figura 5, se puede ver la cantidad de artículos que se lograron extraer en cada una de las fases:

Figura 5

Cantidad de artículos extraídos en cada fase del SMS.



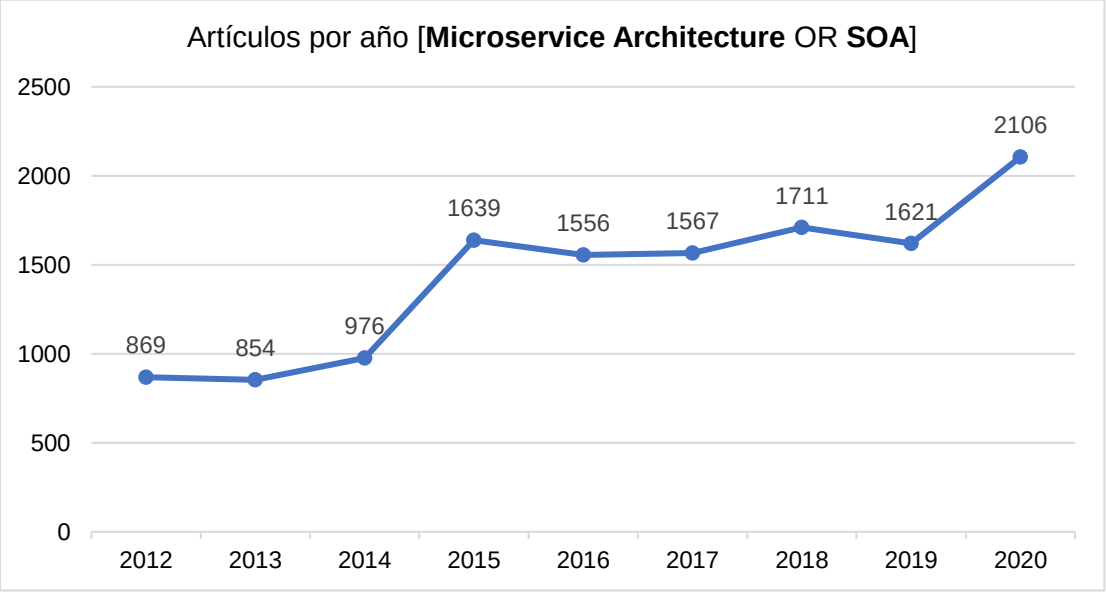
Resultados. Una vez aplicado todo el proceso detallado en este mapeo sistemático, se obtuvo un total de 20 artículos que son de interés y que aportaron positivamente al desarrollo de este trabajo. A continuación, se expone algunos de los datos más importantes que fueron extraídos gracias a este SMS y los que fueron de ayuda para ofrecer una serie de conclusiones importantes.

El primer dato importante que se obtuvo gracias a este SMS y sobre todo gracias a la biblioteca digital ScienceDirect, es el que demuestra que el sector de las

arquitecturas orientadas a servicios o las arquitecturas de microservicios son una temática bastante importante, esto debido a la cantidad de artículos relacionados con SOA. En la figura 6 se puede ver como a partir del año 2015, el número de artículos relacionados con esta temática fueron creciendo y manteniéndose sobre los 1500 artículos por año y llegando al 2020 con la cantidad de 2106 artículos.

Figura 6

Número de artículos por año sobre arquitectura de microservicios (2020)

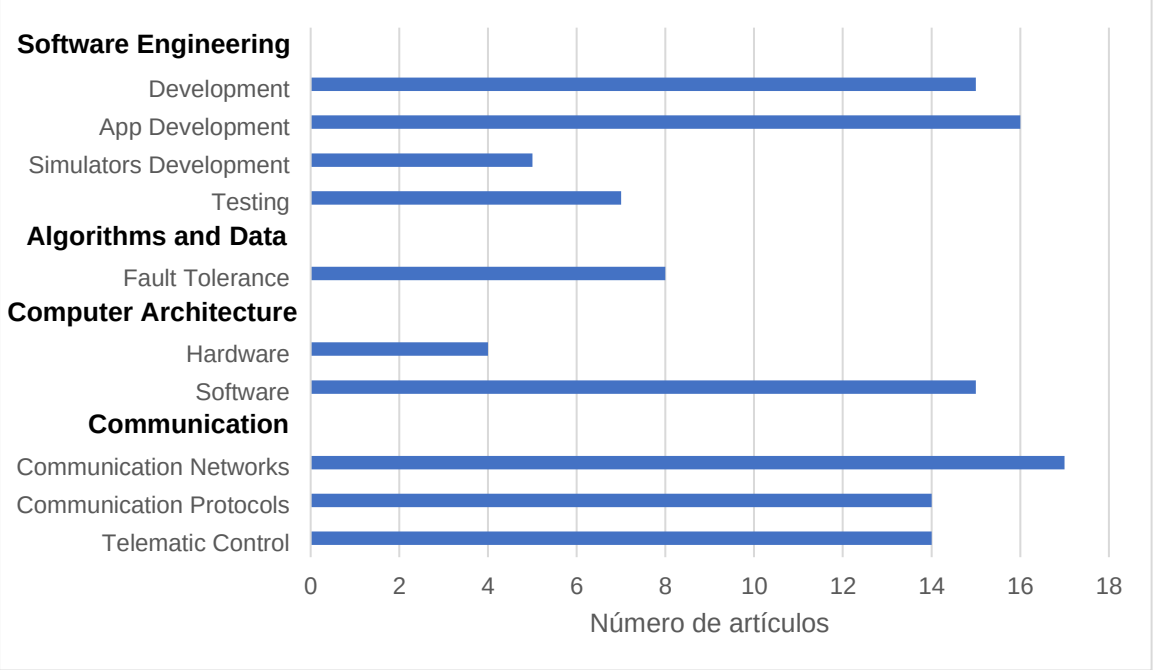


En la figura 7 se puede ver las temáticas que se mencionan en los 20 artículos extraídos en la fase 3, se puede ver que la temática más tratada en estos documentos es *Communication Network* con un total de 17 artículos, lo que representa un 85% del total, le siguen temáticas como *Communication Protocols* con 14 artículos (70% del total) y *Software o App Development* con 16 artículos (80% del total), y se puede decir

que el tema menos tratado a profundidad es el de *Hardware* con 4 artículos (20% del total) .

Figura 7

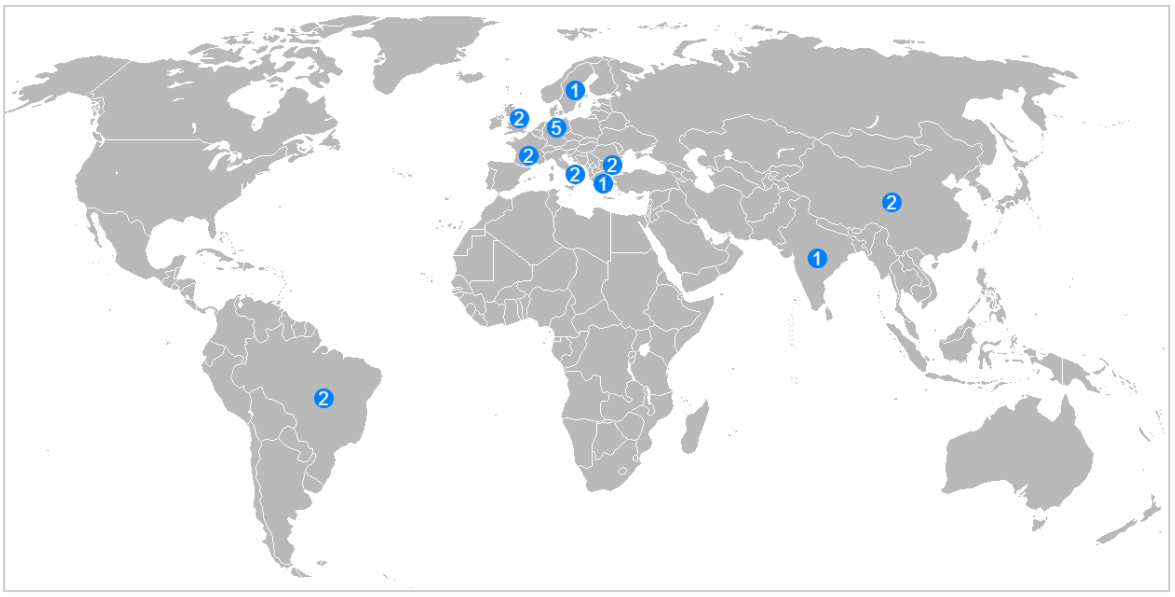
Número de artículos y su tipo, obtenidos en la fase 3 del SMS.



También de todos los documentos estudiados en la fase 3 de este SMS, se pudo extraer que como se puede ver en la figura 8 el país principal en investigar acerca de estos ámbitos y temáticas es Alemania con 5 artículos (25% del total). Es importante mencionar que 18 artículos (90% del total) están abarcados entre Europa y Asia y tan solo 2 artículos (10% del total) se encuentran en Sudamérica y especialmente Brasil.

Figura 8

Ubicación geográfica de los artículos obtenidos en la fase 3 del SMS.



Conclusiones del SMS. Las preguntas que fueron realizadas en el apartado 1.5.1.3 que fueron la motivación para el mapeo profundo de la información, fueron contestadas gracias a los resultados obtenidos y ayudaron a proseguir con la realización del presente trabajo ya que se menciona temáticas adicionales que fueron muy importantes para el desarrollo de producto final. A continuación, se responde las preguntas formuladas:

Q1: ¿Qué técnicas y/o tecnologías existen para el desarrollo de arquitecturas orientadas a microservicios?

Como se pudo ver en la figura 7, en el apartado de Software Engineering y más específicamente en el subapartado *Development,* 15 artículos (75% del total) hablan acerca del desarrollo de este tipo de arquitecturas, se mencionan diversos frameworks

utilizados y orientados a diferentes temáticas, como OpenMTC M2M (Santos, Hoque, & Magedanz, 2017) que está orientada a la comunicación entre máquinas y a un ámbito industrial.

También se menciona frameworks que han sido ampliamente utilizadas en los últimos años y que están enfocados a funcionar adecuadamente con sistemas ciber físicos y aplicaciones IoT/WoT/IoE como: Kubernetes, creado por Google y especializado en gestión y orquestación de contenedores (Eismann, y otros, 2019); Vert.x de Eclipse que corre sobre una máquina virtual de Java, es poliglota y controlado por eventos (Lyu, Biennier, & Ghodous, 2019; Kramer, Frese, & Kuijper, 2019); OSGi, cuyo objetivo es definir especificaciones de software para permitir compatibilidad entre plataformas y brindar múltiples servicios (Bixio, Delzanno, Rebora, & Rulli, 2020); DropWizard, de Java y especializado en el desarrollo de la API REST (Dipsis & Stathis, 2019); Netty, basado en Java y orientado tanto al cliente como al servidor en el desarrollo de aplicaciones (Fan, y otros, 2020); EMF de Eclipse, orientado al modelado y a la generación de código para que pueda ser utilizado en aplicaciones basadas en modelos estructurados (Sampaio, Rubin, Beschastnikh, & Rosa, 2019); Django, escrito en Python, orientado al desarrollo web y respeta el patrón de diseño: modelo-vista-controlador (Benayache, Bilami, Barkat, Lorenz , & Taleb, 2019); Spring, para la plataforma Java y muy peculiar porqué permite el desarrollo de aplicaciones, puede controlar contenedores y tiene soporte para Groovy y Kotlin (Toquica, Benavides, & Motta, 2019).

Finalmente, otros artículos mencionan que desarrollaron sus propias frameworks desde cero, basándose en lenguajes de programación existentes, como Ruby, Java, C/C++, Python, etc. Tal es el caso de Snap4City, una plataforma orientada a la estructuración de ciudades inteligentes y a aplicaciones IoT (Badii, y otros, 2019); otro

ejemplo es New IT Driven SoSM, un framework desarrollado para la manufactura inteligente de cualquier tipo, un avance que se verá en la industria en un futuro próximo. (Tao & Qi, 2019)

Q2: ¿Qué técnicas y/o tecnologías existen para el desarrollo de aplicaciones tanto para el back-end como para el front-end?

La temática de *App Development,* también se encuentra en un puesto notable debido a la cantidad de artículos relacionadas a esta, en 16 artículos (80% del total) se la menciona y con esto las herramientas que se ocupan para desarrollar aplicaciones, en algunos artículos se utiliza los mismos frameworks mencionados en la pregunta Q1, para un desarrollo full stack, pero en otros se mantiene los frameworks para el back-end, pero se utiliza programas aparte para desarrollar el front-end y con esto una interfaz gráfica amigable con el usuario, tal es el caso de Bootstrap, una biblioteca multiplataforma para diseñar aplicaciones y sitios web (Eismann, y otros, 2019); Node-RED, una herramienta de desarrollo de aplicaciones que se basa en flujo de programación y que permite conectar tanto hardware como software y servicios en línea como parte de IoT (Badii, y otros, 2019); algunos artículos no mencionan la herramienta que utilizaron, pero si el lenguaje de programación, como Kotlin (Toquica, Benavides, & Motta, 2019), el cual puede ser utilizado tanto en Eclipse como en Android Studio si se quiere realizar aplicaciones móviles.

Finalmente, algunos artículos solo utilizan interfaces de usuario ya desarrolladas, ya que sus estudios solo se basan en simulaciones o en la realización de pruebas específicas, como el caso de WSN Cooja Simulator o ChartJs Framework (Benayache, Bilami, Barkat, Lorenz , & Taleb, 2019).

Q3: ¿Qué herramientas y/o dispositivos se utilizan para la implementación de sistemas ciber físicos orientados a entornos WoT?

Con este SMS se logró identificar ciertas tecnologías y dispositivos que se pueden utilizar en sistemas ciber físicos y dentro de entornos WoT, en el apartado de *Computer Architecture* y en ambos de sus subapartados *Hardware* con 4 artículos (20% del total) y *Software* con 15 artículos (75% del total), se tratan a ciertos elementos como microcontroladores o sistemas operativos que pueden ser utilizados en aplicaciones IoT/WoT/IoE, tales como: Contiki OS, un sistema operativo que puede funcionar en computadores de 8-bits, hasta sistemas de microcontroladores más complejos, una de sus características es que puede incluir nodos de redes de sensores; TinyOS, un sistema operativo escrito en nesC y orientado a componentes para redes de sensores inalámbricas; Arduino, una placa con los elementos necesarios para que funcione como microcontrolador y que posee su propio sistema operativo. (Benayache, Bilami, Barkat, Lorenz , & Taleb, 2019).

Otros estudios mencionan la importancia de la utilización de ciertos estándares o protocolos de comunicación, tales como Zigbee, un conjunto de protocolos de alto nivel orientados a la comunicación inalámbrica y enfocado en la radiodifusión digital de bajo consumo, mediante el estándar IEEE 802.15.4 de redes inalámbricas de área personal (WPAN), este puede ser utilizado en periféricos, sensores de bajo consumo o en computadores dedicados de placa reducida, tales como Raspberry PI. (Dipsis & Stathis, 2019). En cuanto a Raspberry PI, se la menciona por su importancia y su facilidad para formar parte de un sistema ciber físico, ya sea como núcleo o como un Edge Node o Gateway, entre sus características se encuentra la funcionalidad de mejorar aplicaciones orientadas a Smart Cities por su rápido tiempo de respuesta en tiempo real, además de que su bajo costo no es un impedimento para seguir escalando un

producto final. (Thramboulidis, Vachtsevanou, & Solanos , 2018; Badii, y otros, 2019; McNally, Chaplin, Martínez, & Ratchev, 2020; Dipsis & Stathis, 2019)

Finalmente, en ciertos artículos se menciona que se puede utilizar dispositivos inteligentes para poder adquirir datos o ser utilizados dentro de entornos WoT, esto gracias a las prestaciones que actualmente brindan, como son los celulares, tablets, smartwatch, etc. (Kramer, Frese, & Kuijper, 2019). También se menciona ciertas marcas muy conocidas que son distribuidoras de productos inteligentes orientados a IoT/WoT, como altavoces, pantallas, dispositivos de transmisión, termostatos, detectores de humo, cámaras, cerraduras y timbres, como, por ejemplo: Google Nest LLC, IFTTT, Amazon Alexa. (Dipsis & Stathis, 2019)

Q4: ¿Qué técnicas y/o tecnologías existen para la adquisición, procesamiento y almacenamiento de datos de sistemas ciber físicos en entornos WoT?

Para la adquisición de datos, la mayoría de artículos cuya aplicación está orientada a recolectar datos del entorno, hablan de sensores y actuadores, como se sabe existen múltiples dispositivos que pueden desempeñar estas funciones, lo importante es su enfoque, ya que pueden muestrear contaminación ambiental, temperatura, presión o también pueden ser usados como identificadores como por ejemplo los RFID (Butzin, Golatowski, & Timmermann, 2016; Santos, Hoque, & Magedanz, 2017; Lyu, Biennier, & Ghodous, 2019), no se mencionan marcas específicas, ya que esto dependerá más de cada investigador. También se habla acerca de aplicaciones donde los datos entrantes son las interacciones de los usuarios, en este caso no existe un variable que medir, sino que los datos están orientados para procesarse y brindar un servicio, como la tienda de té mencionada (Eismann, y otros, 2019) . Algo importante que recalcar es el formato de las cadenas de datos que son

enviados a las unidades de almacenamiento, al tratarse de servicios web, la mayoría de artículos mencionan la importancia de utilizar el formato JSON, y en algunos casos utilizan herramientas como RabbitMQ, un software de código abierto, orientado a la negociación de mensajes y que funciona como un middleware de mensajería (Bixio, Delzanno, Rebora, & Rulli, 2020).

En cuanto al almacenamiento de los datos, todos los artículos extraídos mediante este SMS mencionan bases de datos de diversos tipos, ya sea para su utilización o como una revisión de lo que pueden ofrecer, entre las que se mencionan se encuentran: PostgreSQL, una potente base de datos relacional orientada a objetos y de código abierto (Dipsis & Stathis, 2019); Spanner y OceanBase, bases de datos distribuidas, denominadas “NewSQL”, caracterizadas por estar distribuidas globalmente, ser escalables y proporcionar replicación de sitios automática (Fan, y otros, 2020); MySQL, considerada la base de datos relacional más popular del mundo, es de Oracle y posee dos tipos de licencia, pública y comercial (Sampaio, Rubin, Beschastnikh, & Rosa, 2019; Butzin, Golatowski, & Timmermann, 2016; Pop, y otros, 2019; Gifu & Pop, 2020); MongoDB, la base de datos *no* relacional más popular de mundo, orientada a documentos y que guarda sus datos en estructuras con formatos BSON (Sampaio, Rubin, Beschastnikh, & Rosa, 2019; Lyu, Biennier, & Ghodous, 2019); Amazon S3, un almacenamiento de objetos que esta proporcionada a través de una interfaz de servicios Web, junto con Amazon DynamoDB una base de datos no relacional se orienta a aplicaciones más robustas (Kramer, Frese, & Kuijper, 2019). Un par de bases de datos peculiares mencionadas fueron: InfluxDB, una base de datos de series de tiempo, utilizada para la recolección de datos en grandes cantidades y que varían respecto avanza su secuencia (McNally, Chaplin, Martínez, & Ratchev, 2020); y TinyDB, una base de datos temporal y persistente, que está orientada a aplicaciones como

videojuegos, por ejemplo almacenando puntuaciones altas cada vez que se juega (Benayache, Bilami, Barkat, Lorenz , & Taleb, 2019). Finalmente, algunos artículos mencionan que desarrollaron sus propias bases de datos, pero basándose en SQL. (Bixio, Delzanno, Rebora, & Rulli, 2020; Eismann, y otros, 2019; Parmar & Champaneria, 2017)

Una vez la información se encuentra en las bases de datos, para procesarla, en los artículos se menciona que se utiliza las funciones básicas que brindan dichas herramientas, conocidas por el acrónimo CRUD (del inglés Create, Read, Update, Delete), esto junto a las frameworks de desarrollo puede ser utilizado para alcanzar el objetivo propuesto, dependiendo de cada aplicación. (Dipsis & Stathis, 2019; Fan, y otros, 2020).

Otras conclusiones:

Además de haber respondido las preguntas que fueron planteadas, se extrajeron otras conclusiones que fueron relevantes para el desarrollo del presente trabajo:

- Como se mencionó en la sección introductoria, existen dos maneras de invocar los servicios en SOA, la coreografía y la orquestación; con la realización de este SMS se pudo notar que su uso depende netamente de la aplicación a la que está orientada el trabajo, y es muy importante saber diferenciar cual utilizar, ya que se pueden dar casos en los que es mucho más conveniente utilizar la una antes que la otra.
- Con respecto a la arquitectura de microservicios, es importante seguir una estructura que ya se encuentra estandarizada, esto debido a que el desarrollo de software es un proceso sistemático que debe realizarse en orden para evitar problemas en la implementación.

- Como se pudo ver en la figura 7 en el subapartado de *Simulators Development* con 5 artículos (25% del total) *y Testing* con 7 artículos (35% del total), son estudios que requerían una simulación previa o simplemente eran estudios de ensayo, esto quiere decir que, en su mayoría, 75% de los artículos estudiados fueron implementaciones prácticas y aplicadas en muchos casos a ámbitos reales, como la manufactura de zapatos, la venta de Té, la automatización de comunidades u hogares, etc. Esto hace notar que la importancia de los entornos WoT va en crecimiento y aportará significativamente en desarrollos futuros.
- Algunos de los artículos mencionan el concepto ***Manufacturing*** o ***Industrial,*** es importante notar esto, ya que se puede ver que en el futuro las implementaciones basadas en microservicios podrían encontrarse en procesos industriales importantes, que lo que buscaría es un ahorro para las empresas.
- Todos los artículos estudiados son pertenecientes a Conferencias o revistas indexadas pertenecientes a las áreas de interés.
- Un punto de estudio importante en el presente trabajo es la *Tolerancia a Fallos* y se pudo notar que solo 8 artículos (40% del total) mencionan esta temática, sin embargo, no profundizan en su implementación, por lo que es un punto importante a tratar y a estudiar.
- Otra conclusión importante que se extrajo de este SMS es qué no fue muy complicado realizar una comparativa o responder las preguntas planteadas, esto debido a que como se trata de desarrollo de software o de sistemas estandarizados, estos deben cumplir ciertas reglas y normas para su correcto funcionamiento.

- Los protocolos de comunicación más utilizados para este tipo de aplicaciones son los que se ejecutan sobre TCP/IP, tales como HTTP, MQQT, además son adoptados debido a su facilidad de interacción con los servicios web y con archivos HTML, XHML, XML, etc.

Como conclusión final se puede decir que las implementaciones y trabajos desarrollados son ad-hoc para tareas concretas, pero es importante notar la versatilidad que pueden tener este tipo de sistemas y arquitecturas, ya que, si bien su desarrollo puede ser específico para un objetivo, la escalabilidad y la apertura que tienen para cualquier campo de aplicación puede ser de gran interés, ya que no tiene limitantes definidas.

Revisión Sistemática de la Literatura

Después de haber realizado el SMS, se pudo ver el estado de madurez y las líneas de trabajo actuales tanto para las arquitecturas de microservicios como para los sistemas ciber físicos. Cabe recalcar que de los artículos estudiados se han encontró una diversidad considerable en cuanto a las tecnologías utilizadas y a los campos de aplicación, sin embargo muy pocos mencionan la importancia de la *tolerancia a fallos* y al *control telemático* orientado a sistemas ciber físicos, por lo que es conveniente investigar más a fondo estas dos temáticas para conocer a detalle que técnicas y métodos se están utilizando actualmente en estos ámbitos y es la motivación de esta Revisión Sistemática de la Literatura, en adelante SLR (Systematic Literature Review).

La principal diferencia del SLR con respecto al SMS, es que trata de profundizar uno o más temas en concreto y conocerlos a detalle. Además, tiene como propósito integrar los resultados de diversas fuentes de una manera objetiva y sistemática para

poder alcanzar un “estado del arte” en estos temas, es por ello que el proceso del SLR, requiere desarrollar una serie de pasos similares a los realizados en el SMS.

Por lo tanto, a lo largo de esta sección se presenta el SLR, realizado con el objetivo principal de brindar un conocimiento completo y detallado sobre las temáticas *tolerancia a fallos* y *control telemático*, ya que son de mucho interés para el presente trabajo.

Objetivos. Lo que se busca es conocer más a fondo las técnicas, métodos y herramientas que se utilizan para gestionar los fallos dentro de una arquitectura de microservicios para que pueda ser considerada tolerante a fallos y de igual manera que medios son los más óptimos y viables para realizar un control telemático de sistemas ciber físicos, todo esto con la finalidad de tener un bagaje que pueda contribuir con la actual línea de investigación.

El bagaje de información que se va adquirir gracias a este SLR servirá para saber en qué líneas se está trabajando actualmente en el mundo y conocer si las tecnologías han podido ser aplicadas en entornos reales y conocer cuál ha sido su impacto y sus resultados.

Se optó por un SLR debido a las preguntas necesarias que deben plantearse para poder conocer las investigaciones que se han publicado en los últimos años en las temáticas mencionadas, que con el SMS no basto para solventar.

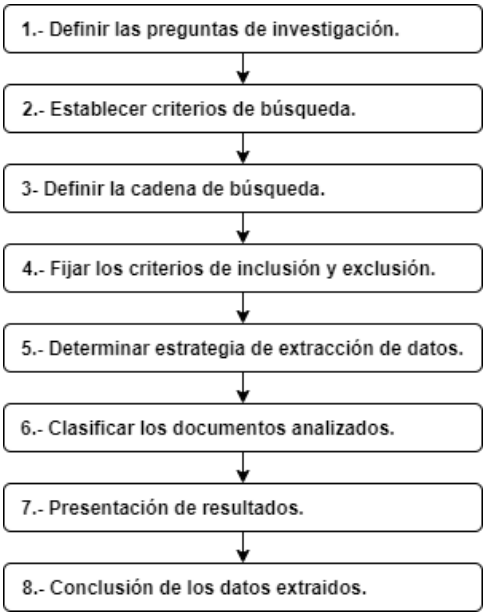
Método de investigación. El presente SLR consta de tres actividades: planteamiento, ejecución y extracción de resultados. El planteamiento consiste en desarrollar un plan de trabajo para la elaboración adecuada del SLR. La ejecución consiste en la búsqueda y extracción de información adecuada a varios parámetros que

deben ser considerados para poder seleccionarla y sintetizarla de manera efectiva. Finalmente, la extracción de resultados consiste en la interpretación de los resultados y de plasmar las conclusiones respecto al trabajo realizado.

Como se mencionó, el proceso del SLR es muy similar al del SMS y se siguió los siguientes pasos:

Figura 9

Proceso seguido para el SLR



Nota: Inspirado en (Moguel Márquez, 2018).

Definir las preguntas de investigación. Al igual que en el SMS, la motivación de este SLR es examinar las tecnologías que existen acerca de las temáticas *tolerancia a fallos* y *control telemático* ya que como el título del presente trabajo refleja: *“Diseño de una arquitectura para el control telemático de Sistemas Ciber-*

físicos basada en microservicios y con tolerancia a fallos." son dos puntos esenciales que deben ser analizados con más detalle.

Tabla 4

Preguntas para investigación del SLR

Pregunta de investigación	Motivación
¿Qué técnicas/tecnologías existen para la tolerancia a fallos en arquitecturas de microservicios?	
Q1: ¿Qué técnicas y/o tecnologías se utilizan para implementar y gestionar la tolerancia a fallos en arquitecturas de microservicios?	Conocer qué tecnologías, técnicas o herramientas existen para implementar la tolerancia a fallos en arquitecturas de microservicios.
Q2: ¿Qué técnicas y/o tecnologías se utilizan para realizar *Benchmarking* de tolerancia a fallos?	Conocer qué tecnologías, técnicas o herramientas existen para realizar pruebas con el objetivo de medir y validar la tolerancia a fallos.
Q3: ¿Qué técnicas y/o tecnologías se utilizan para realizar *Benchmarking* en arquitecturas de microservicios?	Conocer qué tecnologías, técnicas o herramientas existen para realizar pruebas con el objetivo de medir y validar una arquitectura de microservicios.
¿Qué técnicas/tecnologías existen para el control telemático de sistemas ciber físicos en entornos WoT?	
Q4: ¿Qué técnicas y/o tecnologías se utilizan para el control telemático de sistemas ciber físicos?	Conocer qué tecnologías, técnicas o herramientas existen en los sistemas de control de sistemas ciber físicos en entornos WoT y cuál es su nivel de madurez.
Q5: ¿Qué técnicas y/o tecnologías se utilizan para determinar la QoS en cuanto al control de sistemas ciber físicos?	Conocer qué tecnologías, técnicas o herramientas existen para brindar una QoS adecuada en un entorno WoT.

Nota: Se ha considerado investigar acerca de las pruebas de rendimiento o *Benchmarking,* ya que son una herramienta importante para conocer la madurez y la validez de la tecnología. Además, en cuanto a los medios que se utilizan para el control telemático, es importante conocer cuáles son los más óptimos y de este modo poder garantizar una QoS adecuada para la aplicación que está orientada el proyecto. Sin embargo, las temáticas mencionadas como la *Tolerancia a fallos* y *Control telemático* son tratados con más detalle en la sección 1.6 del presente trabajo.

Establecer los criterios de búsqueda primarios y secundarios.

Algunos de los criterios considerados son similares a los presentados en el SMS, pero lo que se busca en este SLR es una mayor profundización técnica y mucho más detalle en la información expuesta acerca de las temáticas específicas, por lo que se descartan revisiones de publicaciones, publicaciones que sean solo de divulgación, capítulos o libros completos, páginas Web y todos los artículos que simplemente tengan como propósito la divulgación científica.

Se utilizaron los mismos motores de búsqueda que en el SMS, ya que se pudo notar que disponen de una cantidad de artículos considerable en las temáticas de interés, y en ámbitos de la *Computer Science, Software Development* y *Communication,* los cuales son: IEEE Xplore, Science Direct, Springer Link y MDPI.

En el SMS también se pudo identificar y diferenciar algunas conferencias y una serie de congresos internacionales que son de relevancia para esta investigación, y se ha considerado que deben ser incluidos, entre algunos se encuentran:

- IEEE World Congress on Services (SERVICES).
- KES International Symposium.
- IEEE Internet Of Things (IoT).
- International Conference on World Wide Web, WWW.

Definir cadena de búsqueda. A partir de las preguntas planteadas en este SLR, se han definido los términos principales en cada una de ellas, para establecer una cadena de búsqueda adecuada que abarque todas las temáticas de interés.

Tabla 5

Términos definidos para la cadena de búsqueda del SLR.

Término principal	Términos alternativos
Fault tolerance. [Q1]	"Cyber resilience" **OR** Failover
Microservices oriented architectures. [Q1]	"Service oriented architecture" **OR** SOA **OR** "Microservice architecture".
Benchmark. [Q2. Q3]	Baseline.
Telematic control. [Q4]	Telecommunication control **OR** Remote control.
QoS. [Q5]	Quality of service.

Por lo tanto, la cadena de búsqueda resultante a partir de los términos mencionados, fue la siguiente:

```
("Fault Tolerance" OR "Cyber resilience" OR Failover)
AND
("Microservices oriented architectures" OR "Service oriented
architecture" OR SOA OR "Microservice architecture".)
AND
("Benchmark" OR Baseline)
AND
("Telematic control" OR "Telecommunication control" OR "Remote control")
AND
(QoS OR "Quality of service").
```

Fijar los criterios de inclusión y exclusión. Para esta búsqueda se tuvieron en cuenta todos los artículos publicados hasta enero del 2021, y los posteriores a enero de 2017, además se debe tener en cuenta que los resultados obtenidos de este SLR llevaron a determinar las características necesarias de las temáticas propuestas para que puedan ser consideradas válidas.

Para cada documento encontrado y para saber si era o no pertinente incluirlo en este trabajo, se decidió que para que las publicaciones sean incluidas deben cumplir con las siguientes condiciones:

- Artículos completos
- Pertenecientes a las ramas de “Ciencias de la computación / Computer Science” o “Ingeniería / Engineering”, de esta última, siempre y cuando tenga que ver con el área de desarrollo tecnológico o las telecomunicaciones.
- Publicaciones que especialmente sean artículos, conferencias, revistas científicas, tesis, etc.
- Estudios que presenten métodos o técnicas que ayuden a la implementación adecuada de la tolerancia a fallos y al control telemático se sistemas que incluyan tanto los sistemas ciber físicos como las arquitecturas de microservicios.
- Publicaciones desde enero de 2017 hasta enero de 2021.
- Artículos escritos en cualquier idioma, mientras sean pertinentes y correctamente publicados o indexados.

Por otro lado, se excluyeron los estudios que cumplieron con al menos una de estas condiciones:

- Los artículos relacionados con campos de ciencia que no involucren el desarrollo tecnológico o electrónico.
- Los artículos que traten de arquitecturas monolíticas y no de microservicios, o que solo abarquen una comparación entre ambas.

- Los escritos que no traten específicamente sobre la tolerancia a fallos y el control telemático dentro de arquitecturas de microservicios, sistemas ciber físicos o Web of Things, y que no sean de interés para el avance del trabajo o que simplemente se mencione alguna de estas temáticas, pero que no sean su foco principal de estudio.
- Artículos que no traten la parte del software de implementación de la tolerancia a fallos o el control telemático como su enfoque, y que únicamente los mencionen como avance para alguna aplicación, sin abordar a profundidad su estructura, diseño y medios más adecuados de desarrollo.
- Los documentos que no se encuentren detallados con claridad y que únicamente se trate de una herramienta introductoria a lo que se va a estudiar.
- Artículos que no tengan una estructura adecuada, como presentaciones de PowerPoint o similares.
- Artículos duplicados o que provengan de la misma fuente.

Determinar la estrategia de extracción de los datos. Al igual que en el SMS, es importante extraer y sintetizar los datos de manera adecuada, por lo que fue importante desarrollar una ficha para poder extraer toda la información e ir registrando todos los hallazgos importantes y además de los datos bibliográficos correspondientes. La clasificación utilizada para este SLR es similar a la del SMS, con ciertas variantes importantes que estén incluidas en las subcategorías de la Ingeniería de Software o de las Telecomunicaciones. Finalmente se realizó un resumen para quitar todos los artículos duplicados o de poco interés.

La ficha de extracción para este SLR, es la siguiente:

Tabla 6

Ficha para extracción de información del SLR.

General Information	
Title	
Launch	
Source	
Relevance	
Is this article relevant to Fault Tolerance in Microservice Architecture Field?	☐ Highly Relevant ☐ Relevant ☐ Irrelevant
Is this article relevant to Telematic Control in CPS Field?	☐ Highly Relevant ☐ Relevant ☐ Irrelevant
Application domain?	☐ Surveillance and Security ☐ Environmental Control ☐ Location and Telematic control. ☐ Commerce and entertainment ☐ Digital services (medical, economic, industrial, etc.)
Benchmark	☐ Yes ☐ No
Study	
Architecture	☐ Service oriented ☐ Microservice ☐ Monolithic
Focus of the study	☐ **Software Engineering** ☐ Development (implementation, language, etc.). ☐ UI development. ☐ Baseline Software. ☐ **Algorithms and data structures** ☐ Fault Tolerance (Collision avoidance, Cascading failure). ☐ Control algorithm. ☐ **Communication** ☐ Communication Networks. ☐ Communication Protocols (MQTT, HTTP, etc.) ☐ Telematic control. (Internet, WiFi, Bluetooth).
Fault Tolerance Implementation.	☐ Very Detailed ☐ Detailed ☐ Not Detailed
Telematic control Implementation.	☐ Very Detailed ☐ Detailed ☐ Not Detailed

Clasificación de los documentos extraídos. Al igual que en el SMS el proceso de extracción para este SLR se lo realizo en tres etapas:

1. En la primera fase se utilizó la cadena de búsqueda expuesta en el apartado 1.5.2.5 y para el filtrado en cada una de las bibliotecas digital, solo se ha tenido en cuenta: **Title + Abstract + Keywords,** obteniendo un total de 575 resultados entre las distintas bibliotecas digitales, de la siguiente manera:
    - **IEEE Xplore:** 37 resultados.
    - **Springer Link:** 387 resultados.
    - **ScienceDirect:** 124 resultados.
    - **MDPI:** 27 resultados.
2. En la segunda fase se realizó un filtrado de los artículos encontrados en la primera fase, pero tomando en cuenta los criterios expuestos en el numeral 1.5.2.6 y al igual que en el paso anterior los filtros utilizados fueron: **Title + Keywords + Abstract,** obteniendo un total de 141 resultados entre las distintas bibliotecas digitales, de la siguiente manera:
    - **IEEE Xplore:** 27 resultados.
    - **Springer Link:** 30 resultados.
    - **ScienceDirect:** 67 resultados
    - **MDPI:** 17 resultados.
3. En la tercera y última fase, se aplicó un filtro mucho más exhaustivo en el que a partir de los artículos obtenidos en la segunda fase, solo se consideran los que son verdaderamente de interés y que abordan muy detalladamente los métodos y técnicas que pueden ser utilizados para el presente trabajo. Los filtros utilizados fueron: **All Metadata,** es decir, **Title + Keywords + Abstract + Full Article,**

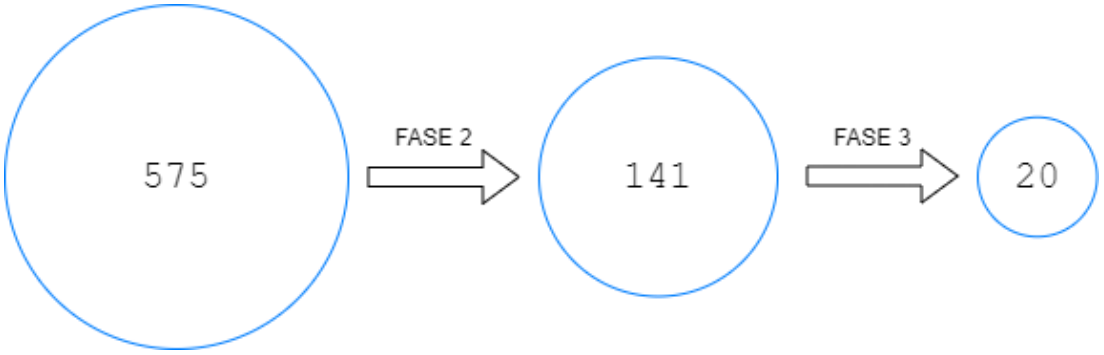
obteniendo un total de 20 resultados entre las distintas bibliotecas digitales, de la siguiente manera:

- **IEEE Xplore:** 9 resultados.
- **Springer Link:** 3 resultados.
- **ScienceDirect:** 6 resultados
- **MDPI:** 2 resultados.

En la figura 10, se puede ver la cantidad de artículos que se lograron extraer en cada una de las fases:

Figura 10

Cantidad de artículos extraídos en cada fase del SLR.

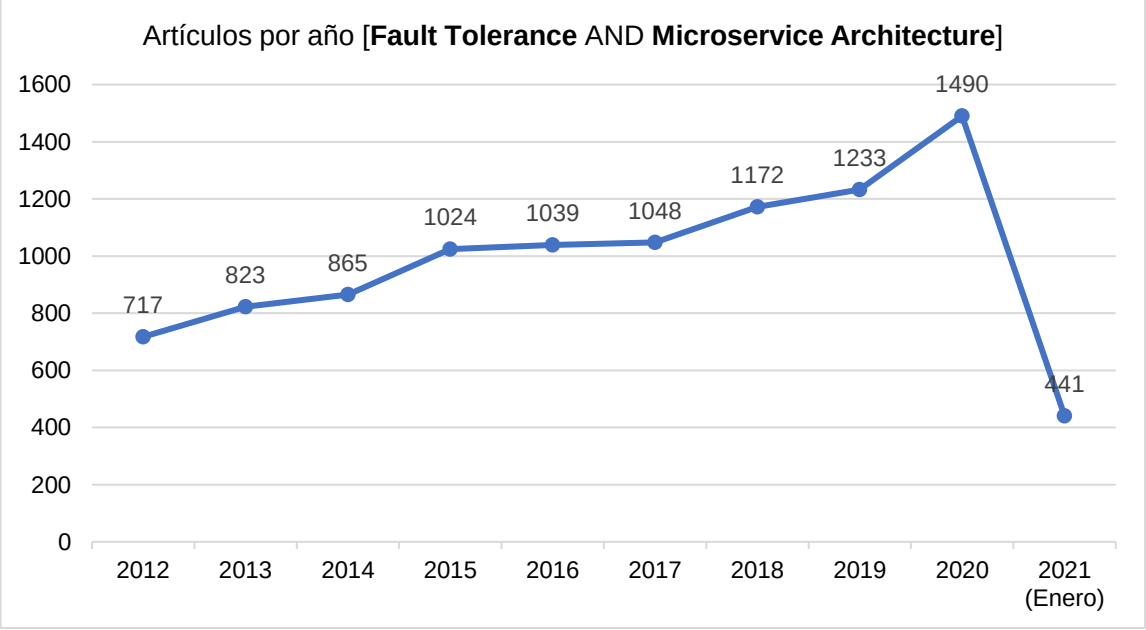


Resultados. Una vez aplicado todo el proceso detallado en esta revisión sistemática de la literatura, se obtuvo un total de 20 artículos que son de interés y que aportaron positivamente al desarrollo de este trabajo gracias al detalle y profundidad abordado en las temáticas de interés *Tolerancia a fallos* y *Control telemático*. A continuación, se expone algunos de los datos más importantes que fueron extraídos gracias a esta SLR y los que son de ayuda para ofrecer una serie de conclusiones importantes.

El primer dato importante que se obtuvo gracias a la SLR y sobre todo gracias a la biblioteca digital ScienceDirect, es el que demuestra que en el sector de las arquitecturas orientadas a servicios o en las arquitecturas de microservicios, la importancia de la tolerancia a fallos ha sido crucial para el desarrollo y crecimiento de las mismas, siendo parte de sus características esenciales, esto debido a la cantidad de artículos relacionados con *Fault Tolerance*. En la figura 11 se puede ver como a partir del año 2015, el número de artículos relacionados con esta temática fueron creciendo y manteniéndose sobre los 1000 artículos por año y llegando al 2020 con la cantidad de 1490 artículos, además es muy importante mencionar que, hasta la fecha, enero de 2021, ya hay publicados 441 artículos lo que demuestro una proyección muy prometedora en este campo de estudio.

Figura 11

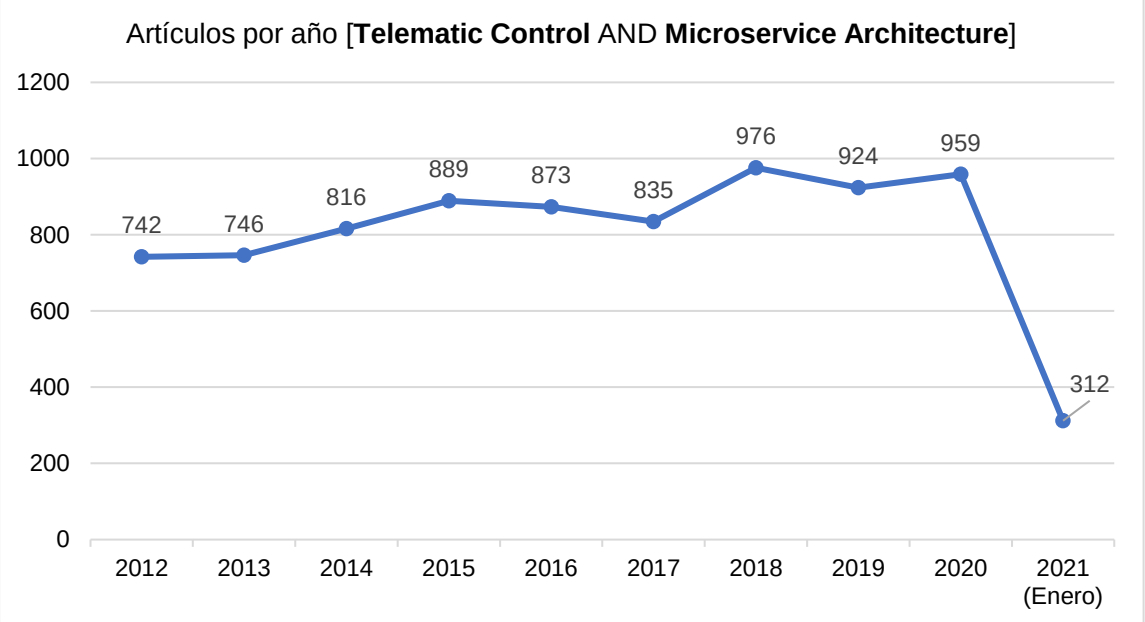
Artículos por año sobre tolerancia a fallos en arquitecturas de microservicios (enero 2021)



Otro dato importante es el que se refleja a la temática de *Telematic Control* en arquitecturas orientadas a microservicios, si bien no hay una cantidad considerable como en el resto de temáticas, en la figura 12 se puede apreciar que a partir del 2015, el número de artículos relacionados con este tópico se han mantenido sobre los 800, lo cual demuestra que si hay un campo de estudio en esta área, además es muy importante mencionar que, hasta la fecha, enero de 2021, ya hay publicados 312 artículos lo que demuestro una buena proyección en este campo de estudio.

Figura 12

Artículos por año sobre control telemático en arquitecturas de microservicios (enero 2021)

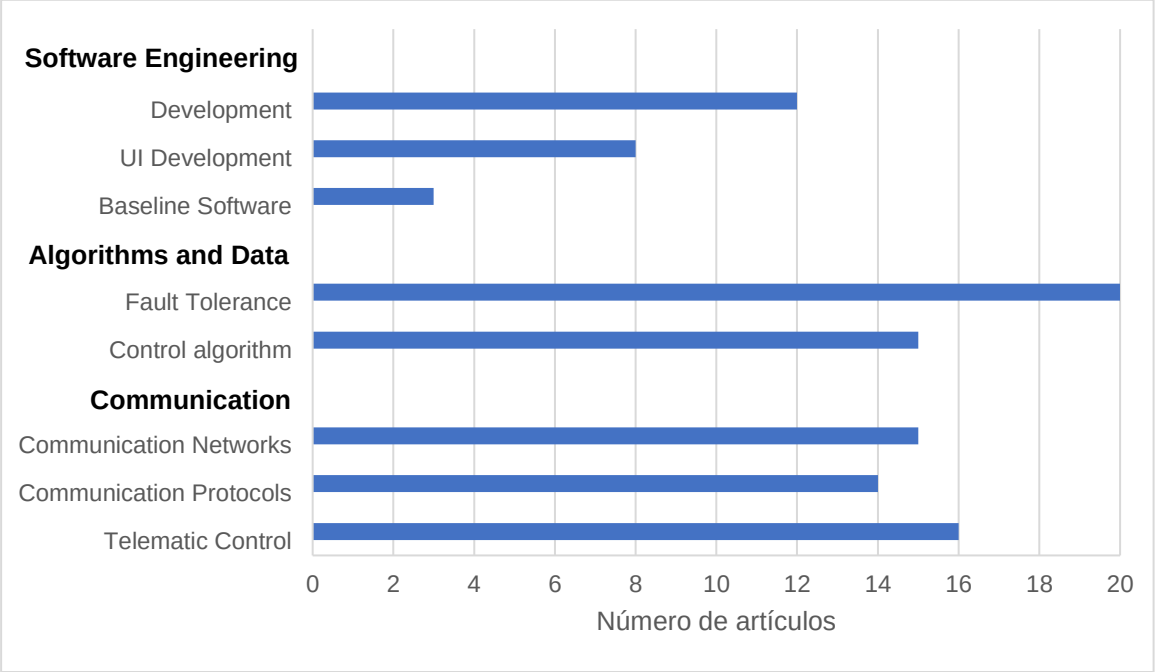


En la figura 13 se puede ver las temáticas de los 20 artículos estudiados, pero al tratarse de un análisis con temáticas específicas, se puede apreciar que los 20 artículos mencionan *Fault Tolerance* (100% del total), también la temática de *Telematic Control* es mencionada en 16 artículos (80% del total), le siguen las temáticas de *Control Algorithm* con 15 artículos (75% del total) y *Communication Network* también con 15

artículos (75% del total), finalmente se puede decir que el tema menos tratado es el de *Baseline Software,* ya que solo se menciona algo al respecto en 3 artículos (15% del total).

Figura 13

Número de artículos y su tipo, obtenidos en la fase 3 del SLR.



Conclusiones del SLR. Gracias a la información obtenida en esta revisión de la literatura, las preguntas planteadas en la sección 1.5.2.3 lograron ser contestadas y se pudo aclarar las dudas más importantes acerca de las temáticas de estudio que eran de interés para profundizar con esta SLR.

A continuación, se procede a responder las preguntas:

Q1: ¿Qué técnicas y/o tecnologías se utilizan para implementar y gestionar la tolerancia a fallos en arquitecturas de microservicios?

Como se pudo ver en la figura 13, de los 20 artículos estudiados, en todos se menciona la tolerancia a fallos y algunas soluciones para que se pueda implementar dentro de algún sistema y sea capaz de gestionar la misma, tal es el caso de Docker, un conjunto de productos que se denominan plataformas como servicio (PaaS) que utilizan la virtualización para entregar un software dentro de contenedores, que tienen como característica la modularidad y la orquestación que permite sistemas más dinámicos y la tolerancia a fallos puede ser lograda de manera adecuada ya que la aplicación se distribuye a lo largo de diferentes capas, entonces en caso de presentarse algún error, solo se lo gestiona en la capa adecuada evitando el impacto en todo el desempeño del sistema. (Alam, y otros, 2018); también se mencionan softwares específicos orientados a la tolerancia a fallos en sistemas informáticos, tan es el caso del desarrollado por Hewlett Packard Enterprise y denominado NonStop, funcional y adaptable con un entorno de software abierto, implementado para tareas cruciales y utilizado por su característica de permitir un acople adecuado entre componentes tanto de hardware como de software. (Richter, Konrad, Utecht, & Polze, 2017); además de softwares específicos, también se mencionan métodos que se han desarrollado aplicando algoritmos como el de *Hearthbeat Detection,* que aplicado a las arquitecturas de microservicios puede ser muy útil para encontrar y gestionar errores dentro de una red, una de sus desventajas en que no se puede diagnosticar que tan grave puede ser una falla. (Zang, y otros, 2018); y métodos que pueden ser aplicados a diferentes entornos, como: Network Fault Aware (NFA) que gestiona fallos de la red mientras se ejecuta la aplicación, Resource Fault Aware (RFA): que gestiona fallos en los nodos mientras se

ejecuta la aplicación y Hybrid Fault Strategy (HYD): que gestiona tanto los fallos en la red como los nodos mientras se ejecuta la aplicación. (Lakhan & Li, 2019).

Finalmente los artículos mencionan plataformas propias que son desarrolladas para objetivos específicos, pero todos mencionan como característica principal una arquitectura y un sistema distribuido por capas para la mejor gestión de fallos, además de usar orquestación como su método principal para llamar a los procesos (McNally, Chaplin, Martínez, & Ratchev, 2020; Malyuga, Perl, Slapoguzov, & Perl, 2020); tal es el caso de BigVM, cuyos mecanismos son transparentes de extremo a extremo y se reflejan en la aplicación de usuario (Zheng, Zhang, Zheng, Fu, & Liu, 2017); o un sistema de microservicios dinámico que está orientado a controlar el flujo de clientes que ingresan a una plataforma que brinde algún servicio específico (Baboi, Iftene, & Gîfu, 2019); se mencionan también diferentes patrones de diseño para implementar la tolerancia a fallos y que corresponden a diferentes niveles de consistencia (ONE, ALL, QUORUM) (Gorbenko, Romanovsky, & Tarasyuk, 2019).

Q2: ¿Qué técnicas y/o tecnologías se utilizan para realizar *Benchmarking* de tolerancia a fallos?

En la figura 13 se puede ver que *Baseline Software* solo se menciona en 3 artículos (15% del total), es decir solo en estos se mencionan programas específicos para realizar un *Benchmarking* y evidenciar las buenas prácticas respecto a esta área de interés que es la *Tolerancia a fallos,* se menciona métodos y modelos orientados a probar las capacidades de la prevención de fallos en un sistema, tal es el caso de DevOps Capability Model, el cual consiste en tres fases: desarrollo, prueba y operación, por lo que en todo su tiempo de utilización desde su implementación, permite que tanto su mantenimiento y gestión sean más rápidas y confiables, además aporta

automatización para realizar las pruebas y notificar resultados (Huang, Zhuang, Sun, & Zhang, 2020); en otro artículo se desarrolla un algoritmo enfocado específicamente al Benchmarking de la tolerancia a fallos denominado ORCAS, los autores mencionan que de por si los microservicios brindan un tolerancia a fallos adecuada, pero con este método se mejoran características como: aumentar la confiabilidad y la resiliencia del sistema cuando es sometido a este tipo de procesos, usar simulaciones de un sistema para evitar hacer inyecciones de errores constantemente y finalmente reducir la cantidad total de benchmarks que se deben realizar, disminuyendo el tiempo de ejecución de las pruebas y evitando perdidas de producción. (Hoorn, Aleti, Dullmann, & Pitakrat, 2018). Finalmente, un artículo aborda el Benchmarking de la tolerancia a fallos, de una manera diferente y sin la utilización de algún software específico, sino independientemente en cada uno de los microservicios, haciendo una inyección de falla en diferentes casos o entornos de prueba en los que van cambiando diferentes factores, de este modo logran determinar cuál es el desempeño real de la tolerancia a fallos, de donde concluyen que para circunstancias específicas el desempeño general del sistema no es óptimo y por lo tanto algo está fallando en la implementación de la tolerancia a fallos y su gestión de los mismos. (Lakhan & Li, 2019).

Q3: ¿Qué técnicas y/o tecnologías se utilizan para realizar *Benchmarking* en arquitecturas de microservicios?

Hace algunos años no había una medición específica orientada a las diferentes arquitecturas, pero desde la aparición de las arquitecturas a microservicios se ha visto un cambio en el enfoque y en la forma de evaluar el desempeño de las mismas, ya que una de sus características principales es ser tolerantes a fallos y además ser escalables. De los artículos estudiados, en 9 (45% del total) se menciona como se

realiza Benchmarking de este tipo de arquitecturas y que la forma más adecuada de hacerlo es ir probando cada microservicio por separado y su interacción con los otros microservicios del sistema y poco a poco el desempeño de todo el sistema pueda ser evaluado, en las arquitecturas monolíticas esto no era posible, ya que se evalúa todo en conjunto y ahí la necesidad de muchos más recursos. (Richter, Konrad, Utecht, & Polze, 2017); en algunos de los artículos se desarrolla entornos SandBox para la realización del Benchmarking, es decir el sistema va a estar inmerso en un proceso que es orientado específicamente para la realización de pruebas, de esto modo ya depende de cada desarrollador y a donde va orientada su aplicación, sin embargo pueden verse ejemplos como la evaluación de sistemas de Cloud Computing (Malyuga, Perl, Slapoguzov, & Perl, 2020), sistemas híbridos para Smart Cities (Kramer, Frese, & Kuijper, 2019; Badii, y otros, 2019), y sistemas de análisis de datos, donde el desarrollo de una interfaz gráfica es crucial, ya que la evaluación de desempeño se basa netamente en el análisis de gráficos (Ma, Fan, Chuang, Liu, & Lan, 2019); aparte de que se deba probar cada microservicio por separado, es importante considerar las características de hardware que se dispone y la memoria del sistema que se le otorgado a cada uno de los microservicios, ya que esto puede influir en el desempeño de la arquitectura total, además se puede ir variando la cantidad de memoria RAM que se le otorga a cada contenedor y de este modo identificar la que sea más adecuada (Trunov, Voronova, Voronov, Sukhachev, & Strelnikov, 2018), de este modo también se puede un panorama adecuado de todas las características físicas que debe tener toda la implementación del sistema y considerar tanto CPU, como unidad de almacenamiento óptimos (Zheng, Zhang, Zheng, Fu, & Liu, 2017).

Q4: ¿Qué técnicas y/o tecnologías se utilizan para el control telemático de sistemas ciber físicos?

Gracias al desarrollo tecnológico y a la facilidad que ahora se tiene para la implementación de interfaces de comunicación, el control telemático ha visto un crecimiento importante los últimos años, ya que gracias al uso del espectro radioeléctrico se puede usar diferentes tecnologías para implementar el mismo. En los artículos estudiados, en 2 (10% del total) se utiliza la radiofrecuencia como método de comunicación ya que su propósito era netamente investigativo (Brito, y otros, 2019; Trunov, Voronova, Voronov, Sukhachev, & Strelnikov, 2018); en otros 2 (10% del total) se utiliza una Red de Área Local (LAN), ya que eran aplicaciones con propósitos específicos y orientados a empresas, una orientada al control distribuido y eficiente de la energía (Huang, Zhuang, Sun, & Zhang, 2020) y la otra especializada en la manufactura de calzado y en desarrollar un proceso óptimo para despachar los pedidos (McNally, Chaplin, Martínez, & Ratchev, 2020); en otros 3 artículos (15% del total) se utiliza redes WAN, específicamente en aplicaciones orientadas a Smart Cities, se mencionan tecnologías como LORA, una técnica de modulación de espectro ensanchado de baja potencia que funciona muy bien en redes WAN (Badii, y otros, 2019), también se menciona a SigFox, un servicio de comunicaciones inalámbricas, que utiliza la banda ISM por su gran alcance y su facilidad de atravesar objetos sólidos, además requiere poca energía por lo que se denomina red de área ampla de baja potencia (LPWAN) (Kramer, Frese, & Kuijper, 2019), y se mencionan servicios como Ingenu, orientados a la comunicación inalámbrica M2M para que puedan convertirse y considerarse dispositivos IoT (Lakhan & Li, 2019).

Finalmente, en 13 artículos (65% del total), la tecnología más utilizada es el Internet, con sus diferentes protocolos de comunicación, se menciona los términos de

Cloud Computing o Fog Computing como administradores de la utilización de los dispositivos, denominados como EDGE (Alam, y otros, 2018), además por lo general se menciona el uso servicios web y la utilización de arquitecturas RESTful por su facilidad de comunicación entre componentes gracias al protocolo de comunicación HTTP y sus solicitudes en formato JSON, que pueden ser comprendidas tanto en el lado del cliente como del servidor, logrando una interacción adecuada entre los servicios correspondientes en el back-end y permitiendo que se reflejen las respuestas adecuadas en el front-end (Richter, Konrad, Utecht, & Polze, 2017).

Q5: ¿Qué técnicas y/o tecnologías se utilizan para determinar la QoS en cuanto al control de sistemas ciber físicos?

La calidad de servicio o QoS (del inglés Quality of Service) es muy importante en la comunicación con los sistemas ciber físicos, y más aún si se desea controlarlos mediante internet, ya que con esto se garantiza el tráfico y la cantidad de datos y paquetes que pueden llegar de extremo a extremo. En los artículos mencionados utilizan múltiples métodos para poder garantizar y determinar la calidad de servicio dependiendo de la aplicación, en algunos se está pendiente de la QoS en cada microservicio y para medirlo usan métricas establecidas por los desarrolladores y en caso de que haya algún disparador que indique que estos datos sean incorrectos, el sistema procede a corregir y enviar una respuesta adecuada para el usuario (Yue, Wu, & Xue , 2020); en otros artículos se basan más en la experiencia del usuario como tal y para garantizar la calidad de servicio, algunos verifican los datos con un microservicio específico para esta tarea y de ser correctos se procede a la ejecución del requerimiento, caso contrario se notifica al usuario de que debe volver a realizar la acción, siendo así un proceso iterativo para cada usuario (Trunov, Voronova, Voronov,

Sukhachev, & Strelnikov, 2018; Badii, y otros, 2019); otros estudios usan la retroalimentación de los usuarios como garantía de la QoS, ya que por ejemplo en aplicaciones como las videollamadas, se puede valorar estas métricas directamente con una encuesta de aceptación al usuario una vez la llamada termine (UAT, del inglés User Acceptance Test).

Finalmente, algunos artículos mencionan que la QoS puede ser garantizada aplicando las métricas adecuadas, ya que si se garantizan las métricas que dependen netamente de la empresa, la cuales son las de rendimiento y eficiencia de los componentes y los equipos, la métrica de satisfacción de usuario será siempre positiva (Zheng, Zhang, Zheng, Fu, & Liu, 2017; Laleh, Paquet, Mokhov, & Yan , 2018); en un artículo se menciona que hay plataformas que sirven para garantizar una calidad de servicio optima, como el caso de Asperathos, especializada en facilitar el desarrollo y control de aplicaciones que corran en entornos Cloud (Brito, y otros, 2019).

Otras conclusiones

Además de haber respondido las preguntas que fueron planteadas, se han extraído otras conclusiones que pueden ser muy relevantes para el desarrollo del presente trabajo:

- Al igual que en el SMS, este SLR ayudo para evidenciar que tanto la temática de *Tolerancia a fallos* como la de *Control telemático* son un campo de estudio de interés dentro de la comunidad científica, además de que existen muchos enfoques y tecnologías que pueden utilizarse para su implementación.
- Se pudo evidenciar que los artículos abordaban con mucho detalle el proceso tanto para implementar la *Tolerancia a fallos* como el *Control telemático,* lo

que es de gran ayuda para el desarrollo del presente trabajo, ya que son dos de las temáticas principales.

- Se evidenció que la aparición de las arquitecturas de microservicios ha ayudado tanto al desarrollo de aplicaciones que requieren la tolerancia a fallos como uno de sus ejes principales, como también a la prueba de desempeño de esta herramienta, ya que en estructuras monolíticas no se podía hacer un benchmarking fiable ni de la tolerancia a fallos ni de las arquitecturas como tal.
- Al igual que en el SMS, en este SLR se pudo evidenciar que la tolerancia a fallos y el control telemático son importantes en todos los campos de aplicación, pero si es verdad que a lo largo de este estudio se pudo apreciar que las investigaciones van más orientadas a ámbitos que en un futuro serán muy populares y comerciales, como el caso de las ciudades inteligentes, el control de robot autónomos, y la automatización de locales comerciales.
- Si bien se pudo apreciar que el desarrollo tecnológico ha permitido la interconexión de componentes de manera inalámbrica, el medio o la herramienta más utilizada, es sin lugar a dudas el Internet, ya que es lo más accesible tanto para implementar un servicio como para poder llegar a un público objetivo sin la necesidad de tener un conocimiento muy profundo y detallado de su funcionamiento.
- Los entornos WoT/IoT se han popularizado exponencialmente los últimos años y es debido a que las comunicaciones inalámbricas ahora permiten una interconexión segura entre componentes tanto de software como de hardware con configuraciones sencillas, por lo que la proyección es que en un futuro

todo pueda estar interconectado a la nube, y esa es la principal razón de la importancia de una buena gestión y control telemático.

- La calidad de servicio, ha estado presente desde la existencia que apareció el concepto de “cliente” y siempre se ha buscado la satisfacción del mismo en todos los aspectos, la única diferencia es que ahora QoS se refiere más al ámbito tecnológico, pero siempre buscando optimizar el servicio y garantizarle una buena experiencia de uso al usuario.
- Todos los artículos estudiados son pertenecientes a Conferencias o revistas indexadas pertenecientes a las áreas de interés.
- Otra conclusión importante que se extrajo de este SLR es qué no fue muy complicado responder las preguntas planteadas, esto debido a que la temática de estudio de las arquitecturas, se trata de desarrollo de software o de sistemas estandarizados, y ya que estos deben cumplir ciertas reglas y normas para su correcto funcionamiento, la tolerancia a fallos y el control telemático son dos cosas que están de cierto modo implícitas en esta tecnología.

Como conclusión final se puede decir que las implementaciones y trabajos desarrollados son ad-hoc para tareas concretas y aplicaciones específicas, sin embargo, es importante notar la versatilidad que pueden tener este tipo de sistemas y arquitecturas, y también el detalle al usar ciertas herramientas en concreto, como en este caso para la implementación de la *Tolerancia a fallos* y el *Control Telemático*, esto definitivamente ha ayudado a determinar cuáles son las herramientas que más conviene utilizar en el diseño, desarrollo e implementación del presente proyecto, las cuales se definirá en la siguiente sección.

Definición de las características del sistema

A partir de los estudios anteriores, tanto realizados en el SMS como en el SLR, se ha observado que existen múltiples tecnologías en cuanto al desarrollo de las arquitecturas de microservicios como en la implementación de sistemas ciber físicos y todos los componentes necesarios, por lo cual es importante fundamentar y definir cuáles son las herramientas que se utilizaron y las características generales que tiene el producto final.

Arquitectura de microservicios

En el desarrollo de software, estás arquitecturas han sido un gran avance al momento de construir una aplicación e ir dividiéndola en pequeños servicios, los microservicios, los cuales van ejecutando sus procesos propios y se comunican entre ellos con diferentes mecanismos. Gracias a los estudios realizados se pudo apreciar que no todas las implementaciones siguen los mismos patrones o características, pero entre las más comunes se encontraron las siguientes:

- **Sus componentes son servicios:** Un componente de software debe ser remplazable, actualizable y sobre todo independiente, gracias a los microservicios esto se cumple adecuadamente, ya que no hay la dependencia de una biblioteca y un requerimiento de memoria, sino que los servicios pueden controlados remotamente, ya sea mediante un servicio web o un RPC (llamada a procedimiento remoto).
- **Enfocada a la funcionalidad de un negocio:** Con los microservicios se puede lograr una distribución adecuada de un negocio, ya que el sistema va a estar organizado de tal manera que cada servicio tenga una responsabilidad específica.

- **Orientada a productos:** Por lo general siempre se trata al desarrollo de alguna aplicación como un proyecto con un propósito específico, la diferencia de la arquitectura de microservicios es que además de cumplir con su finalidad, también va orientada a ayudar al usuario a mejorar la experiencia que tenga usando la aplicación.
- **Descentralizada:** Al tratarse de una arquitectura con múltiples servicios colaborativos, permite utilizar diferentes lenguajes de programación, herramientas y tecnologías dentro de cada servicio. De esta manera s puede escoger la herramienta más adecuada para cada tipo de trabajo y no estar regidos por una única estructura de diseño y desarrollo.
- **Datos descentralizados:** Al igual que en el punto anterior, cada microservicio puede gestionar su propia base de datos, ya sea en la misma instancia o en sistemas completamente diferentes.
- **Tolerante a fallos:** Como ya se mencionó reiteradas veces en este documento, las arquitecturas de microservicios están diseñadas de tal manera que de ser el caso y falla alguna dependencia, el cliente sea capaz de responder y evitar una falla en cascada del sistema, para esto hay diferentes patrones que se pueden utilizar: tiempos de espera extendidos, reintento, cola de reintento, circuit breaker, compartimentos adicionales.
- **Automatización de infraestructura:** En la mayoría de los artículos estudiados se menciona que las arquitecturas de microservicios son de gran utilidad en la actualidad y esto gracias a que la mayoría de productos y sistemas que se han ido desarrollando están orientados a

automatizar procesos, en cualquier ámbito y campo de aplicación, ya sea industrial, comercial, social, etc.

- **Diseño evolutivo:** La gran ventaja de este tipo de arquitecturas es la facilidad con la que se las puede mejorar, remplazar o actualizar, gracias a que sus servicios funcionan independientemente y cada contendor puede ser alterado de tal manera que no afecte la experiencia de los consumidores.

Una vez detalladas todas las características de este tipo de arquitectura, es importante especificar la herramienta que sirvió para su implementación, como se pudo ver existen muchas herramientas que pueden servir a este propósito, pero la seleccionada para el desarrollo del presente trabajo junto a todos sus componentes y prestaciones, es *Spring Boot,* un framework de desarrollo de código abierto basado en Java que está orientado específicamente a crear microservicios y que permite compilar aplicaciones web como archivos ejecutables .jar, además funciona para integrar los servicios ya sea por coreografía u orquestación, siendo esta última la que se va a utilizar, donde un microservicio estará encargado de guiar y dirigir al resto de los procesos.

Tolerancia a fallos

Como se pudo ver en el punto anterior, la tolerancia a fallos es algo intrínseco de las arquitecturas de microservicios gracias a su diseño y estructuración, pero es importante mencionar algunas de sus características principales para tener claro que es lo que se busca con su implementación:

- **No deben existir puntos de falla:** Esto quiere decir que en caso de que el sistema sufra algún fallo, debe seguir operando sin interrupciones durante el proceso de reparación.
- **Aislamiento de componentes que están fallando:** El sistema debe poder aislar el fallo hacia una locación específica, para que no existan reincidencias, por lo que es importante incorporar mecanismos de detección de fracaso y que puedan clasificar de que tipo son los mismos.
- **Contener fallos:** Es importante tener mecanismos de apoyo que eviten que se provoque un fallo en cascada.

Como se mencionó en el apartado anterior, existen diferentes patrones que se pueden utilizar en la tolerancia a fallos: tiempos de espera extendidos, reintento, cola de reintento, circuit breaker, compartimentos adicionales. El patrón que se utilizó en este trabajo es *Circuit Breaker* y la tecnología para implementarlo es *Hystrix*, una librería desarrollada y ofrecida por Netflix, diseñada para aislar puntos de acceso a sistemas remotos, servicios y librerías de terceros, permite gestionar diversos aspectos tales como: timeouts, estadísticas de éxito y fallo, semáforos, lógica de gestión de error, lógica de latencia y de esto modo deteniendo fallos en cascada, mejorando la resiliencia en sistemas complejos distribuidos donde la probabilidad de fallo es inevitable, además su compatibilidad con Spring Boot es una ventaja que se puede utilizar a favor.

Sistema ciber físico

Un sistema ciber físico a diferencia de un sistema tradicional, tiene una característica principal y es que está diseñado de tal manera que muchos elementos están conectados en una red, donde interaccionan entre si y tienen una disponibilidad

de entradas y salidas físicas a diferencia de ser dispositivos aislados. Para su implementación existen un sin número de herramientas, tecnologías y dispositivos dependiendo de la aplicación. Para el presente trabajo se utilizó un computador de tamaño reducido que, gracias a sus prestaciones y a su versatilidad de aceptar múltiples sistemas operativos y lenguajes de programación, permite una interacción adecuada con la arquitectura desarrollada con Spring Boot, además sus múltiples puertos de entrada y salida, hacen que se pueda integrar diversos sensores y actuadores que sirvieron para el propósito final de lo que se busca en esta tesis.

Control telemático

El desarrollo tecnológico en el área de las telecomunicaciones, y los avances y estudios en las redes de comunicación han permitido que actualmente existas múltiples medios para la interconexión y control de sistemas y dispositivos, así como la red celular, la red ISM, y las más popular y la que se utilizó en el desarrollo de este proyecto, el Internet, la facilidad de utilización que brinda esta tecnología es muy superior a muchas otras, y más aún si lo que se quiere implementar es un servicio web ya que gracias a su familia de protocolos como TCP/IP o HTTP permiten una trasmisión de datos adecuada y funcionan mejor en arquitecturas o servicios web tipo RESTful, que es lo que se implementó para que pueda existir una comunicación tanto en los componentes de software como de hardware.

Capítulo II

Tecnologías Relacionadas

Arquitecturas de microservicios

El eje principal del presente trabajo son las arquitecturas de microservicios, por lo que es importante mencionar los diferentes tipos de estructuras y variantes que existen, además de las tecnologías más utilizadas actualmente que ofrecen las mismas funcionalidades y siguen los mismos principios de SOA, pero que no necesariamente son lo mismo.

SOA (Service Oriented Architecture)

Las arquitecturas orientadas a servicios son un concepto de software que como su nombre lo dice utiliza servicios (programas, rutinas o secuencias específicas) diseñadas para cumplir los requisitos de un negocio o empresa. SOA permite crear sistemas amplios, flexibles y versátiles que ayudan a mejorar el rendimiento y a reducir los costos de un proceso de empresa o negocio, además la forma en la que invocan sus servicios facilita la interacción tanto de sus componentes internos como los de terceros. (eCityclic, 2020). En la figura 14 se puede apreciar como SOA ayuda en la integración de los sistemas que pueden integrar un modelo de negocio o empresa.

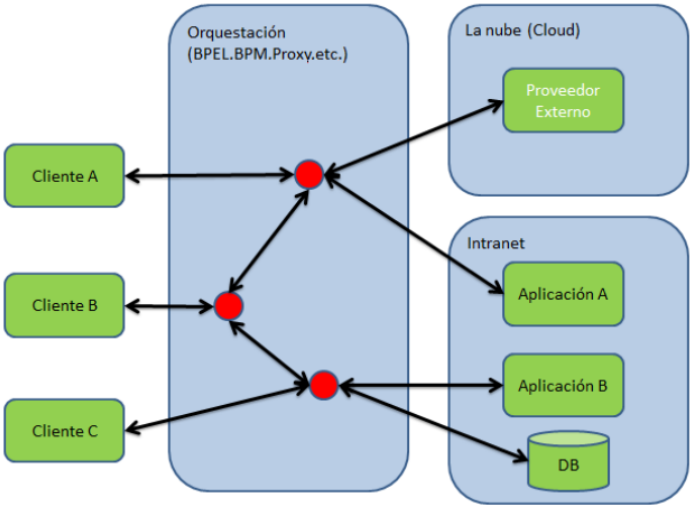
Características de SOA. Entre los beneficios principales de SOA se encuentras algunas de sus características principales que son:

- Permite reutilizar componentes.
- Permite reutilizar los mismos procesos para crear otros servicios.
- Permite disponer de más servicios en menos tiempo.

- Cada servicio debe tener ciertos requerimientos que lo identifiquen dentro de un catálogo que sea comprensible y reutilizable.
- Finalmente, SOA carece de individualidad, ya que puede ser un conjunto de servicios o bien un servicio para un sistema más grande. (eCityclic, 2020).

Figura 14

Arquitectura orientada a servicios



Nota: En la figura se puede apreciar el proceso de integración que puede brindar SOA dentro de una empresa. Tomado de (Blancarte, Oscar Blancarte Software Architect, 2014).

Ventajas de SOA. Además de las importantes características que brinda, SOA también tiene ventajas que brindan una interoperabilidad entre aplicaciones que sean heterogéneas, entre las otras ventajas de SOA son:

- Aumenta la eficiencia en todos los procesos de la empresa.

- Reduce los costos de inversión realizada en los sistemas.
- Reduce costos de mantenimiento.
- Facilita que los sistemas puedan cambiar gracias a la herencia de los mismos.
- Fomenta la innovación del desarrollo de los sistemas ya que va adaptándose al dinamismo del mercado.
- Simplifica diseño y optimiza la organización (eCityclic, 2020).

Desventajas de SOA. Dentro de las desventajas que se tiene en las arquitecturas orientadas a servicios, se tiene algunas desventajas y sobre todo consideraciones que se debe tener en cuenta:

- Se debe ser muy exigente con el servicio escogido, ya que se debe evitar los extremos y cumplir toda la coherencia.
- Se debe entender los servicios como algo limitado y no como una aplicación totalmente completa.
- Se necesita la mayor simplicidad posible al momento de diseñar la arquitectura.
- Se debe garantizar una alta disponibilidad y escalabilidad de los servicios.
- Si un servicio está mal diseñado, puede desencadenar en una falta de disponibilidad.
- Si se tiene múltiples equipos que compartan un recurso, puede que el mismo no esté disponible por un tiempo.
- Si un servicio presenta un error, todo el sistema puede presentar una restricción de tiempo que perjudicará en la calidad que se brinda.

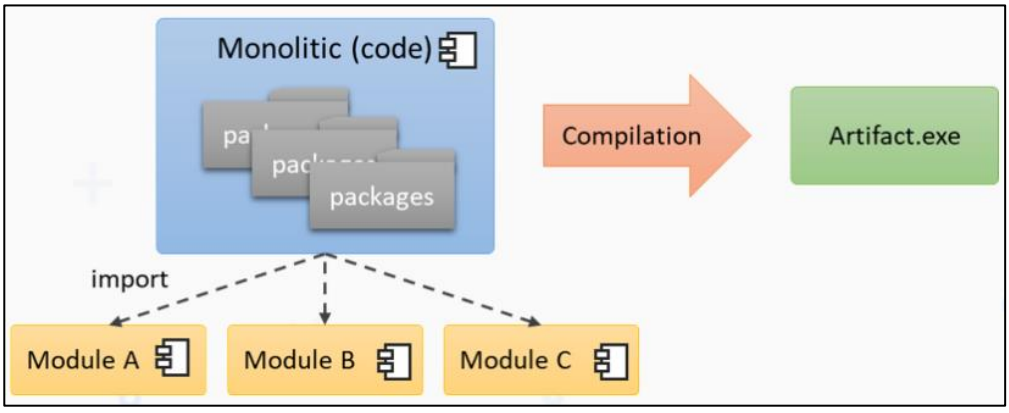
- Si un servicio dependiente varia, invalidará los flujos presentes y también incide en la consistencia de todos los datos (eCityclic, 2020).

Arquitectura Monolítica

Este tipo de arquitectura de software consiste en desarrollar una aplicación que contenga absolutamente todas las funcionalidades necesarias para realizar las tareas a las que va orientada en un solo apartado de memoria, es decir sus componentes trabajan juntos y comparte los mismos recursos, en este sentido se puede decir que es una unidad cohesiva de código. Un monolito puede estar construido como una sola unidad de software o estar constituido de varios módulos o librerías, pero lo que lo distingue es que, al momento de su compilación, todo es empaquetado como una única pieza, es decir sus módulos y librerías junto con la aplicación principal, esto puede apreciarse en la figura 15 (Blancarte, Oscar Blancarte Software Architect, 2017).

Figura 15

Arquitectura Monolítica.



Nota: En la figura se puede apreciar el proceso de compilación de una aplicación basada en una arquitectura monolítica. Tomado de (Blancarte, Oscar Blancarte Software Architect, 2017).

Características de una arquitectura monolítica. A continuación, se analiza las características que sirven para determinar y distinguir cuando una aplicación está definida por este estilo en específico:

- Además de sus dependencias de compilación, las aplicaciones basadas en esta arquitectura son autosuficientes, es decir no requieren de nada para funcionar.
- Para poder realizar una tarea, se realizan todas las operaciones necesarias en cada uno de los módulos de extremo a extremo.
- Generalmente son aplicaciones grandes, pero esto no es un requerimiento.
- Son por lo general una unión datos privados. Donde cada instalación administra su propia base de datos.
- La totalidad de la aplicación corre sobre una única plataforma (Blancarte, Oscar Blancarte Software Architect, 2017).

Ventajas de una arquitectura monolítica. Las arquitecturas monolíticas son fáciles de desarrollar ya que solo existe un componente a implementar, entonces es sencillo para un equipo pequeño iniciar el proyecto y ponerlo en marcha rápidamente. Es muy fácil de escalar instalando la aplicación en varios servidores y utilizando un balanceador de cargas. Gracias a su independencia de factores externos, mitiga gran parte de los errores que podrían producirse en ámbitos de comunicación, red, integraciones, etc. Pueden ser completamente autónomas, ya que pueden funcionar aparte del resto de aplicaciones. Al estar estructuradas en base a un solo componente, las aplicaciones monolíticas tienen un desempeño mucho mejor y más rápido, ya que no necesitan ocupar recursos distribuidos para completar algún proceso. Finalmente

son fáciles de probar gracias a que su funcionalidad se encuentra disponible de principio a fin y además de esa única unidad código, no se necesita ninguna otra dependencia (Blancarte, Oscar Blancarte Software Architect, 2017).

Desventajas de una arquitectura monolítica. Debido a que su software es una pieza única, esto implica que se utilice solo una tecnología de desarrollo e impide que se puedan aprovechar de todas las tecnologías disponibles. Si se escala una parte, se debe escalar toda la aplicación, gastando recursos innecesarios para su funcionalidad. Mientras la aplicación va creciendo de manera exponencial, entonces se necesita un equipo de desarrollo más grande y es muy difícil que se pueda dividir el trabajo sin afectar en la funcionalidad que algún otro miembro del equipo este modificando. Con cada modificación se debe volver a compilar toda la aplicación, por lo que debe manejarse por versiones. Si en el desarrollo no se dispone de una alta disponibilidad y nodos de redundancia, si falla algo, falla absolutamente todo, quedando el sistema inoperable. Debido a su estructura unitaria, es muy fácil caer en prácticas poco adecuadas ya que organizar el trabajo tiende a ser muy complicado. Finalmente, si se quiere hacer mantenimiento de alguna aplicación ya implementada, puede ser muy abrumador para algún desarrollador que no la conozca a fondo. (Blancarte, Oscar Blancarte Software Architect, 2017)

Arquitectura de Microservicios

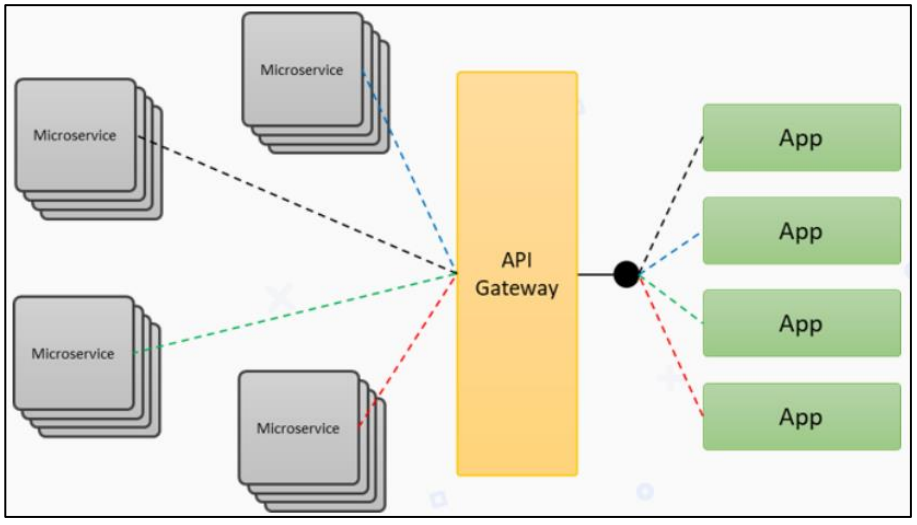
Este tipo de arquitectura se ha tornado en una de las más populares en los últimos años. Consiste en crear pequeños componentes encargados de realizar una tarea en específico, son autosuficientes y no dependen del resto de componentes de la aplicación, en este sentido se puede decir que los microservicios son lo opuesto a las

arquitecturas monolíticas ya que lo que se busca con estos es desmenuzar una aplicación en algunos y pequeños componentes que realicen tareas específicas que ayuden a la problemática general.

Como se puede ver en la figura 16, las arquitecturas de microservicios lo que buscan es dividir toda la funcionalidad pequeños componentes que puedan ser independientes y autosuficientes, a diferencia de las arquitecturas monolíticas que implementan todo en una sola unidad, además en los microservicios es común observar un componente llamado API Gateway, el cual es encargado de permitir el acceso y de controlar al resto de microservicios (Blancarte, Oscar Blancarte Software Architect, 2018).

Figura 16

Arquitectura de Microservicios.



Nota: En la figura se puede apreciar la estructura de una aplicación basada en una arquitectura de microservicios. Tomado de (Blancarte, Oscar Blancarte Software Architect, 2018).

Características de una arquitectura de microservicios. Una característica de los microservicios es que deben poder comunicarse unos con otros para poder realizar el proceso completo de alguna aplicación en específico, además cada uno puede operar independientemente bajo su formato o estilo de programación y con su propia base de datos. Algo importante que mencionar es que los microservicios funcionan bajo una arquitectura distribuida, esto quiere decir que cada uno puede ser desplegado y funciona independientemente del resto de los componentes. Para poder acceder a la información y requerimientos de cada microservicio, se genera una cola de mensajes que pueden ser generados a partir de diferentes protocolos, ya sea SOAP, REST, RPC, etc. Esto permite que, si se desea realizar algún cambio, un microservicio no afecte a los demás que están funcionando al mismo tiempo (Blancarte, Oscar Blancarte Software Architect, 2018).

Ventajas de una arquitectura de microservicios. Entre las ventajas de las arquitecturas de microservicios se puede mencionar que es altamente escalable, ya que permite implementar varias instancias del mismo componente y balancear la carga entre ellas. Es mucho más ágil ya que todos los componentes tienen un tiempo de desarrollo independiente y diferente, lo que permite desplegarlos más rápidamente y sin interferir en la operación del resto. Son fáciles de probar y de desplegar ya que no dependen de fuentes externas ni de requerimientos de hardware específicos. Su desarrollo no es complicado gracias a su independencia y su alcance, esto ayuda a los programadores a comprender su funcionamiento rápidamente y no hay la necesidad de grandes equipos de trabajo. Lo más importante es que son reusables e interoperables, ya que los componentes puedes ser utilizados en diferentes aplicaciones si se desea y pueden

funcionar adecuadamente ya que utilizan estándares abiertos y no importa la tecnología bajo la que estén desarrollados (Blancarte, Oscar Blancarte Software Architect, 2018).´

Desventajas de una arquitectura de microservicios. En cuanto a las desventajas que se puede encontrar al implementar una arquitectura de microservicios, se puede mencionar que el desempeño quizá no sea el óptimo, ya que los microservicios al estar distribuidos pueden agregar latencias significativas a aplicaciones donde el tiempo de respuesta sea lo más importante. Debido a los múltiples componentes de una aplicación, esta se ve en riesgo de tener múltiples puntos de falla, lo cual debe ser gestionado de manera adecuada. La funcionalidad no es necesariamente lineal, por lo que no se puede hacer una trazabilidad y seguimiento ordenado del proceso, ya que cada microservicio arroja logs por separado que después deben juntarse para tener una traza completa. Finalmente, para implementar una arquitectura de microservicios robusta, es importante tener un equipo de desarrollo especializado, ya que todos los componentes y tecnologías que se agregan deben ser administrados de una manera adecuada, algo que se torna muy complicado para un equipo de desarrollo con poca experiencia (Blancarte, Oscar Blancarte Software Architect, 2018).

SOAP (Simple Object Access Protocol)

SOAP es un protocolo ligero desarrollado para el intercambio de información en entornos descentralizados y distribuidos. Está basado en el protocolo XML que consiste de tres partes principales: un sobre (envelope) que define un marco de trabajo para identificar un mensaje y saber cómo procesarlo, un conjunto de reglas de codificación para poder expresar tipos de datos definidos por la aplicación y una convención para

representar llamadas y respuestas para procesos remotos. SOAP puede sr utilizado junto con múltiples protocolos, pero por lo general se lo utiliza con HTTP o HTTP Extension Framework. (World Wide Web Consortium (W3C), 2007).

Características de SOAP. Además de las partes principales que conforman a SOAP, este dispone de algunas características que son esenciales:

- **Extensibilidad:** Gracias a la seguridad y a su protocolo embebido de Web Services Routing Protocol (WS-Routing) que están presentes en su desarrollo.
- **Neutralidad:** Al estar orientado a aplicaciones de extremo a extremo, funciona bajo los protocolos de comunicación y transporte TCP/IP, esto permite que interactúe y funcione adecuadamente con otros protocolos como JMS, HTTP o SMTP.
- **Independencia:** Esto gracias a que permite cualquier tipo y modelo de programación.

Además, una arquitectura SOAP está formada por varias capas de especificación:

- MEP (Message Exchange Patterns) orientada al formato del mensaje.
- Capa de enlaces subyacentes de protocolos de transporte.
- Capa de modelo para procesamiento de mensajes.
- Capa de extensibilidad orientada a protocolos.

Finalmente, gracias a la unión de todas estas características, se puede decir que un mensaje SOAP es un documento XML común, pero tiene una estructura definida en la especificación del protocolo (World Wide Web Consortium (W3C), 2007).

Ventajas de SOAP. Entre sus ventajas y como ya se mencionó anteriormente, SOAP permite la utilización de diferentes protocolos como HTTP, SMTP o JMS, esto gracias a que funciona bajo el protocolo de transporte TCP.

También brinda la posibilidad de generar aplicaciones cliente/servidor, desarrollados en distintos lenguajes de programación, que gracias a su amplia estandarización deben cumplir con reglas concretas para formar el mensaje, ya sea un contrato entre cliente y servidor o un simple mensaje, pero siempre en formato XML.

Debido a estas cualidades, SOAP es ampliamente utilizado en aplicaciones o entornos empresariales donde sea requerido un contrato claro y además se disponga de una fiabilidad y seguridad en cuanto a las comunicaciones. (Chakray, 2020)

Desventajas de SOAP. Debido a su amplia estandarización, SOAP es poco flexible y suele haber muchos errores de desarrollo, ya que no se conocen dichos estándares.

La utilización de TCP junto con los estándares propios del protocolo, lo han hecho tener un peor rendimiento frente a tecnologías más actuales orientadas a Web Services.

Debido a estos motivos por lo general para desarrollar APIs SOAP, es importante contar con productos, herramientas adicionales o empresas de consultaría especializadas que ayuden a gestionar y crear un producto de este tipo desde cero (World Wide Web Consortium (W3C), 2007).

REST (REpresentational State Transfer)

REST es un estilo de arquitectura de software desarrollado para sistemas distribuidos como la World Wide Web, donde se requiere una cantidad de datos masivos de diferentes tipos, utiliza directamente HTTP tanto para la obtención de datos como la ejecución y procesamiento de los mismos en cualquier formato (XML, JSON, etc.). (Wikipedia, 2020)

A diferencia de SOAP, REST soluciona el inconveniente de desarrollar un servicio complejo de gestionar ya que solo trabajaba con XML, haciendo mucho más fácil el desarrollar una API REST. Además, REST se apoya en HTTP y utiliza los mismos verbos que son GET, POST, PUT y DELETE, finalmente los datos son almacenados como una serie de recursos URL que pueden ser limitados por el desarrollador dependiendo de lo que busque en su BACKEND. (Rosa Moncayo, 2018)

Características de REST. Como se mencionó, la APIs REST se distinguen porque se basan fuertemente en el protocolo de comunicación HTTP, esto gracias a que utilizan sus métodos de envió y de respuesta para funciones específicas, además su funcionamiento está basado en algunas características esenciales:

- Se maneja por solicitudes o peticiones HTTP, es decir se envía un REQUEST a un servidor, el cual dispone de toda la información necesaria para dar una respuesta en concreto, RESPONSE.
- Al basarse en HTTP que es un protocolo consolidado, no necesita modificaciones o invenciones nuevas.
- Utiliza los mismos métodos básicos de HTTP:
  - DELETE: Borrar un recurso o un dato.
  - GET: Obtener recurso en concreto.

- PATCH: Modificar un recurso que no sea un dato.
- POST: Crear nuevos recursos.
- PUT: Modificar recursos y datos.

- Todos sus objetos se manejan mediante URIs, así es mucho más fácil y conveniente obtener información de algún usuario o recurso en específico. (Rosa Moncayo, 2018)

Ventajas de REST. La utilización de REST ha visto un amplio crecimiento ya que, para empezar, permite separar el cliente del servidor, esto quiere decir que se los puede desarrollar en diferentes lenguajes y no tienen por qué estar desarrollados bajo el mismo paradigma.

Su notable crecimiento ha hecho que su comunidad online en plataformas como GitHub crezca de gran manera. Permite crear diseños de microservicios orientados a dominios (DDD) y además que se tenga la posibilidad de enviar y recibir la información en cualquier tipo de formato, aunque lo más habitual sea JSON. Es independiente de cualquier plataforma, entonces puede ser utilizada ya sea en Windows, Mac, Linux, o el SO que se desee.

Finalmente brinda mucha más escalabilidad, ya que, separa cliente y servidor lo que es muy adecuado al momento de valorar su rendimiento en comparación de SOAP, ya que REST tiende a ser mucho más ligero y apropiado gracias a sus características y a la utilización de HTTP. (Rosa Moncayo, 2018; Chakray, 2020)

Desventajas de REST. Entre las desventajas de REST se puede decir que, a pesar de la utilización de HTTP dentro de su desarrollo, no existe un estándar definido que indique como se deben utilizar dichos métodos. Además, una API REST no genera un contrato por defecto entre el cliente y servidor, sino que es un añadido por lo que es menos segura que SOAP. Otro de los grandes inconvenientes es el manejo de grandes

cantidades de datos o de la imposibilidad de realizar más de una llamada de conexión, aunque esto último puede ser solventado implementando al mismo en servicios RESTful (Chakray, 2020).

SOAP vs REST

A pesar de que SOAP ha sido la opción preferida para muchas aplicaciones y entornos empresariales, resulta ser muy compleja y poco flexible. Es por esto que en los últimos años se ha ido migrando a servicios basados en REST y gracias a sus actualizaciones e interoperabilidad de tecnologías ahora puede manejar cantidades de datos masivas. Ambas arquitecturas tienen nichos definidos, pero aparentemente REST será la que tendrá mayor aceptación a futuro, sin embargo, es importante mencionar las ventajas y prestaciones que brindan las dos, como se puede ver en la tabla 7.

Tabla 7

Ventajas de SOAP vs Ventajas de REST.

	Ventajas SOAP	**Ventajas REST**
Desarrollo	Muchas operaciones con pocos recursos.	Pocas operaciones con muchos recursos.
Enfoque	Diseño de aplicaciones distribuidas.	Escalabilidad y rendimiento para sistemas distribuidos de hipermedia a gran escala.
Protocolos	SMTP, JMS, HTTP POST.	HTTP (POST, GET, PUT, DEL)
Formato de datos	Tipado fuerte, XML.	XML, JSON.
Sincronismo	Síncrono y asíncrono.	Síncrono
Seguridad	WS Security.	HTTPS
Comunicación	Origen a destino.	Punto a punto.

Tecnologías de Back-End

En desarrollo de software y desarrollo web, el Backend es la parte que se encarga de que toda la lógica desarrollada para qué la aplicación funcione de manera correcta, son todas las acciones que no son perceptibles por el usuario, como por ejemplo la comunicación con el servidor, él envió de información a la base de datos.

En el desarrollo del backend se debe ser muy meticuloso y tener mucho cuidado, ya que un pequeño error puede provocar que toda la aplicación se caiga, entre algunas funciones que se realizan en el backend, se pueden encontrar las siguientes:

- Desarrollo de funciones que optimicen el proceso.
- Conexión y envío de información a las bases de datos.
- Acciones de lógica que funcionan con diferentes módulos y librerías.

Todo desarrollador de backend debe estar especializado con diferentes lenguajes de programación y tecnologías relacionadas, y mucho más en arquitecturas de microservicios que son las que pueden abarcan múltiples lenguajes y tecnologías (Arjonilla, 2017). En cuanto a las tecnologías más utilizadas actualmente se tiene algunas como: PHP (Hipertext Preprocessor) es un lenguaje de código abierto muy popular en el desarrollo de aplicaciones web y especialmente popular ya que puede ser incrustado en HTML. A diferencia de muchos otros lenguajes que utilizan HTML nativo para mostrar el contenido, PHP permite que esto se lo haga de una manera mucho más rápida y reducida (The PHP Group, 2020). Ruby que es un lenguaje balanceado, fue creado a partir de muchos otros lenguajes como Perl, Smalltalk, Eiffel, Ada y Lisp, generando un lenguaje funcional e imperativo. Para Ruby todo es tratado como un objeto pudiendo asignar propiedades y acciones a todas las partes del código. Es considerado altamente flexible, ya que permite a todos los usuarios alterarlo libremente, además dispone de bloques diseñados para añadir cláusulas de cómo debe actuar cada

método (Ruby, 2020). Python que es un lenguaje de programación que se ha popularizado debido a la simpleza en su código y a la fácil legibilidad del mismo, es multiparadigma ya que en cierta parte soporta parcialmente la orientación a objetos, programación imperativa y programación funcional (Wikipedia, 2021). Pero los lenguajes que se van a profundizar a continuación son los más relevantes para el presente trabajo:

JAVA

Es una tecnología utilizada para que las aplicaciones dentro de la Web sean más útiles e interesantes. Es importante mencionar que no es lo mismo que *JavaScript,* una tecnología más sencilla y netamente para exploradores Web.

Java permite a los usuarios muchas de las tareas que actualmente se las hacen a diario, como jugar online, chatear, subir fotografías, realizar visitas virtuales, además es muy utilizada en los servicios actuales como, por ejemplo, cursos en línea, banca en línea, mapas interactivos, etc.

Finalmente, Java es la base para prácticamente todas las aplicaciones de red, además se utiliza como estándar para poder compilar y distribuir aplicaciones móviles, juegos, contenido web y softwares empresariales. Adicionalmente Java ha comenzado a tener un valor innegable debido a que permite:

- Desarrollar software en una plataforma y ejecutarla virtualmente en una diferente.
- Desarrollar aplicaciones web que funcionan con lenguajes de tipado como HTML.
- Combinar aplicaciones desarrolladas bajo Java y que permiten gran nivel de personalización.

- Permite desarrollar software potente que puede adecuarse a cualquier tipo de dispositivo electrónico, como celulares, sensores, módulos inalámbricos, etc (Oracle, 2020).

JavaScript

JavaScript es un lenguaje de programación interpretado, se define como multiparadigma por ser orientado a objetos, imperativo y dinámico. En el desarrollo de frontend tiene dos funciones principales, la primera es que sirve para interactuar con los usuarios, ya que gracias a JavaScript se puede escuchar y entender las acciones que realizan los usuarios en la interfaz gráfica y la segunda función de JavaScript es que sirve para comunicarse con el Backend, sirve como un puente para los usuarios que entregan los datos a través de una interfaz gráfica y JavaScript los envía hacia el backend para poder responder adecuadamente a sus requerimientos. Para esto se utiliza un API, que es donde el frontend hace peticiones de envió de información y para recibir información (Castro, 2021).

Así como para el frontend, si se quiere usar JavaScript para el Backend hay que tener algunas consideraciones. En primer lugar, es necesario utilizar Node.js, un entorno de software que ejecuta JavaScript y con la ayuda de algunas librerías, permite la entrada y salida de datos en stream y da soporte a peticiones como las de HTTP (Bliss, 2018). En segundo lugar, en necesario entender todas las ventajas y características que brinda la utilización de Node.js y JavaScript en el backend:

- Node.js utiliza un Event Loop, un constructor que pone en espera y va despachando tareas o procesos, y esto sin generar un bloqueo, sino que las tareas simplemente se van apilando en el stack y se irán resolviendo de manera asíncrona. Esto es beneficioso ya que en otro tipo de

lenguajes una tarea lenta o que requiera mucho procesamiento colgaría el servidor, con Node.js esto no sucede.

- Las peticiones que se hacen tanto de HTTP como las de entrada y salida son inmensamente rápidas, inclusive con una gran cantidad de peticiones y con un hardware simple, lo que se refleja en ahorro.
- Debido a que puede utilizarse tanto en backend como en frontend, se puede obtener una mayor productividad y más facilidad de mantenimiento, incluso para un desarrollador Jr.
- JavaScript es tan popular que la reutilización o el reciclaje de código facilita y agiliza en gran medida la línea de producción, sobre todo si la línea de desarrollo se centra en productos similares.
- Node.js es ligero, rápido, dispone de librerías muy maduras como WebSockets que además son muy fáciles de usar y no requieren d configuraciones tan exhaustivas.
- Finalmente, quizá el punto más importante es que el gestor de paquetes de Node.js dispone de más de 700mil librerías, convirtiéndose en el más grande respecto a otros lenguajes (Bliss, 2018).

Python

Python es un lenguaje de programación que se ha popularizado debido a la simpleza en su código y a la fácil legibilidad del mismo, es multiparadigma ya que en cierta parte soporta parcialmente la orientación a objetos, programación imperativa y programación funcional. Es fácil de interpretar, dinámico y multiplataforma. Entre las características más importantes se encuentran:

- Es multiparadigma, sin obligar a los desarrolladores a adoptar un estilo de programación en específico ya que permite varios estilos.
- Usa tipado dinámico y un conteo de referencias propio para poder administrar mejor la memoria
- Es fácil escribir nuevos módulos y extensiones en C o C++.
- Una de las características más importantes es su enlace dinámico de métodos durante la ejecución del programa (Wikipedia, 2021).

Una de las grandes ventajas de Python es que ofrece una amplia cantidad de librerías y frameworks desarrollados bajo su sintaxis, lo que facilita el desarrollo de aplicaciones de toda índole, entre los más populares para el diseño e implementación de aplicaciones Web, se encuentra Flask, un framework minimalista que ofrece una ágil y eficiente metodología de desarrollo, ya que permite levantar una aplicación con muy pocas líneas de código.

Tecnologías de Front-End

En el desarrollo de Software, el frontend es todo lo contrario al backend, es la parte que puede ver el usuario y con la que puede interactuar, es donde se encuentran los elementos gráficos y los elementos de la aplicación, como su nombre lo dice, es el frente, donde se pueden incluir estilos, colores, animaciones de todo el diseño final. Para poder lograr todo esto, es importante y necesario trabajar sobre el código, al igual que en el backend, con la diferencia de que se utilizan diferentes tipos de lenguaje, más cercanos a la interacción y comprensión de los usuarios (Nestrategia, 2021). Para convertirse en desarrollador de frontend, es importante conocer y dominar algunas tecnologías que son muy útiles para su desarrollo y que se define a continuación:

HTML

HTML es un lenguaje de marcado que gracias a sus etiquetas permite definir la estructura y contenido que se desea en una aplicación web. Todo está definido por etiquetas, por ejemplo, si se desea un párrafo se usa la *p,* si se desea una imagen se usa *img,* y así para cada requerimiento específico. Cada etiqueta debe tener una estructura que debe estar construida de la siguiente manera:

- Apertura de la etiqueta.
- Contenido de la etiqueta.
- Cierre de la etiqueta,

Para que una etiqueta funcione correctamente, necesita de atributos que definan su comportamiento, también existen etiquetas invisibles (*header, footer, etc*) y meta etiquetas que permiten que nuestra página o aplicación web sea entendida por robots como Google. Finalmente, lo más importante al momento de escribir HTML es la semántica, ya que cada elemento debe tener el orden adecuado y la etiqueta correcta para que la aplicación funcione correctamente (Castro, 2021).

CSS

CSS permite personalizar una página o una aplicación web, dando estilo a las etiquetas HTML previamente estructuradas. Es importante para poder acomodar posición, color, animación, y demás propiedades de diseño. Para usar CSS en las etiquetas HTML, es necesario seguir algunos pasos:

- Escribir la etiqueta a modificar junto a una llave ({) de apertura.
- Escribir el nombre de la propiedad que se quiere modificar y seguido dos puntos (:).

- Escribir el nuevo valor que se quiere dar a la propiedad, por ejemplo, si es un color, puede ser *green* y seguido un punto y coma (;).
- Finalmente una llave de cierre (}).

Para ser un desarrollador de frontend completo es muy importantes saber utilizar CSS, ya que ayuda a seguir buenas prácticas, elaborar un código propio y ahorrar tiempo y dinero. (Castro, 2021)

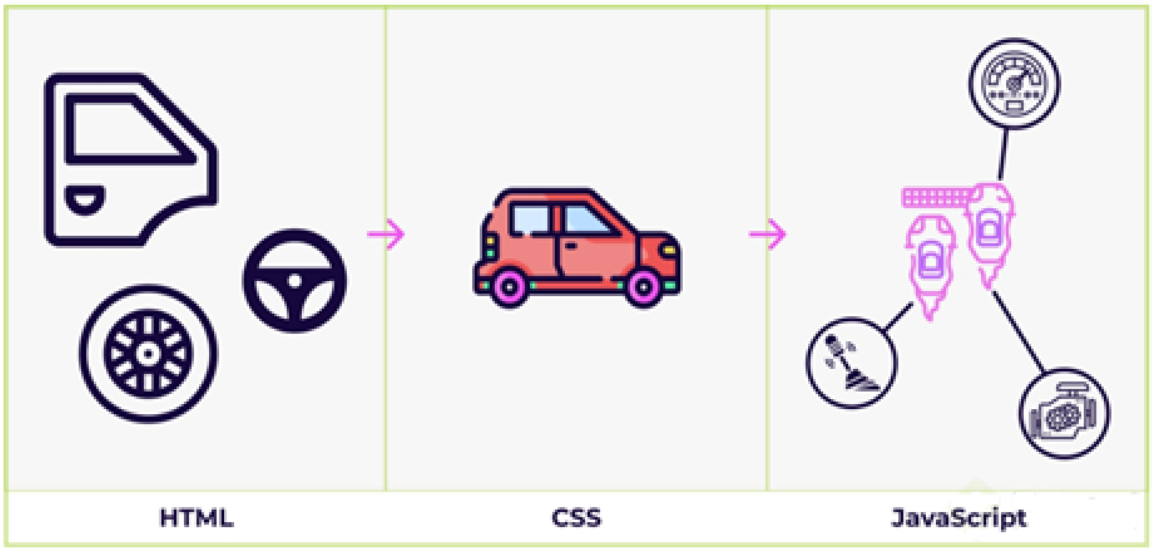
JavaScript

JavaScript es un lenguaje de programación interpretado, se define como multiparadigma ya que es orientado a objetos, imperativo y dinámico. Tiene dos funciones principales, la primera es que sirve para interactuar con los usuarios, ya que gracias a JavaScript se puede escuchar y entender las acciones que realizan los usuarios en la interfaz gráfica. JavaScript es el paso final para estructurar un frontend adecuado, HTML sirve para estructurar, CSS para dar forma y JS se programan los enlaces necesarios para que el producto final pueda obedecer las órdenes enviadas a través de la interfaz de usuario (Castro, 2021). Esto puede entenderse mejor con el ejemplo presentado en la figura 17.

La segunda función de JavaScript es que sirve para comunicarse con el Backend, sirve como un puente para los usuarios que entregan los datos a través de una interfaz gráfica y JavaScript los envía hacia el backend para poder responder adecuadamente a sus requerimientos. Para esto se utiliza un API, que es donde el frontend hace peticiones de envió de información y para recibir información (Castro, 2021).

Figura 17

Procesos de CSS, HTML y JavaScript.



Nota: La imagen fue tomada de (Castro, 2021).

Además del uso y del imperativo conocimiento de las tecnologías anteriores para poder desarrollar el frontend, también es importante tener conocimiento de frameworks, librerías y preprocesadores que expanden la capacidad de las interfaces de usuario que se pretende desarrollar, entre algunos de estos se encuentran:

Frameworks, Librerías y Preprocesadores

Al momento de desarrollar software es importante contar con ciertas tecnologías que ayuden a agilizar el proceso y además brinden prestaciones que en la actualidad son esenciales para el correcto funcionamiento de una aplicación, un desarrollador siempre debe estar a la vanguardia y buscar las mejores opciones para brindar un servicio.

Descubrimiento y publicación de servicios

Este es un principio fundamental en las arquitecturas orientadas a servicios (SOA), consiste en que tanto nuestros servicios como los servicios ya implementados por terceras personas, estén disponibles, publicados, accesibles y etiquetados correctamente con metadatos que permitan identificarlos para lanzar búsquedas minuciosas para identificar cuales servicios pueden ser reutilizados en tareas específicas. Sin embargo, para aplicar correctamente este principio se debe mencionar y tener muy claro un térmico clave: La Gobernanza. Una gobernanza fuerte está basada en la buena organización de los departamentos de TIC, con esto se garantiza una dirección adecuada de la estrategia para publicar estos servicios, con esto se logra optimizar la reutilización de los recursos y asegurar el crecimiento ordenado y el mantenimiento de la gama de los servicios publicados conforme a las necesidades de negocio que vayan apareciendo (Morales, 2021).

Entre las tecnologías que permiten la implementación de este principio se encuentra *La descripción, el descubrimiento y la integración universal (UDDI)* que consiste en registrar los servicios comerciales existentes en internet en un registro que está basado en XML. Estos registros pueden ser encontrados manual o automáticamente a través de APIs o de GUIs especializadas para encontrar dichos servicios web, también existen mecanismos de multidifusión como WS-Discovery que reduce la necesidad de registros centralizados (Alzaghoul, 2021).

Los contenedores de Oracle para J2EE o OC4J proveen de UDDI y en su guía se menciona los tópicos esenciales para que se la implemente de manera correcta, entre los que se encuentran:

- **Registro UDDI:** Consiste en tres tipos de búsqueda:
    - *Páginas en blanco:* Contiene información de contacto.

- *Páginas amarillas temáticas:* Contiene categorías industriales basadas en taxonomías estandarizadas.
- *Páginas verdes de servicios:* Contiene información técnica de los servicios web que son expuestos por su proveedor.

- **Archivo UDDI:** Consiste en cuatro tipos de estructura de datos la fácil localización y entendimiento de la información:
  - *businessEntity:* Es el nivel más alto de la estructura de datos que contiene información descriptiva acerca del negocio que publica el servicio.
  - *businessService:* El segundo nivel de la estructura de datos que contiene información descriptiva acerca del área técnica a la que está enfocada el servicio web.
  - *bindingTemplate:* El tercer nivel de la estructura de datos que contiene información técnica específica del servicio web y referencias de las especificaciones de su interfaz.
  - *tModel:* El cuarto y último nivel de la estructura de datos que brinda una descripción de las especificaciones técnicas del servicio web orientados a facilitar la búsqueda a los consumidores dentro de todos los servicios registrados.

Además de estas prestaciones, Oracle también dispone de apartados empresariales, que, si bien trabajan bajo los mismos enfoques, el manejo de los datos es mucho más amplio y se utilizar tecnologías diferentes como Oracle UDDI Enterprise Web Service Registry, que como se mencionó son para cargas empresariales (Oracle, 2012).

Balanceo de cargas

La distribución del tráfico que circula por una red es actualmente algo imperativo dentro de una estructura de software, esto permite que la información sea distribuida entre múltiples servidores y sistemas, el beneficio de un buen balanceo de carga es capaz de asegurar que se obtenga una mayor disponibilidad de los servicios que se están ofertando. Una aplicación o servicio web debe estar disponible el 100% del tiempo para los clientes, y es una de las características principales que se busca con esta tecnología. Para poder implementar un balance de carga adecuado se pueden usar ciertas herramientas que pueden agilizar este proceso, entre las que pueden estar:

- *Adaptador de Red:* Permite dividir las cargas entre diferentes tarjetas que pueden estar basadas en diferentes algoritmos, además cada IP es tratada de forma individual, haciendo que solo vaya en una ruta de conexión. El adaptador de red más pequeño obtiene todo el tráfico.
- *Software SafeKit:* Ofrece un clúster de servidores para que la aplicación sea mejor al momento de brindar soluciones de escalabilidad. Además, no requiere ningún de ningún otro servidor externo para garantizar el balance de carga.
- *NGINX y NGINX Plus:* Distribuye la carga de trabajo entre varios servidores de manera uniforme, ofrece las ventajas de escalabilidad y de redundancia, haciendo que en caso de que un servidor falle, siempre va haber otro disponible.
- *KEMPs Free LoadMaster:* Esta orientado a las pequeñas empresas y a desarrolladores que busquen una alternativa para el balanceo de cargas, ya que es libre, desarrollado por una empresa bien establecida y

finalmente brinda la seguridad de brindar un servicio de calidad a pesar de no tener costo a diferencia de muchos otros balanceadores actuales.

- *Snapt:* Es un balanceador de carga que también ofrece la prestación de acelerador web y además es de las mejores opciones de firewall de aplicaciones cloud, DevOps y también despliegues virtualizados (Mundowin, 2021).

Seguridad

La seguridad como tal es una disciplina encargada de proteger la información almacenada en un sistema, garantizando su integridad y privacidad. Existen varios tipos de seguridad y todas están relacionadas con la calidad de un sistema, ya que, si se trata de un software, hardware o sistema en general de baja calidad, entonces será indudablemente fácil de penetrar. Los enfoques principales de la seguridad son en primer lugar que el producto siga funcionando correctamente a pesar de cualquier ataque malicioso, y en segundo lugar que disponga de la resistencia adecuada y proactiva frente a cualquier ataque (ITTGWEB, 2021).

Existen diferentes tipos de seguridad informática que una empresa debe tener en cuenta para proteger su información y su prestigio, estos son los tres principales tipos de seguridad informática:

- *Seguridad de hardware:* Son los que proporcionan una seguridad más robusta en comparación a los otros tipos, ya que es un dispositivo físico como: cortafuegos, firewalls o módulos de seguridad de hardware (HSM). Adicionalmente la seguridad informática concierne también en cómo se debe proteger los equipos de cualquier daño.

- *Seguridad de Software:* Es utilizada para proteger la aplicación o el software en sí de posibles ataques maliciosos, hackeos y otros riesgos, a tal punto que el programa siga operando correctamente a pesar de los riesgos potenciales. El enfoque de este tipo de seguridad informática es aprovechar las mejores prácticas de la ingeniería de software y tener presente la seguridad desde el primer momento que s empieza a desarrollar el software. Así como hay equipos de hardware especializados, también hay programas especializados como: antivirus, cortafuegos, antispam, y otros más de control de contenido específico.
- *Seguridad de red:* Está enfocada en proteger los datos que circulan por una red. Evita que se propaguen amenazas como: virus, spyware o malware por la red. Al igual que en la seguridad de software, hay softwares especializados para la seguridad de red, como: antispyware, sistemas de prevención de intrusiones (IPS) y redes privadas virtuales (VPN) (Universidad Internacional de Valencia, 2018).

Tolerancia a fallos

A lo largo del presente trabajo se ha mencionado en reiteradas ocasiones a la tolerancia a fallos, debido a que es un eje principal de la investigación, además también se ha mencionado que es algo intrínseco dentro de las arquitecturas de microservicios, ya que al ser un sistema distribuido el control y la gestión de fallos es algo que puede realizarse de una manera mucho más sencilla. La tolerancia a fallos es una tecnología diseñada específicamente para que un sistema siga funcionando de manera continua inclusive si se da el caso de alguna entrada incorrecta o algún fallo (Wikipedia, 2021). Las características principales de la tolerancia a fallos son:

- *No deben existir puntos de falla***:** Si el sistema sufre algún fallo, debe seguir operando sin interrupciones durante el proceso de reparación.
- *Aislamiento de componentes que están fallando:* El sistema debe poder aislar el fallo hacia una locación específica, para que no existan reincidencias, por lo que es importante incorporar mecanismos de detección de fracaso y que puedan clasificar de que tipo son los mismos.
- *Contener fallos:* Es importante tener mecanismos de apoyo que eviten que se provoque un fallo en cascada.

La tolerancia a fallos es tratada mediante tres vías principales, las cuales son esenciales para su correcta implementación:

- *Replicación:* Funciona para proporcionar múltiples instancias idénticas del sistema a donde se dirigen las solicitudes en paralelo, para así poder elegir el resultado adecuado sobra la base de un quorum.
- *Redundancia:* En el sistema existen más de una instancia idéntica del sistema y en el caso de que una falle, poder cambiar a una diferente para que el proceso no se vea afectado.
- *Diversidad:* Proporciona más de una instancia distinta de una misma especificación, y poder utilizarla como duplicado para hacer frente al error de una aplicación en concreto. (Wikipedia, 2021)

A pesar del costo que puede representar el adquirir sistemas y hardware específicos tolerantes a fallos, la prevención de un proceso puede evitar la pérdida de mucho dinero, además hay alternativas de software que hoy en día son una de las opciones más rentables al momento de implementar tolerancia a fallos y que brindan todas las cualidades anteriormente mencionadas, entre estos pueden encontrarse a

Resilience4J de Java, Sentinel de Alibaba y Hystrix de Netflix, compañías que deben garantizar una operación de sus sistemas del 100% del tiempo.

Herramientas de Software

SpringBoot

SpringBoot es una infraestructura que fue diseñada con el objetivo de facilitar el desarrollo backend de aplicaciones basadas en Spring, esto debido a que configurar Spring es muy complicado ya que consta de archivos XML que deben ser cuidadosamente configurados inclusive para aplicaciones pequeñas, como decir “¡Hola Mundo!”. El objetivo de SpringBoot es proporcionar un kit de herramientas que permitan desarrollar una aplicación y “simplemente ejecutarla”, ya que la mayoría de su configuración viene por defecto y es muy sencilla de manejar porque son solo anotaciones y no archivos XML completos (Perry, 2017). Las ventajas de usar SpringBoot son las siguientes:

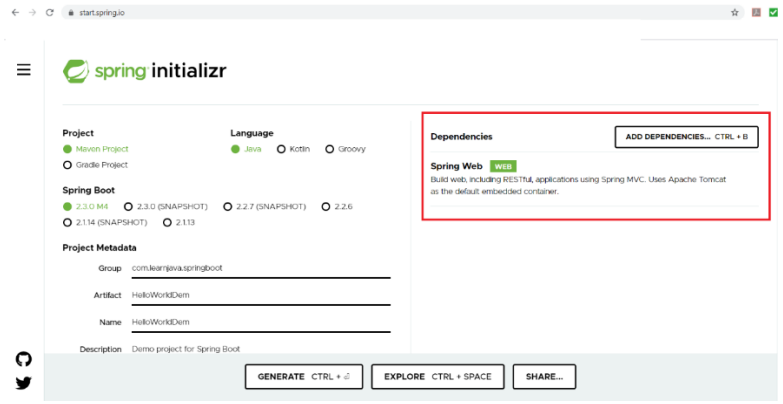
- Es intuitivo, esto gracias a que sus valores predeterminados son razonables y permite desarrollar una aplicación sin profundizar solo en configuraciones. Por ejemplo, SpringBoot tiene contenedores como Tomcat, Jetty o Undertow incorporados y no hace falta ejecutar archivos WAR (SpringBoot, 2021; Perry, 2017).
- Es personalizable, esto gracias a que permite cambiar todos los valores predeterminados si así se desea para poder cumplir con los requerimientos propuestos, además esto es posible tanto en la configuración inicial, como también en el ciclo de desarrollo (Perry, 2017). De igual manera, toda la configuración de las librerías de terceros es posible de revisar, testear y manipular (SpringBoot, 2021).

- Finalmente inicializarlo y empezar utilizarlo son de las grandes ventajas que ofrece SpringBoot, además cada iniciador puede ser con respecto al tipo de aplicación que se va a desarrollar. La configuración manual es limitada y se agiliza gracias a estos inicializadores, además que tienen soporte para diversos sistemas operativos como Linux, Mac y Windows (Perry, 2017).

Existen varias maneras para generar una aplicación basada en Spring, la primera se puede apreciar en la figura 18, es la página web oficial de Spring Boot, como se puede ver esta permite crear un proyecto una vez se le haya proporcionado los datos básicos del mismo, entre las opciones que se tiene, es que puede ser basado en Maven o Gradle, se puede seleccionar el lenguaje entre Java, Kotlin o Grovy, se puede escoger la versión de Spring Boot y finalmente se debe seleccionar las dependencias que por lo general es Web. Finalmente se escribe un nombre y adicionalmente se tiene la opción de desplegar el proyecto con todos los módulos disponibles de Spring, en caso de no conocer el nombre del que se quiere utilizar.

Figura 18

Spring Initializr

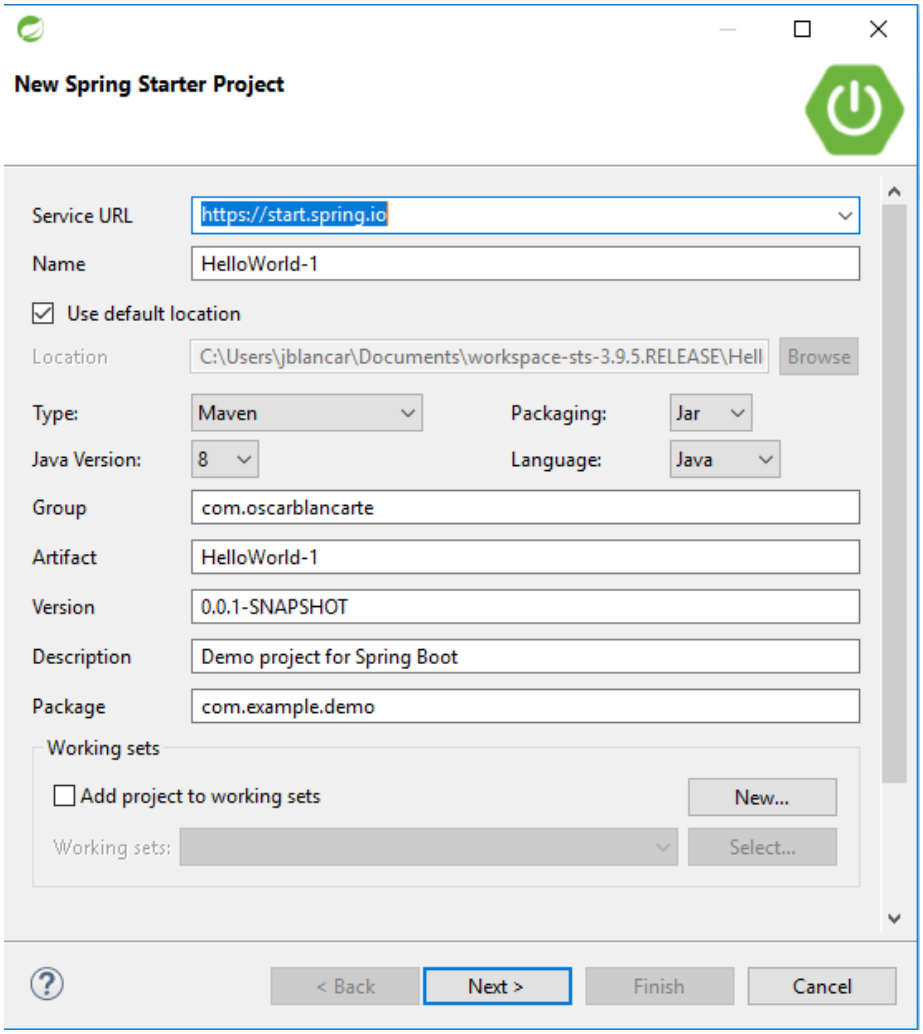


Nota: Figura tomada de (Blancarte, Oscar Blancarte Software Architect, 2018).

Para la segunda opción y que por lo general es la más utilizada, se tiene Spring Suite que es la que se aprecia en la figura 19, este es un IDE basado en Eclipse y su utilización es muy intuitiva por lo que no genera mucha dificultad. En la primera ventana pide los datos básicos del proyecto, y en las siguientes ventajas se dispone de las mismas opciones de la página web, el tipo de proyecto (Maven, Gradle), el lenguaje (Java, Kotlin, Grovy) y la dependencia (Web por defecto).

Figura 19

Spring Boot IDE



Nota: Figura tomada de (Blancarte, Oscar Blancarte Software Architect, 2018).

Después de generar un proyecto con cualquiera de las dos alternativas, se genera un prototipo de aplicación que es la clase MAIN para correr una aplicación más grande y se verá similar al código de la figura 20.

Figura 20

Main del proyecto basado en Spring.

```
1  package com.example.demo;
2
3  import org.springframework.boot.SpringApplication;
4  import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6  @SpringBootApplication
7  public class HelloWorldApplication {
8      public static void main(String[] args) {
9          SpringApplication.run(HelloWorldApplication.class, args);
10     }
11 }
```

Nota: Figura tomada de (Blancarte, Oscar Blancarte Software Architect, 2018).

Antes de ejecutar la aplicación, es necesario crear un servicio REST, que es el que se va a probar y visualizar en el navegador, esto se lo desarrolla en una nueva clase, pero en el mismo paquete del main, un servicio REST se verá similar a la figura 21.

Figura 21

Clase REST para el proyecto basado en Spring.

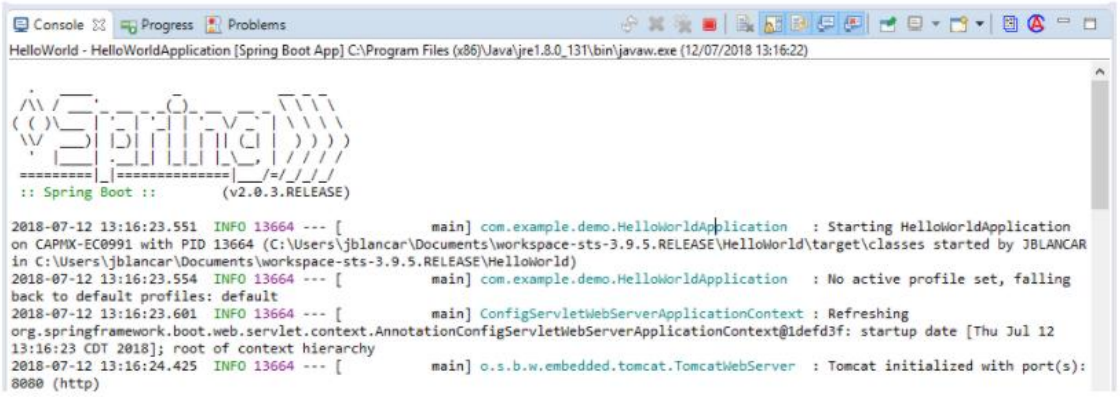
```java
package com.example.demo;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HelloRest {
    @RequestMapping("/hello")
    public String helloWorld() {
        return "Hello World";
    }
}
```

Nota: Figura tomada de (Blancarte, Oscar Blancarte Software Architect, 2018).

Finalmente, en un Tomcat embebido se despliega el proyecto viéndose en la consola algo similar como en la figura 22 y en un navegador en el localhost se puede empezar a probar y visualizar el desarrollo del proyecto como en la figura 23.

Figura 22

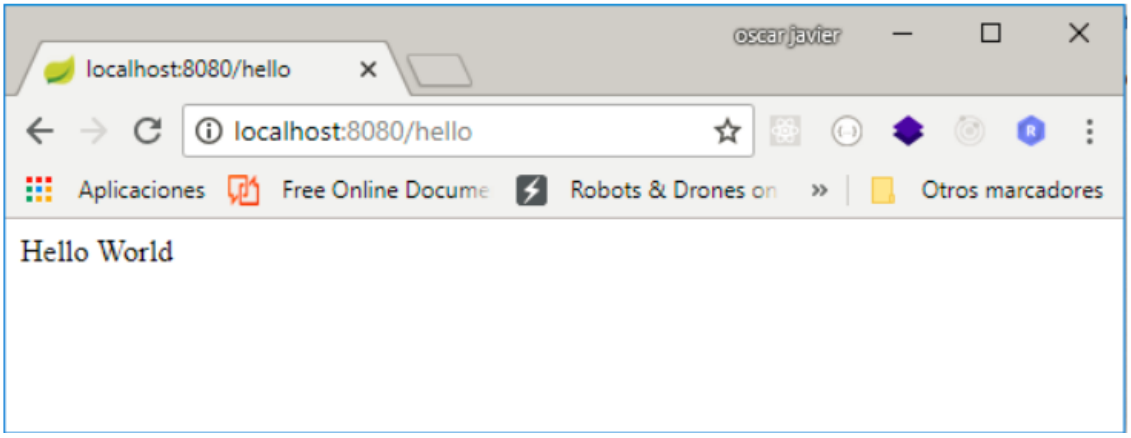
Consola en un Tomcat embebido.



Nota: Figura tomada de (Blancarte, Oscar Blancarte Software Architect, 2018).

Figura 23

Ejemplo de aplicación en el localhost



Nota: Figura tomada de **(Blancarte, Oscar Blancarte Software Architect, 2018)**.

Hystrix

Como se mencionó en los antecedentes de la presente investigación, *Hystrix* es una librería open source desarrollada y proporcionada por Netflix, está diseñada para aislar puntos de acceso a sistemas remotos, servicios y librerías de terceros gracia a la implementación del patrón *Circuit Breaker* que permite gestionar diversos aspectos tales como: timeouts, estadísticas de éxito y fallo, semáforos, lógica de gestión de error, lógica de latencia y de esto modo deteniendo fallos en cascada, mejorando la resiliencia en sistemas complejos distribuidos donde la probabilidad de fallo es inevitable (Rodríguez, 2015; Crespo, 2016).

Recientemente se comunicó que Hystrix ha ayudado al desarrollo de nuevas herramientas de software con el mismo propósito y ha pasado a modo de mantenimiento, sin embargo, su utilización sigue constante y presente en muchas aplicaciones en las que su utilización sea lo más viable, Hystrix tiene tres puntos principales que son los que la hacen una librería tan robusta y utilizada:

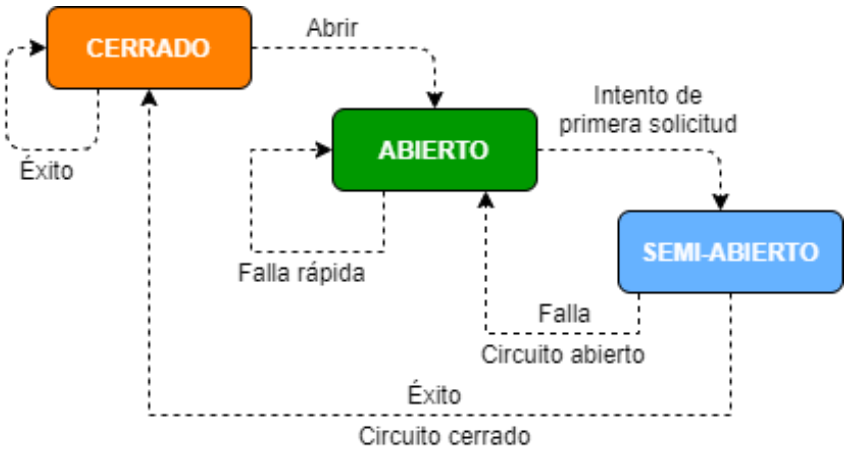
- *Tolerancia a fallos y latencia:* Detiene fallos en cascada, los retrocesos y los procesos degradados se tratan muy cuidadosamente, se garantiza una rápida recuperación ante el fallo y finalmente se separa los threads de las alertas mediante disyuntores.
- *Operación en tiempo real:* Toda su configuración, manejo y supervisión es en tiempo real, Todos los cambios en el servicio entran en vigencia inmediatamente mientras se expanden por todo el sistema. Todas las decisiones cambios y resultados se ven en segundos.
- *Concurrencia:* Toda la ejecución de Hystrix se la realiza en paralelo, se almacena todas las solicitudes en caché y se va procesando todo mediante lotes y colapso de solicitudes (Netflix, 2018).

Para comprender el funcionamiento de Hystrix y su patrón *Circuit Breaker* de una manera más práctica, se puede apreciar en la figura 24, lo que hace Hystrix es monitorear las fallas de un método específico y si las fallas empiezan a acumularse hasta un umbral definido, este abre el circuito para que las futuras solicitudes fallen automáticamente con motivo de prevención, de este modo se van almacenando en un método de reserva específico hasta que el sistema deje de reflejar errores en su proceso de funcionamiento y claramente mantiene el circuito abierto.

Dentro de una arquitectura de microservicios, la tolerancia a fallos y el patrón *Circuit Breaker* son muy importantes para gestionar los datos y la recepción y envió de los mismos, de este modo se logra la funcionalidad esperada de la arquitectura, además la facilidad de replicar una patrón como este, permite que la escalabilidad sea fluida y no impida, ni interrumpa el funcionamiento normal de todo el sistema.

Figura 24

Diagrama de Estados de Circuit Breaker



Sistemas de ingeniería, informáticos y de comunicación

Uno de los ejes que representa el presente trabajo es el manejo de sistemas de hardware remotamente gracias al desarrollo de una arquitectura de software que puede estar enlazada a la red gracias al internet o a protocolos de comunicación actuales. Por ello es importante definir las tecnologías que actualmente existen para saber diferenciar el enfoque que se le dio a la presente investigación.

Sistemas Ciber-físicos (CPS, Ciberphysical Systems)

Los sistemas ciber-físicos son sistemas de ingeniería que están construidos y dependen de la integración tanto de algoritmos computacionales como de componentes físicos. Los avances en CPS han permitido que exista mayor capacidad, escalabilidad, resiliencia, seguridad y usabilidad en muchos sistemas actuales que pueden llamarse críticos, debido a las prestaciones y a la disponibilidad que deben ofrecer los mismos. Ejemplos actuales de los CPS pueden ser: smart grid, sistemas de carros autónomos,

monitoreo médico, sistemas robóticos y la automatización de pilotos de aviación. Como bien se pudo ver, los CPS pueden ser encontrados en varias áreas de la investigación, pero hay que saber diferenciar que si bien son sistemas que integran la parte física con el software, no necesariamente están interconectadas gracias al internet, sino pueden ser redes internas con ciertos protocolos de comunicación (NSF, 2020).

Internet de las Cosas (IoT, Internet of Things)

IoT es un enfoque y un concepto que se refiere a una infraestructura de objetos, personas, sistemas e información, interconectados a través del internet y que tengan una interfaz con sistemas y servicios inteligentes que permitan procesar todos estos datos del entorno físico al entorno virtual y de ser necesario viceversa (ISO/IEC JTC 1, 2014). Los campos de aplicación de IoT al igual que los CPS, son amplios y se puede verlos actualmente en ámbitos empresariales, medios de comunicación, desarrollo y administración de infraestructura, agricultura, medicina, transporte, y demás, pero como se mencionó al principio, el concepto se refiere a una interconexión de objetos que se utilizan en la vida cotidiana a través del internet. (Cisco, 2019)

CPS vs IoT

Como se definió anteriormente, tanto CPS como IoT son perspectivas que se han empezado a implementar en el mundo para un mejoramiento de la informática y de la comunicación tanto de hardware como de software. Sin embargo, ambas vienen de dos ramas de la investigación diferentes, por lo que su énfasis también es distinto:

- CPS nace de la necesitad en los sistemas de control, la informática, servicios y sistemas en tiempo real y las grandes redes de sensores.

CPS enfatiza mucho más los sistemas híbridos y la verificación formal de sistemas dinámicos.

- IoT nace en las necesidades de las redes de comunicación actuales y sobre todo en el creciente desarrollo de las comunicaciones inalámbricas. IoT enfatiza y prioriza mucho más los protocolos de comunicación.

Si bien las diferencias en la práctica entre CPS y IoT no llegan a ser tan evidentes, ya que en ambos paradigmas se consideran temas de privacidad, seguridad y muchos otros, la mayor diferencia podría ser que los dispositivos utilizados en IoT son CPS, pero los CPS no siempre están conectados a internet y no es un requerimiento, por lo que los dispositivos CPS no son por regla general IoT. En el futuro debería haber mucha más interacción entre estas dos tecnologías y en sus comunidades de comunicación, ya que el mundo quizá esté regido por el trabajo conjunto de ambas (Taha, 2016).

Servidor virtual privado (VPS).

Un servidor virtual privado (Virtual Private Server, VPS), es un servicio en la nube utilizado para el almacenamiento tanto de servicios web, como de proyectos que requieran un servidor dedicado y altos tiempos de disponibilidad. Dentro de las características principales de un VPS se encuentran las siguientes:

- Siempre se mantiene en línea y brinda recursos garantizados para cada proyecto.
- Se acopla a las necesidades del usuario, ya qué todos los recursos que ofrece son personalizables y solo se paga por lo que se va a ocupar.

- Es muy útil al momento de desplegar proyectos en el internet sin necesidad de muchas configuraciones.

Si bien existen alternativas gratuitas para el despliegue de un proyecto en internet, estos por lo general tienen muchas limitantes en cuanto a los recursos que brindan, por lo que la mejor opción sería usar un VPS, ya que cumplirá con todos los requerimientos sin mayor problema. (Web Empresa., 2015)

Robótica móvil

La robótica móvil tiene como propósito el desarrollo de robots o máquinas automáticas que se adapten a cualquier entorno y que sean capaces de trasladarse en el terreno específico para el cual estén diseñados, además se diferencian de los robots industriales que solo tienen algunas de sus piezas articulables y están fijos, unidos a una superficie (Moubarak, 2018). Los robots pueden constar de múltiples sensores y actuadores que benefician a su rendimiento y les ayudan a cumplir su objetivo.

Sensores

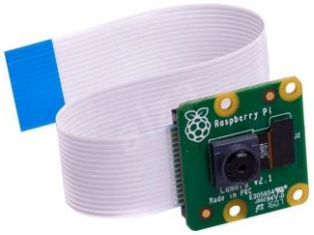
Los sensores que también son conocidos como transductores, son componentes utilizados para adquirir datos del medio o el entorno en el que se encuentran. Los sensores convierten un fenómeno físico en voltaje, que es fácilmente medible y legible. (Dewesoft, 2020). Existen diversos sensores que son utilizados en la robótica móvil y cumplen un papel importante para poder obtener un objetivo determinado.

Cámaras. Las cámaras son sensores que cumplen un papel similar al de los ojos humanos, se encargan de captar los rayos de luz del exterior y transformarlos en imágenes, existen algunos módulos y cámaras orientados a la robótica móvil.

Raspberry PI Camera Module. Es un producto original de la fundación Raspberry PI, existen varios modelos que han sido lanzados a lo largo de los años. El módulo más comercial actualmente es el V2, el cual puede observarse en la figura 25 y se detallan sus características más importantes en la tabla 8:

Figura 25

Raspberry PI Camera Module V2



Nota: Figura tomada de (Raspberry PI, 2017).

Tabla 8

Características de la Raspberry PI Camera Module V2.

RPI Camera Module V2	
Tamaño	Aprox. 25mm x 24mm x 9mm.
Peso	3g.
Resolución	8 megapíxeles.
Modos de video	1080p30, 720p60 y 640 × 480p60/90

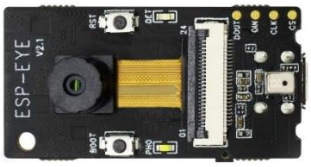
Sensor	Sony IMX219
Resolución del Sensor	3280 × 2464 pixeles.
Costo	Entre 25 y 30$

Nota: Tabla inspirada en (Raspberry PI, 2017).

ESP-EYE. Es un módulo basado en la tecnología ESP32 desarrollada por Adafruit, incorpora prestaciones como un micrófono, una cámara, memoria PSRam de 8MB y memoria flash de 4MB. En la figura 26 se puede observarlo y en la tabla 9 se detallan algunas de sus características más importantes:

Figura 26

Módulo ESP-EYE



Nota: Figura tomada de (Adafruit, 2019).

Tabla 9

Características del Módulo ESP-EYE.

ESP-EYE	
Tamaño	Aprox. 41 x 21mm x 2mm.
Peso	6g.
Resolución cámara	2 megapíxeles.
Chip	WiSoC ESP32 dual core Tensilica LX6

Sensor	OV2640
Memoria	4 MB FLASH/ 8MB RAM
Costo	Entre 25 y 30$

Nota: Tabla inspirada en (Adafruit, 2019)

ESP32-CAM. Es un módulo desarrollado por múltiples compañías y se basan en la tecnología del ESP32, tiene prestaciones como una cámara integrada de tamaño reducid, una memoria PSRAM de 4MB y una memoria flash de 4MB. En la figura 27 se puede observar el módulo y en la tabla 10 se detallan algunas de sus características más importantes.

Figura 27

Módulo ESP32-CAM

Nota: Figura tomada de (Seeed, 2017).

Tabla 10

Características del Módulo ESP32-CAM.

ESP32-CAM	
Tamaño	Aprox. 41 x 27mm x 5mm.
Peso	20g.
Resolución cámara	2 megapíxeles.

Chip	802.11b/g/n Wi-Fi BT SoC Module
Sensor	OV2640 – OV7670
Memoria	4 MB FLASH/ 4MB RAM
Costo	Entre 10 y 20$

Nota: Tabla inspirada en (Seeed, 2017)

Sensores de Luminosidad. Los sensores de luz, también llamados sensores fotoeléctricos, son dispositivos electrónicos que perciben la cantidad de luz o de luminosidad que hay en el ambiente y son capaces de variar o de responder a la intensidad con la que la luz es percibida.

LDR (Light Depending Resistor). Si bien no es directamente un sensor como tal, cumple una función muy similar, ya que su valor varía dependiendo de la cantidad de luminosidad que hay en el entorno en ese momento. Es inversamente proporcional, lo que quiere decir que mientras más luz perciba, menos será la resistencia que ejerce y viceversa, mientras menos luz perciba, la resistencia será mayor (330Ohms, 2017). Para el cálculo del voltaje de salida, por lo general se utiliza un circuito divisor de voltaje con el LDR como una de sus resistencias. En la figura 28 puede apreciarse un LDR y en la tabla 11 se definen algunas de sus características.

Figura 28

LDR (Light Depending Resistor)

Nota: Figura tomada de (330Ohms, 2017).

Tabla 11

Características del LDR.

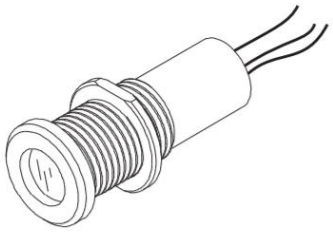
LDR	
Voltaje AC o DC Pico	100 V.
Corriente	5mA.
Disipación de potencia	50mW.
Rango de temperatura	-25ºC a 75ºC.

Nota: Tabla inspirada en (RS Components, 1997).

AD4070. Es un dispositivo fototransistor que ofrece en su salida una tensión variable dependiendo de la proporción de luz que este detecta en el entorno, está diseñado para funcionar y acoplarse adecuadamente a sistemas y mecanismos más complejos sin afectar su funcionamiento ni estética (SL Componentes, 2017). En la figura 29 se puede apreciar el sensor y en la tabla 12 se resumen sus características más importantes.

Figura 29

Sensor de Luminosidad AD4070



Nota: Figura tomada de (SL Componentes, 2017).

Tabla 12

Características del sensor de luminosidad AD4070

AD4070	
Voltaje de alimentación	De 10 a 19 V.
Corriente consumida	**<**1mA.
Voltaje de salida	Analógico de 0 a 4.8V
Rango de temperatura	0ºC a 50ºC.

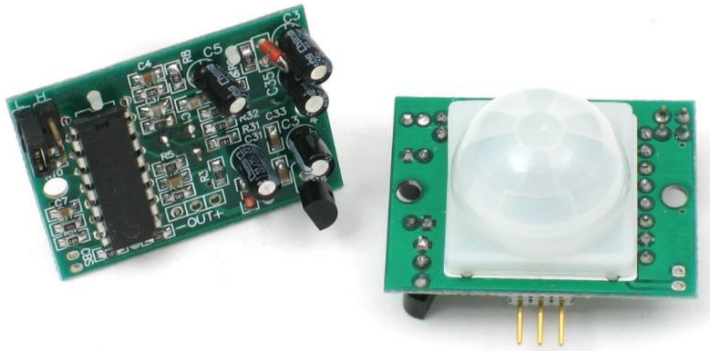
Nota: Tabla inspirada en (SL Componentes, 2017).

Sensores de Movimiento. Un sensor de movimiento o también conocido como sensor de presencia es un dispositivo pensado para optimizar el consumo energético, ya que lo que hace es activar o desactivar un sistema dependiendo de si detecta movimiento en el entorno en el que esté instalado (Soler&Palau, 2018).

PIR Motion. Es un sensor de movimiento desarrollado por Adafruit, que permite captar el movimiento en un rango determinado. Son pequeños, baratos, consumen muy poca energía y son fáciles de acoplar, utilizar e instalar. Su nombre viene de Pyroelectric ("Passive") InfraRed Sensors, lo que significa que utilizan infrarrojos para su funcionamiento (Adafruit, 2019). En la figura 30 se puede apreciar el modelo del sensor y en la tabla 13 se resumen sus características principales.

Figura 30

PIR Motion Sensor



Nota: Figura tomada de (Adafruit, 2019)

Tabla 13

Características del PIR Motion Sensor.

PIR Motion Sensor	
Voltaje de alimentación	De 5 a 12 V.
Corriente consumida	**<**60uA.
Voltaje de salida	Voltaje digital de 3V.
Rango de detección	De 3 a 5 metros.

Nota: Tabla inspirada en (Adafruit, 2019).

Sensores de Contacto. Son sensores que necesitan del contacto físico para saber y poder determinar qué tan lejos o cerca se encuentran de algún objeto, esto es un poco limitante ya que en algunas aplicaciones hasta que el sensor envié su respuesta, ya es tarde para tomar una medida preventiva. Aunque si son muy comunes en aplicaciones donde se necesite detectar objetos a muy corta distancia (Colomer, 2018).

Final de carrera. Son el sensor más típico de los utilizados para determinar proximidad por contacto, pueden ser mecánicos o eléctricos. En los eléctricos se tiene dos partes principales, una parte mecánica que es la cual se desplaza y mueve la segunda parte que es una lámina metálica que al contacto cierra el circuito permitiendo el flujo de corriente (Colomer, 2018). Su funcionamiento es similar al de un interruptor solo que no se quedan estáticos, sino que reaccionan al contacto, en la figura 31 se puede apreciar un modelo del sensor y en la tabla 14 se definen algunas de sus características.

Figura 31

Sensor final de carrera

Nota: Figura tomada de (Colomer, 2018).

Tabla 14

Características del sensor final de carrera

Sensor final de carrera	
Voltaje de alimentación	De 5 a 12 V.
Tiempo de respuesta	**<**10ms.
Voltaje de salida	De 5 a 12 V.
Rango de detección	Depende del accionador.

Nota: Tabla inspirada en (Colomer, 2018).

Acelerómetro. Se denomina acelerómetro a cualquier dispositivo capaz de medir la aceleración, no necesariamente de coordenadas, sino asociada al peso y a las pruebas que pueden hacerse a un dispositivo en un marco de referencia (Ortega, 2006).

Módulo MPU6050. Este es un módulo de medición inercial o IMU (Inercial Measurement Unit), consta de un giroscopio de tres ejes con el que se mide la velocidad angular y un acelerómetro de otros 3 ejes que sirven para medir los componentes X, Y, Z (Naylamp Mechatronics, 2018). En la figura 32 se puede apreciar el módulo y en la tabla 15 se definen las características principales.

Figura 32

Módulo MPU6050

Nota: Figura tomada de (Naylamp Mechatronics, 2018).

Tabla 15

Características del módulo MPU6050

MPU6050	
Voltaje de alimentación	De 2 a 5V.
Tolerancia aceleración máxima	10Kg.
Tasa de datos de salida	Programable
Salida digital	6 ejes, SDA-SCL

Nota: Tabla inspirada en (Naylamp Mechatronics, 2018).

Sensores de proximidad. Un sensor de proximidad es un transductor que está diseñado para detectar objetos o alguna señal que se encuentre cerca del dispositivo. Los sensores de proximidad se distinguen unos de otros según la variable física que utilicen (Colomer, 2018).

Sensor ultrasónico HC SR04. Este tipo de sensor es muy utilizado en proyectos de robótica ya que es muy preciso, su costo es bastante bajo y es simple de utilizar. Funciona tanto para esquivar objetos como para simular mapas de habitaciones y también señalar a que distancia puede estar algún objeto (Proyectos con Arduino, 2019). Su principio de funcionamiento está basado en emitir y captar la emisión de ondas acústicas, es decir el emisor envía ondas que rebotan en el objetivo y regresar al receptor, de este modo cumple con su objetivo (Colomer, 2018). En la figura 33 se puede observar el sensor y en la tabla 16 constan algunas de sus características.

Figura 33

Sensor ultrasónico HC SR04

Nota: Figura tomada de (Proyectos con Arduino, 2019).

Tabla 16

Características del sensor ultrasónico HC SR04

HC SR 04	
Voltaje de alimentación	De 2 a 5 V.
Corriente consumida	<2mA.

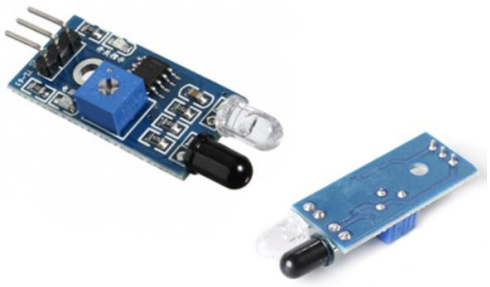
Voltaje de salida	Digital de 0 a 4.8V
Rango de medición	15º a 30º

Nota: Tabla inspirada en (Proyectos con Arduino, 2019)

Sensor Infrarrojo IR FC-51. Este es un tipo de sensor óptico que utiliza la luz infrarroja para su propósito. Los diodos LEDs son los que se utilizan más comúnmente para este tipo de sensores, ya que son pequeños, robustos y tienen una vida útil muy amplia. Al igual que en un sensor ultrasónico, consta de un emisor y un receptor. Su funcionamiento está basado en la emisión de luz infrarroja hacia el objetivo, la cual se refleja y el receptor puede captar esto, percibiendo dicho objeto (Colomer, 2018). El bajo costo y la fácil utilización de este tipo de sensores, lo ha hecho muy popular en innumerables proyectos y aplicaciones. En la figura 34 se puede apreciar el sensor infrarrojo y en la tabla 17 se definen algunas de sus características importantes.

Figura 34

Sensor Infrarrojo IR FC-51



Nota: Figura tomada de (Llamas, 2021).

Tabla 17

Características del sensor infrarrojo IR FC-51

Sensor Infrarrojo IR FC-51	
Voltaje de alimentación	De 2 a 5 V.
Corriente consumida	<43mA.
Voltaje de salida	Digital de 0 a 4.8V
Rango de medición	0º a 35º

Nota: Tabla inspirada en (Llamas, 2021).

Actuadores.

Los actuadores, a diferencia de los sensores que captan la energía externa y la transforman en energía eléctrica, transforman un tipo de energía en el inicio o activación de algún mecanismo o proceso automatizado, con la finalidad de cumplir algún objetivo.

Motores. En robótica móvil los actuadores más comunes son los motores, ya que el objetivo es que los robots sean móviles y los motores son los únicos que pueden generar la energía mecánica necesaria para este fin.

Motor DC. Un motor de corriente continua es el que se utiliza en robots móviles ya que su dependencia de energía es a partir de una batería que solo suple de corriente continua, este tipo de motores transforman la energía eléctrica en energía mecánica y provocan rotación gracias a un campo magnético. Existen motores DC estándar, paso a paso y sin escobillas, los que se tratan a continuación son los estándares, que se

aprecian en la figura 35 y en la tabla 18 constan algunas de sus características principales. Algo importante que mencionar es que, por lo general para el control y utilización de estos motores en robótica móvil, se debe hacer uso de algún driver o puente h que permita su cambio de dirección de rotación (Bricolabs Wiki , 2018).

Figura 35

Motor DC

Nota: Figura tomada de (Bricolabs Wiki , 2018).

Tabla 18

Características del Motor DC.

Motor DC	
Voltaje de alimentación	De 1.5 a 9 V.
Corriente consumida	<280mA.
Velocidad sin carga	12000±15%rpm
Corriente de arranque	<5A.

Nota: Tabla inspirada en (Bricolabs Wiki , 2018).

Plataformas de Hardware

Raspberry PI

Raspberry PI es una serie de computadores de placa reducida, sus creadores la catalogan como una" computadora del tamaño de una tarjeta de crédito", su costo es muy bajo y está diseñada para el desarrollo de software y la programación para cualquier público interesado, corre bajo un sistema operativo que puede ser AROS, Linux, UNIX o Windows, el cual se comprime dentro de una tarjeta SD que viene con la Raspberry PI. Su portabilidad y fácil implementación la han catapultado en gran medida y han popularizado su utilización en innumerables proyectos y aplicaciones, como la robótica móvil (RaspberriPi Foundation, 2015). Dispone de diversos modelos, pero el más popular es el 3 Model B, el cual se puede apreciar en la figura 36 y en la tabla 19 se resumen sus características principales.

Figura 36

Raspberry PI 3 Model B

Nota: Figura tomada de (RaspberriPi Foundation, 2015).

Tabla 19

Características de la Raspberry Pi 3 Model B

Raspberry Pi 3 Model B	
CPU	Quad Core 1.2GHz Broadcom BCM2837 de 64bits
Memoria	Capacidad de SD FLASH/ 1GB RAM
Puertos	• HDMI • 4 USB • 100 base Ethernet • CSI para cámara • DSI para display • Micro SD
Chips integrados	○ BCM43438 wireless LAN ○ Bluetooth Low Energy (BLE)
Costo	Entre 35 y 45$

Nota: Tabla inspirada en (RaspberriPi Foundation, 2015).

ESP32

ESP32 es un término utilizado para referirse a una familia de chips SoC (System on Chip), caracterizado por su bajo costo y su poco consumo de energía. Fue diseñado por Expressif Systems y su enfoque fue integrar tecnologías Wi-Fi y Bluetooth para comunicarse inalámbricamente con otros periféricos (Expressif Systems, 2016). En la figura 37 se puede ver el SoC ESP32 y en la tabla 20 se resume algunas de sus características principales.

Figura 37

SoC ESP32

Nota: Figura tomada de (Expressif Systems, 2016).

Tabla 20

Características del SoC ESP32

SoC ESP32	
CPU	Xtensa LX6 de doble núcleo (o de un solo núcleo) de 160 o 240 MHz y 600 DMIPS de 32-bit.
Memoria	520 KiB SRAM
Puertos	• 12-bit SAR ADC -18 canales • 4 SPI • Ethernet MAC con DMA dedicado • 2 interfaces I2S • 2 interfaces I2C • 3 UART
Chips integrados	○ Wi-Fi: 802.11 b/g/n ○ Bluetooth: v4.2 BR/EDR y BLE
Costo	Entre 35 y 45$

Nota: Tabla inspirada de (Expressif Systems, 2016).

Herramientas para realizar pruebas

En todo proyecto desarrollado con una metodología específica, se requiere realizar pruebas funcionales, de rendimiento y de usabilidad, esto con el fin de evaluar si se pudo cumplir o no con los objetivos planteados. Hay múltiples herramientas tanto de código abierto como de pago que brindan las prestaciones necesarias al usuario para que pueda valorar su proyecto.

Gatling

Gatling es una herramienta de código abierto diseñada para pruebas de funcionabilidad y rendimiento, se enfoca en proporcionar una solución óptima y sin ningún costo a los desarrolladores que necesitan evaluar las cargas que pueden soportar sus sistemas. Su funcionamiento se basa en scripts que se acoplan a cualquier tipo de escenario, además no requiere muchas configuraciones para obtener el resultado esperado, ya que se puede simular entornos de cientos a miles de usuarios y peticiones por segundo. Para poder utilizarlo, se pude hacerlo bajo herramientas de compilación como Maven, Gradle o SBT, y con cualquier IDE que permita abrir sus scripts, dependiendo de los requerimientos de cada usuario. Finalmente, la mayor ventaja que ofrece Gatling es que proporciona una amplia documentación para poder escribir scripts de pruebas propios y también para poder leer las cartillas en formato HTML que se generan al finalizar una prueba, donde se detallan cada una las peticiones, usuarios y demás variables de prueba asignadas. (Gatling, 2013).

Sistema de escalas de usabilidad

Un sistema de escalas de usabilidad (System Usability Scale, SUS), es una herramienta metodológica que está enfocada en medir algún objeto, proyecto,

dispositivo o aplicación. La mayor ventaja que tiene es que, a pesar de ser extremadamente simple de utilizar, con el tiempo y las investigaciones ha demostrado que los resultados que proporciona son extremadamente confiables y acertados.

Este sistema consta de las siguientes 10 preguntas con un rango puntaje del 1 al 5, siendo 1 totalmente en desacuerdo y 5 totalmente de acuerdo, estas preguntas deben ser adaptadas dependiendo de los requerimientos del evaluador remplazando el espacio en blanco con la palabra objeto, aplicación, sistema, dispositivo, etc.

- Creo que usaría este […].
- Encuentro este […] innecesariamente complejo.
- Creo que el […] fue fácil de usar.
- Creo que necesitaría de una persona con conocimientos técnicos para poder usar este […].
- Las funciones de este […] están bien integradas.
- Creo que el […] es muy inconsistente.
- Imagino que la mayoría de la gente aprendería a usar este […] en forma muy rápida.
- Encuentro que el […] es muy difícil de usar.
- Me siento confiado al usar este […].
- Necesité aprender muchas cosas antes de ser capaz de usar este […].

Para la medición final se debe tener en cuenta que no se trata de un porcentaje, sino de un puntaje sobre 100 puntos, para calcular el puntaje final, a las preguntas impares se les debe restar 1 al puntaje y para las preguntas pares, al valor de 5 se le resta el puntaje de la pregunta, finalmente se suma todos los valores. (UXpañol, 2017).

Capítulo III

Diseño de la Arquitectura

Requisitos de diseño

Como ya se mencionó en el capítulo 1 y como se pudo visualizar en la figura 1, el sistema implementado constará de tres partes principales, el Front-End, el Back-End y un robot que demuestra la funcionalidad práctica del presente trabajo, pero para un diseño adecuado es importante conocer y tener en cuenta los requisitos esenciales que necesita la arquitectura para poder cumplir con los objetivos planteados. Tras la investigación que se realizó en el capítulo 2, se pudieron ir conociendo las herramientas, tecnologías y sobre todo las estructuras de diseño más utilizadas actualmente en el desarrollo de sistemas similares al propuesto, por lo que se puede desglosar los requisitos funcionales y no funcionales del proyecto, los cuales se presentan en la tabla 21 y la tabla 22 respectivamente.

Tabla 21

Requisitos de diseño funcionales.

Requisito	Descripción
Requerimientos orientados al usuario. (Front-End).	• Debido a que la aplicación de control está orientada tanto a computador, como a dispositivos móviles, se debe optar por la mejor opción que ofrezca la solución más viable. • Los controles deben ser entendibles y deben ser fáciles de manejar. • La información extra que ofrezca la aplicación debe ser legible y debe brindar la confiabilidad que se busca,

Requisito	Descripción
Requerimientos que debe cumplir la arquitectura de software. (Back-End).	• Al ser una arquitectura de microservicios, lo que se busca es una independencia total tanto en funcionamiento como en el manejo de datos, de un microservicio con otro. • Debe ser escalable y versátil para cualquier aplicabilidad que se le quiera dar. • Debe brindar la opción de poder acceder a su información sin ninguna restricción de ubicación o dispositivo. • La información debe ser almacenada instantáneamente en las bases de datos de cada microservicio que se este utilizando, además debe permitir guardar cualquier cantidad de datos. • La arquitectura debe tener un Gateway que permita la comunicación con cualquier microservicio. • Se debe contar con un microservicio orquestador que funcionara como proxy para el resto de la arquitectura. • La arquitectura debe contar con un servidor de nombre, capaz de registrar y localizar los servicios ya existentes.
Funcionalidades que debe cumplir la arquitectura de software. (Back-End).	• La arquitectura de software debe poder comunicarse tanto con la aplicación de control del Front-End, así como con el servidor externo albergado en el robot. • Debe ser capaz de gestionar errores en caso de que algún microservicio falle o, que el servidor externo del robot no se encuentre en línea.

Requisito	Descripción
	• Puede calcular el tiempo desde que se realiza la petición desde la aplicación de control en el Front-End y hasta que llega a su destino, además brinda la información del enrutamiento de los microservicios.
Requerimientos que debe cumplir el robot.	• Debe contar con un microprocesador que brinde la opción de albergar un servidor y además que permita la conectividad a internet. • Debe tener un servidor interno en el cual se pueda levantar una aplicación web. • Debe contar con los sensores y actuadores necesarios para la aplicabilidad práctica.
Funcionalidades que debe cumplir el robot.	• Internamente debe poder manejar y procesar la información que le brinden sus sensores. • Debe poder conectarse con la arquitectura de software den el back end. • Debe poder conectarse con la aplicación de control en el front-end y enviarle la trasmisión de video de su cámara interna.
Detalles técnicos y manejo de datos.	• Toda la arquitectura del sistema seguirá un diseño de tipo RESTful, tanto en el front-end, back-end y en el servidor del robot. • Todos los datos que serán manejados tanto para peticiones y para respuestas serán de tipo JSON, los que se utilizarán para todas las validaciones. • Las peticiones que se hacen para el envió de datos son realizadas

Requisito	Descripción
	con el protocolo de comunicación HTTP. • La arquitectura estará albergada en un servidor en la nube que ofrece las prestaciones necesarias para el funcionamiento total del sistema. • Todos los elementos de la arquitectura deben contar con una conexión a internet para su correcto funcionamiento.

Tabla 22

Requisitos de diseño no funcionales.

Requisito	Descripción
Cualidades que la aplicación de control puede tener. (Front-End).	• La interfaz de control deberá estar dividida de tal manera que sea fácil poder manejar el robot y también visualizar la transmisión de video sin ningún inconveniente. • El tiempo de respuesta de las peticiones se debe intentar que sea lo más rápido posible para una mejor QoS. • La aplicación solo contará con una ventana en la que estarán repartidos todos sus elementos de manera estética y uniforme. • Se debe tratar de diferenciar cuando se utiliza alguno de los controles, ya sea con alguna notificación o algún cambio estético de color. • La aplicación desplegara alertas que dependerán de la respuesta a su petición, que podrían ser de éxito, fallo o alguna advertencia, dichas alertas contaran con un

Requisito	Descripción
	ícono y además un texto que indique la operación realizada.
Cualidades que la arquitectura de software puede tener. (Back-End).	• Puede contar con un servicio de configuración extra, el que estará vinculado directamente a un repositorio git, ya sea en una computadora local o ya sea en un repositorio online como GitHub. • Las bases de datos puedes estar albergadas en una maquina local o en la nube, además de que la base de datos de cada microservicio puede estar en locaciones diferentes. • Cada microservicio puede estar albergado en cualquier locación, ya sea física o en la nube. • Los microservicios pueden funcionar independientemente y pueden comunicarse directamente o a través del Gateway con otros microservicios.
Cualidades que puede tener0 el robot.	• El robot puede ser fijo o móvil. • Puede contar con la cantidad de sensores y actuadores que se desee. • La fuente de alimentación del robot puede ser directamente de la red eléctrica o puede contar con baterías recargables. • Si existiese actuadores como servomotores, la libertad de movimiento estará limitada dependiendo de la aplicabilidad práctica necesaria para demostrar su funcionamiento. • La conexión a internet puede ser mediante LAN, WLAN, o por datos móviles. Esto dependerá de la prestación que ofrezca el microcontrolador.

Requisito	Descripción
Restricciones generales.	• El acceso a las bases de datos no será directamente desde la aplicación de control, si se quiere acceder a una base de datos específica se deberá hacerlo desde algún otro servicio, como Postman, o directamente desde algún administrador de bases de datos como WorkBench. • Al ser una aplicación web, se debe contar con un navegador web en el dispositivo que se la vaya a utilizar. • Sin internet, o sin una conexión en la misma red de todos los componentes del sistema, este queda en su totalidad obsoleto. • La latencia está definida directamente por el proveedor de internet y su QoS, o de la distancia de los componentes en el caso de una red local. • Si en el robot se utilizan baterías, la autonomía estaría determinada por la capacidad de la batería y por el uso que se le de a las mismas.

Estructura de diseño del sistema

Como partes fundamentales de este trabajo de titulación y como su título lo indica, tenemos al control telemático, los sistemas ciber físicos y una arquitectura orientada a servicios y tolerante a fallos. Después de haber definido los requisitos esenciales del diseño, se puede elaborar el modelo general que tendrá el sistema, empezando por el front-end con el envío de los datos, después del Back-end para el procesamiento de los datos y finalmente el robot con su servidor interno para la recepción y aplicabilidad practica dependiendo de la información recibida. En la figura

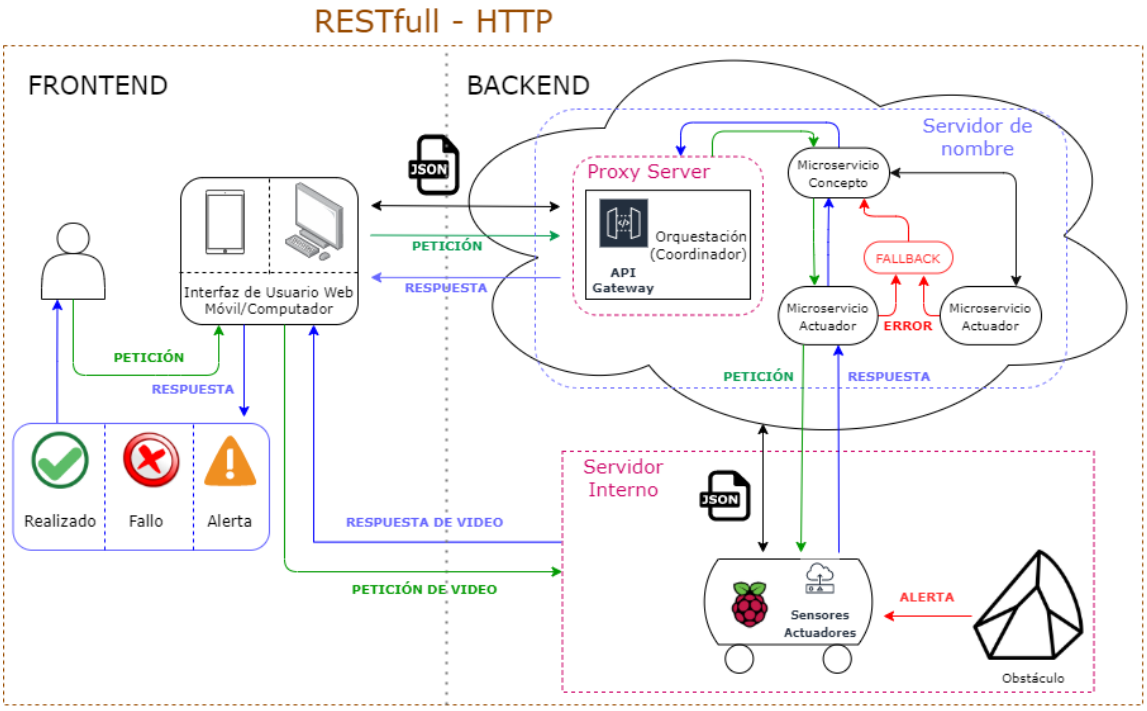
38 se puede observar la estructura del diseño del sistema una vez se han tomado en cuenta todas las consideraciones previas, así como el alcance práctico que se busca alcanzar. En cuanto a estructura general, en primer lugar, tenemos el Front-End que está conformado por la aplicación web que funcionará tanto en computador como en dispositivos móviles, será encargada de enviar las peticiones y recibirá una respuesta ya sea de éxito, fallo o alerta. En segundo lugar tenemos el Back-End, que es donde se encuentran los microservicios configurados independientemente, tolerante a fallos, con su propia base de datos, y registrados en un servidor de nombre, también se tiene un servicio especial que funcionará como Gateway y Proxy para la comunicación de toda la arquitectura, y finalmente se tiene el robot, con su servidor interno albergado en su microcontrolador que funcionara para la comunicación con el resto del sistema y además procesará los datos de los sensores y los actuadores, y devolverá una respuesta dependiendo a la acción realizada o una alerta en el caso de que haya un obstáculo al frente. Es importante mencionar que todo el sistema es RESTfull y funciona bajo el protocolo de comunicación HTTP y los datos de envío y recepción son en formato JSON.

Frontend

La aplicación de control es una de las partes principales en el desarrollo del presente proyecto, ya que es la que permite el ingreso de las peticiones y también la toma de decisiones por parte del usuario dependiendo de las respuestas que se va recibiendo por parte del Back-End.

Figura 38

Estructura general del diseño de la arquitectura.



Estructura de la aplicación de control

Al tratarse de una aplicación que funciona tanto en computadores como en dispositivos móviles, la distribución de cada uno de sus componentes debe ser adecuado para brindar una experiencia satisfactoria al usuario y hacer el manejo de la misma mucho más fluida y cómoda. En la figura 39 se puede observar la aplicación de control vista desde un computador, mientras que en la figura 40 se puede ver la aplicación vista desde un dispositivo móvil, en este caso un celular.

Figura 39

Aplicación de control vista desde un computador.

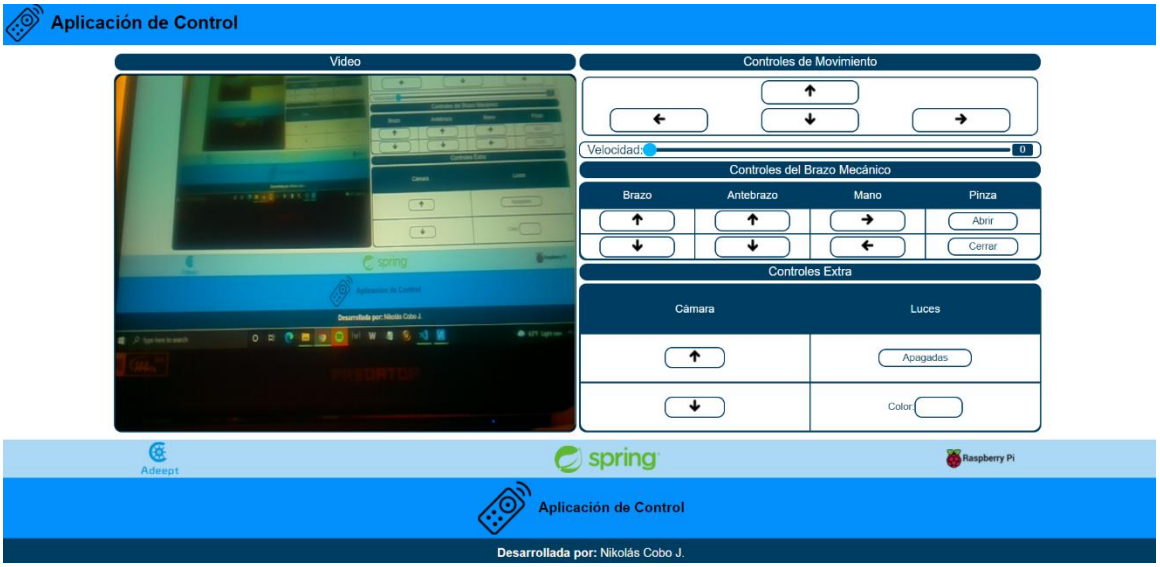


Figura 40

Aplicación de control vista desde un celular.

Como se pudo observar hay 4 componentes diferentes, el primero es el recuadro del video que trasmitirá el robot, el segundo son los controles de movimiento del robot, el tercero, son los controles del brazo mecánico del robot y el cuarto son los controles extra del robot, como por ejemplo sus luces. Adicionalmente existe un componente que no es visible en la ventana principal y es debido a que es él que gestiona las alertas en el frontend, dependiendo de la respuesta del robot y del backend.

Funcionamiento de los componentes de la aplicación de control

Cada componente de la aplicación de control funciona de una manera diferente, además de qué tienen un propósito diferente, es importante mencionar cuales son las prestaciones que cada uno brinda al usuario, a excepción del componente de video que se conecta directamente con el servidor interno del robot, todos los demás componentes pasan a través del back-end del sistema para cumplir con su funcionalidad, en la figura 41 se puede apreciar el funcionamiento de los componentes del front-end a excepción del componente de video, cada control o botón de la aplicación de control en el front-end se conecta con un “microservicio conceptual” específico en el back-end. Microservicio conceptual se refiere al concepto de la acción, por ejemplo “Mover adelante”, significa que el robot va a moverse hacia adelante y de esta manera se relaciona con cada botón en el front-end.

Componente de video. En la figura 42 se puede observar el componente de video de la aplicación de control, este básicamente ofrece la trasmisión en tiempo real que el robot este enviando en ese momento.

Figura 41

Funcionamiento de los componentes de la aplicación de control.

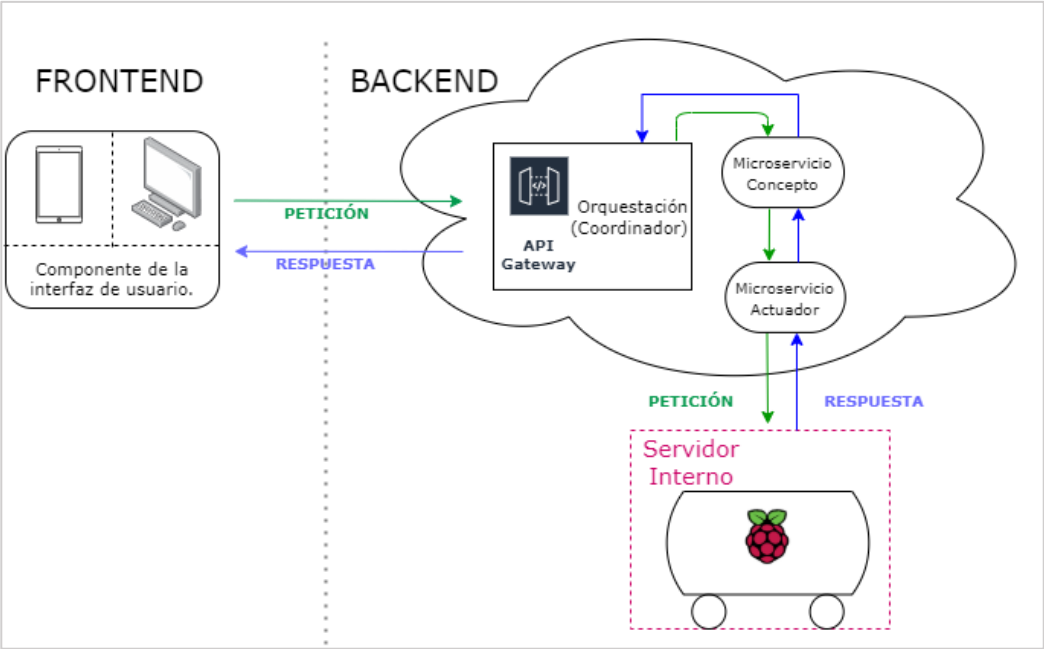
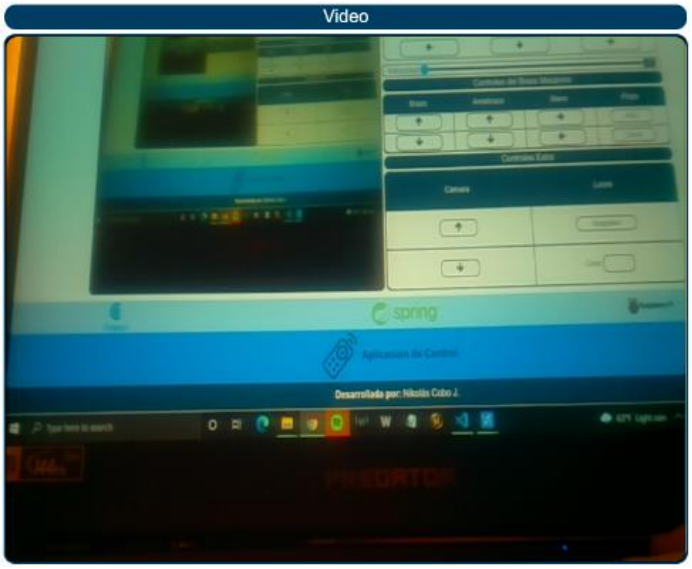


Figura 42

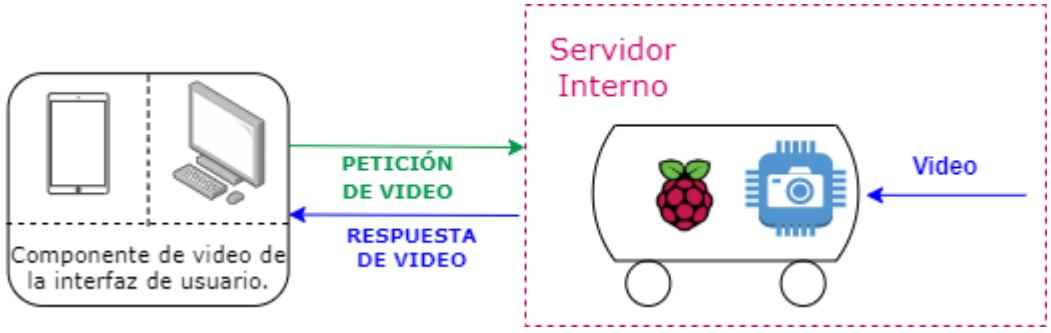
Componente de video de la aplicación de control



Para que el video pueda visualizarse, es necesario que el servidor interno del robot se encuentre funcionando, de esta manera la petición que el componente de video haga será atendida de manera correcta, caso contrario se visualizará el componente en negro. La transmisión del video se hace directamente del robot a la aplicación de control en el front-end y no pasa a través del back-end, ya que al ser en tiempo real no necesita validaciones de éxito o error. En la figura 43 se puede observar el funcionamiento del componente de video.

Figura 43

Funcionamiento del componente de video de la aplicación de control



Componente de controles de movimiento. En la figura 44 se puede observar la distribución de los controles de movimiento dentro del componente, su funcionamiento está ilustrado en la figura 41, cada uno de los botones de movimiento se conectan con un microservicio conceptual específico, en este caso en la tabla 23 se resume a que concepto va relacionado cada botón en la parte del backend.

Figura 44

Componente de controles de movimiento del robot

Tabla 23

Controles de movimiento y su relación con cada microservicio conceptual

Control de movimiento	Microservicio conceptual
Adelante ↑	Mover Adelante
Atrás ↓	Mover Atrás
Izquierda ←	Mover Izquierda
Derecha →	Mover Derecha

Componente de controles de brazo mecánico. En la figura 45 se puede ver la distribución de los controles de movimiento del brazo mecánico, como se puede ver están divididos en cuatro secciones: brazo, antebrazo, mano y pinza, esto debido a que en el robot cada una de estas partes funcionan de manera independiente. El funcionamiento de todos los controles de este componente está ilustrado en la figura 41 y cada uno se relaciona con un microservicio conceptual diferente, esta relación se encuentra resumida en la tabla 24.

Figura 45

Componente de controles de movimiento del brazo mecánico del robot.

Tabla 24

Controles de brazo mecánico y su relación con cada microservicio conceptual

Control de movimiento del brazo mecánico	Microservicio conceptual
Brazo	
Subir ↑	Subir Brazo
Bajar ↓	Bajar Brazo
Antebrazo	
Subir ↑	Subir Antebrazo
Bajar ↓	Bajar Antebrazo
Mano	
Izquierda ←	Mover Mano Izquierda
Derecha →	Mover Mano Derecha
Pinza	
Abrir	Abrir Pinza
Cerrar	Cerrar Pinza

Componente de controles extra del robot. El último componente que conforma la aplicación de control, es el de los controles extra del robot, como por ejemplo el manejo de las luces o el movimiento de la cámara, en la figura 46 se puede observar dicho componente, al igual que el resto de los controles del robot, el funcionamiento de estos controles se puede observar en la figura 41 y en la tabla 25 se puede ver el resumen de los microservicios conceptuales a los que están relacionados.

Figura 46

Componente de controles extra del robot.

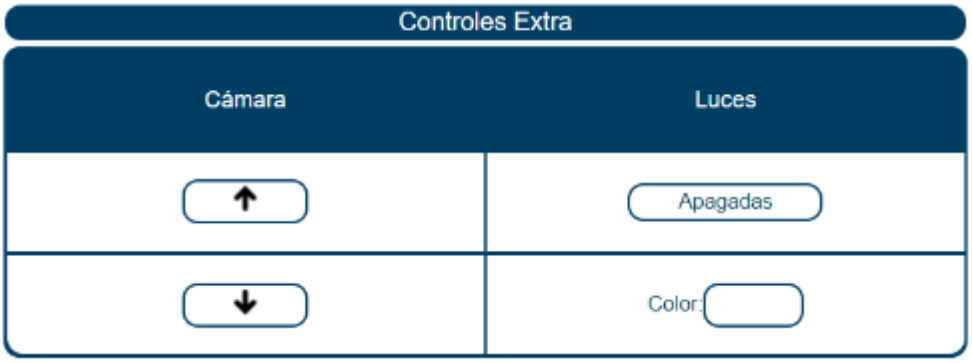


Tabla 25

Controles de brazo mecánico y su relación con cada microservicio conceptual

Control extra	Microservicio conceptual
Cámara	
Subir ↑	Subir Cámara
Bajar ↓	Bajar Cámara
Luces	
Apagadas/Encendidas	Apagar Luces / Encender Luces
Color	Cambiar Color Luces

Algo que es importante mencionar es que el control para cambiar luces del robot, dispone de algunos colores dentro de los cuales se puede escoger, en la tabla 26 se encuentra un resumen de dichos colores.

Tabla 26

Colores disponibles para las luces del robot

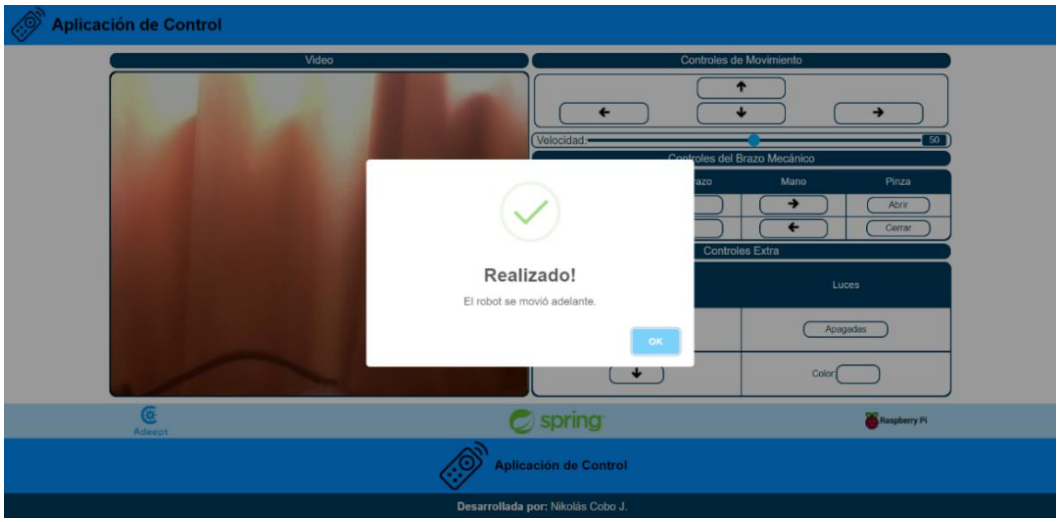
Descripción	Color
Rojo	
Verde	
Azul	
Amarillo	
Cian	
Magenta	
Blanco	

Componente de las alertas de la aplicación de control. Si bien este es un componente que no se puede visualizar directamente, es muy importante para la correcta interacción del usuario tanto con la aplicación de control, como con el sistema en general, ya qué, las alertas son las encargadas de notificar al usuario el estado acerca de la petición que realizó. Dentro de las alertas que se pueden presentar en la aplicación de control, se encuentran las siguientes:

Alerta de éxito. En la figura 47 se puede observar cómo se visualiza esta alerta en la aplicación de control, esta notificación se visualiza en caso de que la petición realizada haya sido exitosa.

Figura 47

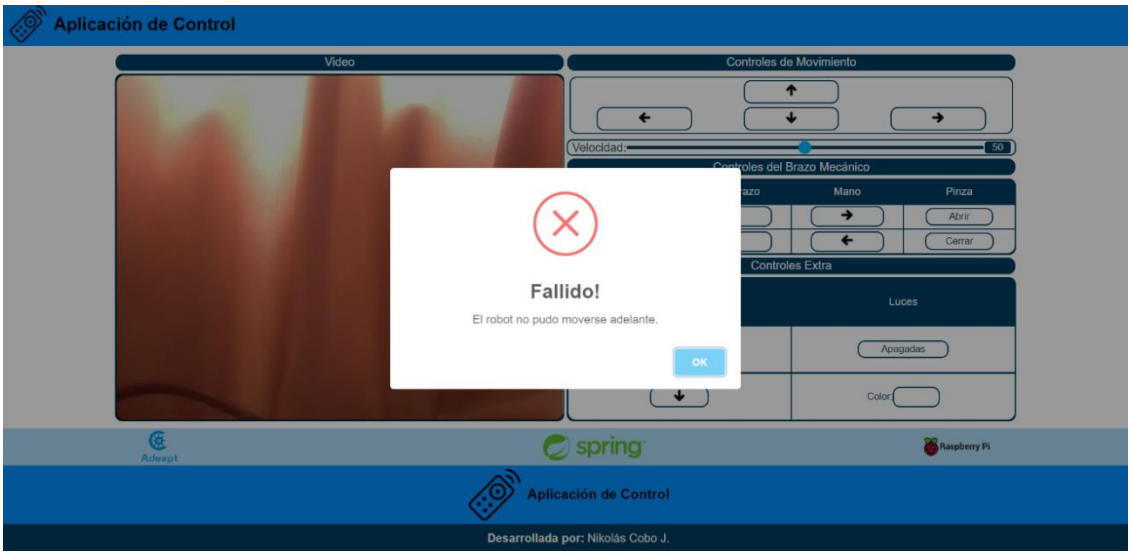
Alerta de éxito de la aplicación de control



Alerta de fallo. En la figura 48 se puede observar cómo se visualiza esta alerta en la aplicación de control, esta notificación se visualiza en caso de que la petición realizada haya fracasado.

Figura 48

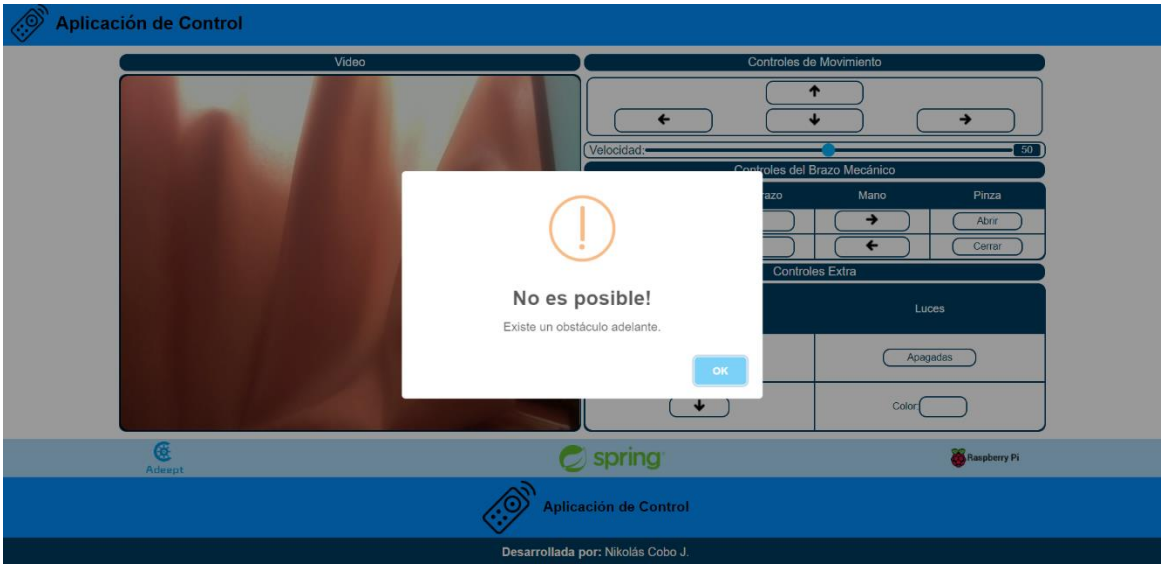
Alerta de fallo de la aplicación de control



Alerta de advertencia. En la figura 49 se puede observar cómo se visualiza esta alerta en la aplicación de control, esta notificación se visualiza en caso de que la petición realizada haya generado una advertencia, como por ejemplo que se encuentra un obstáculo frente al robot y no puede moverse.

Figura 49

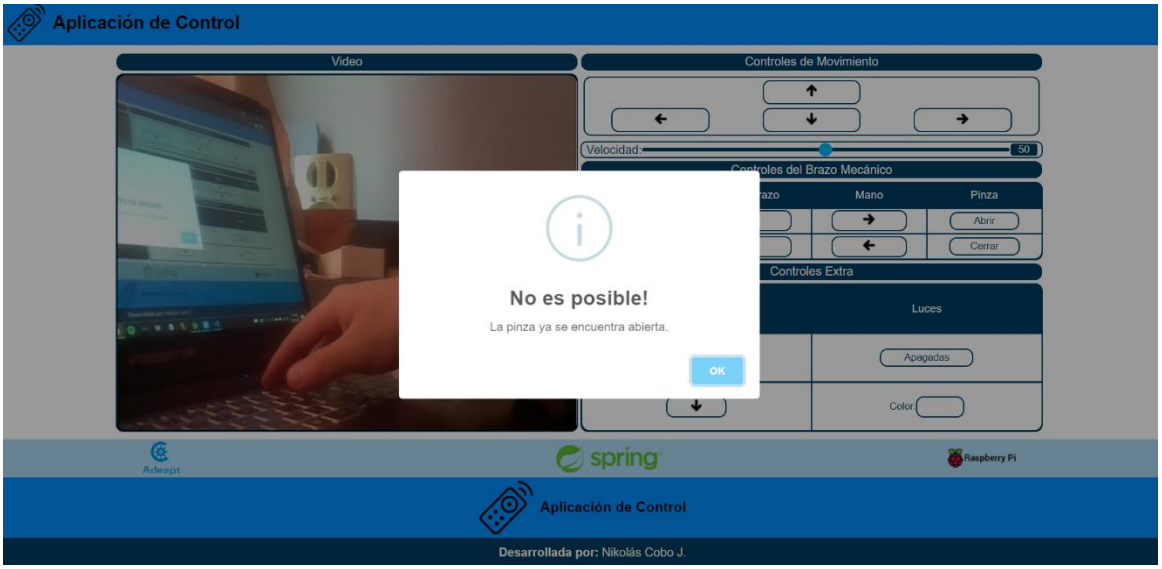
Alerta de advertencia de la aplicación de control



Alerta de información. En la figura 50 se puede observar cómo se visualiza esta alerta en la aplicación de control, esta notificación se visualiza en caso de que la petición realizada haya generado cierto caso en el que la acción no puede seguir realizándose, como por ejemplo cuando un servo motor ya ha alcanzado su límite y no puede seguir rotando en la misma dirección.

Figura 50

Alerta de información de la aplicación de control



Backend

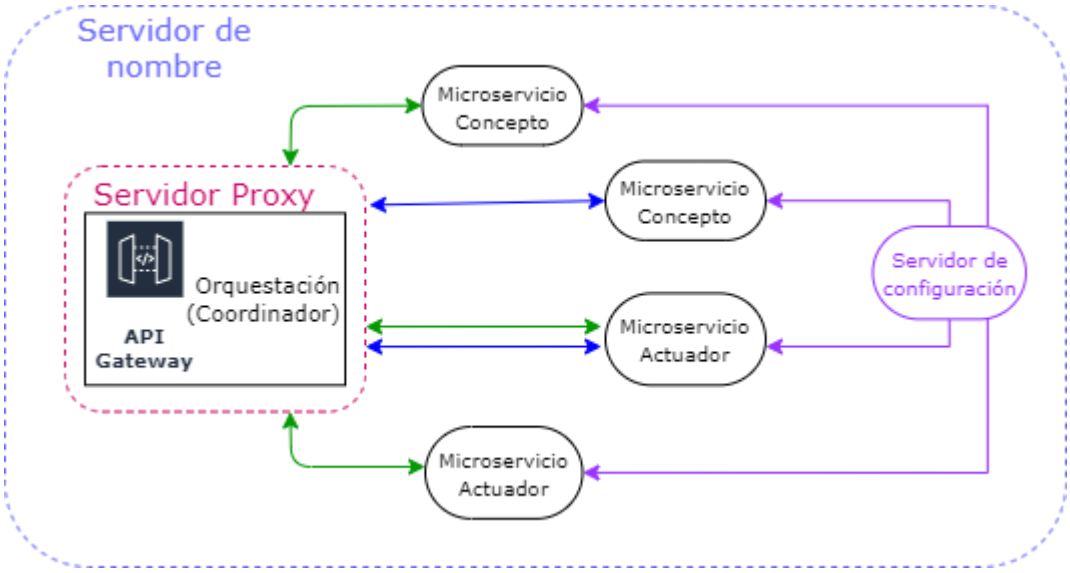
Si bien la aplicación de control puede ser remplazada por cualquier otro servicio capaz de enviar peticiones, el Backend no puede ser remplazado tan fácilmente y se vuelve la parte fundamental de este proyecto de titulación ya que, es el componente del sistema encargado de gestionar los datos para poder enviarlos tanto hacia el servidor del robot, como a la aplicación de control en el frontend, además es capaz de gestionar un fallo en caso de haberlo y finalmente al ser una arquitectura orientada a servicios, almacena la información en bases de datos independientes de cada uno de los microservicios implementados.

Estructura general del backend

El backend está conformado por varios componentes importantes, el enfoque del presente trabajo es que la arquitectura debe ser orientada a servicios y tolerante a fallos, de este modo es que la estructura general debe cumplir con ciertos requisitos para ser considerada adecuada para este proyecto. En la figura 51 se puede observar la estructura general del backend, para empezar cada uno de los microservicios deben estar registrados en un servidor de nombre, el que es encargado de monitorizar y localizar los microservicios existentes, en segundo lugar para la comunicación entre microservicios se debe tener un servidor proxy, encargado del enrutamiento entre los microservicios, además este servidor cumple también el papel de orquestador y de API Gateway, en tercer lugar, pero opcional se puede tener un servidor de configuración, en donde se pueden personalizar las propiedades de cada microservicio de manera externa. Algo importante que se debe mencionar, es que se podría decir que se tiene dos tipos de microservicios diferentes, aunque en estructura interna muy similares, se diferencian porque los microservicios conceptuales son los que gestionan los datos que se enviarán a los microservicios actuadores, los cuales son los que se comunicarán directamente con los actuadores del robot. Finalmente se debe tener en cuenta que absolutamente todos los componentes del backend son considerados microservicios, tanto el servidor de nombre, el servidor proxy, el servidor de configuración y los propios microservicios para la aplicación práctica.

Figura 51

Estructura general del backend



Funcionamiento general del backend

Como ya se ha podido ver en secciones anteriores, básicamente el papel que cumple el backend es el de parte central del sistema, donde se gestionan los datos y se envían tanto al frontend como respuestas o al servidor del robot como peticiones, en la figura 52 podemos ver un poco del funcionamiento general del backend, en primer lugar se tiene al API Gateway qué recibe las peticiones enviadas desde el frontend, esté al cumplir también el papel de un proxy procede a enrutar la información a los microservicios conceptuales a donde se quiere acceder, en el microservicio conceptual procesa la información y a través del proxy accede al microservicio actuador, el cual envía la petición al servidor interno del robot, adicionalmente cabe mencionar que cada uno de los microservicios tanto conceptuales como actuadores son tolerantes a fallos, esto quiere decir que en caso de que acontezca un error, se retornará al estado anterior

para evitar un fallo en cascada en todo el sistema. Para un mayor entendimiento, en la figura 53 se encuentra un diagrama de flujo que ilustra mejor este proceso.

Figura 52

Funcionamiento general del backend

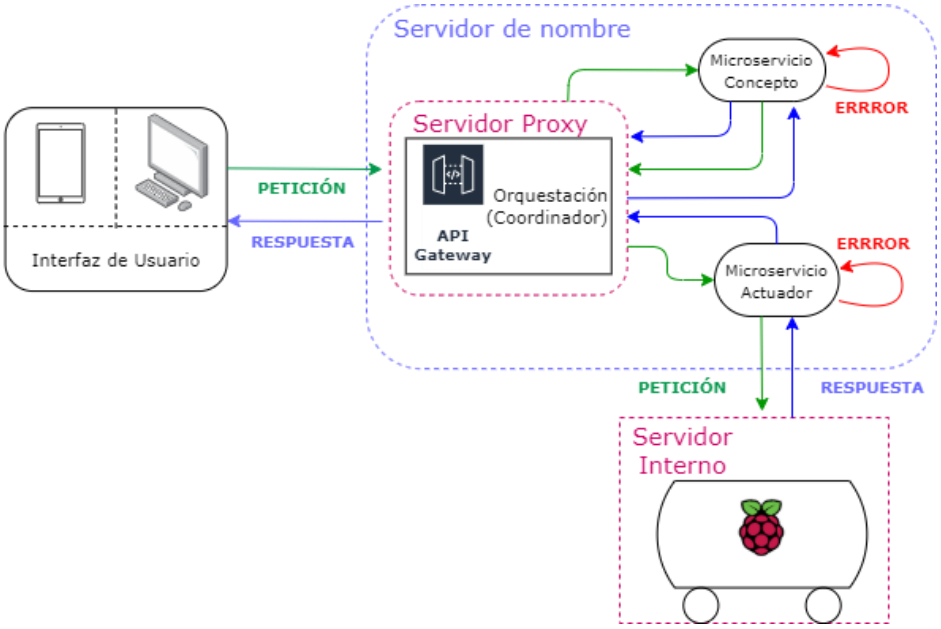
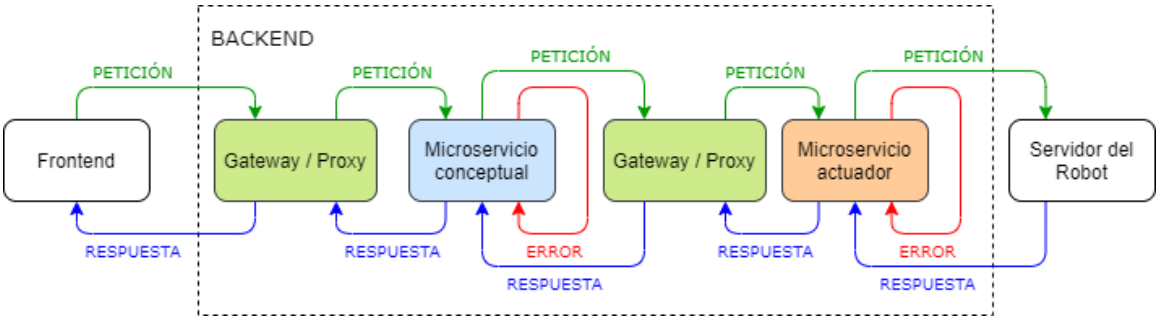


Figura 53

Diagrama de flujo del funcionamiento general del backend



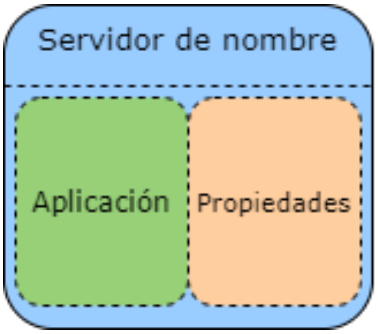
Estructura interna de los microservicios

Cada uno de los microservicios cumplen un papel fundamental para el funcionamiento adecuado de todo sistema desarrollado en el presente trabajo de titulación, dentro de los microservicios existen algunos muy importantes como los encargados de la configuración, el enrutamiento y el registro de los mismos, y también existen otros microservicios que son los que se encargan de gestionar los datos y tomar las decisiones correspondientes para cada petición.

Servidor de nombre. Este microservicio es el más importante en una arquitectura de microservicios donde se quiera organizar, localizar y registrar los microservicios, ya qué, se encarga de realizar todas estas tareas y adicionalmente brinda la versatilidad necesaria si este fuese el caso. En la figura 54 se puede observar su estructura interna, la cual es bastante sencilla, ya que solo consta de su componente de aplicación y su componente de propiedades, pero su funcionalidad es muy potente.

Figura 54

Estructura del servidor de nombre



Servicio de Gateway, Proxy y Orquestación. Este microservicio es fundamental en una arquitectura donde se quiera utilizar una orquestación de procesos, ya que cumple el papel de control activo u orquestador, y es el encargado de gestionar todas las interacciones necesarias que deben haber con los microservicios dependiendo de las peticiones que el usuario vaya realizando, además en un sistema RESTfull como el presentado en el presente proyecto, también cumple el papel de API Gateway, es decir la puerta de enlace para cualquier servicio externo que quiera comunicarse con el backend. En la figura 55 se puede observar su estructura interna y algo importante que mencionar es que este microservicio en sus propiedades alberga la información de enrutamiento del resto de microservicios, finalmente es el que puede calcular tiempos de respuesta gracias a que posee filtros internamente.

Figura 55

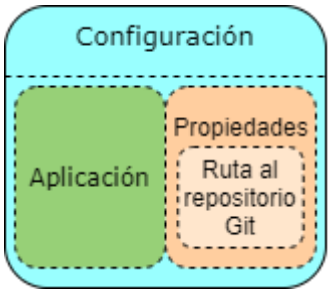
Estructura del servicio de Gateway, Proxy y Orquestación

Servicio de configuración. Este microservicio puede ser opcional, ya que son las configuraciones que se pueden hacer en las propiedades intrínsecas de cada microservicio específicamente, pero la utilización de un microservicio para la configuración puede facilitar la organización y el manejo de los microservicios cuando se tiene gran cantidad, de este modo se puede acceder y variar la configuración que se

desee inclusive de manera remota, ya que los archivos de configuración de cada servicio deben estar albergados en algún repositorio Git local o en la nube. En la figura 56 se puede ver la estructura de este microservicio y es importante mencionar que en sus propiedades debe constar la información de la ruta al repositorio Git.

Figura 56

Estructura del servicio de configuración



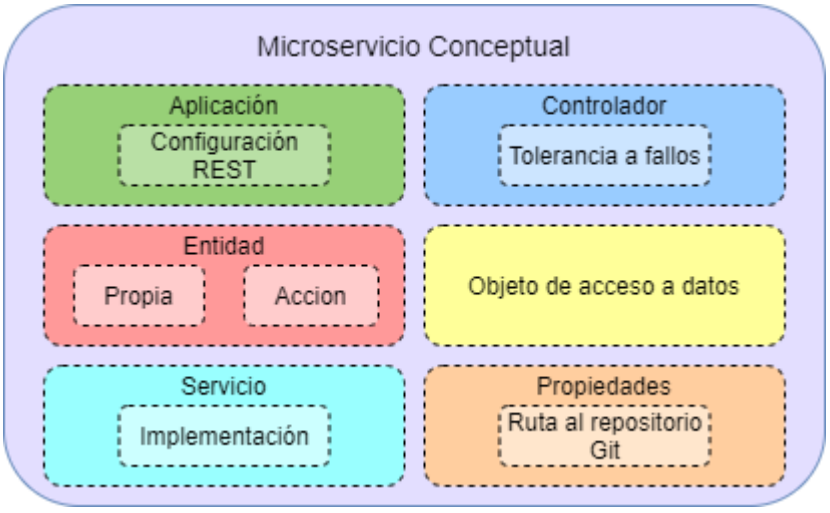
Microservicios conceptuales. Dentro de la arquitectura y la conexión de los microservicios, los conceptuales son el nexo primario para acceder a los microservicios actuadores, se les ha denominado “conceptuales” debido a que son orientados a un concepto o una acción en particular que se envía desde el frontend y que se desea que realice el robot.

Estructura de los microservicios conceptuales. Cada uno de estos microservicios tienen una estructura similar, en la figura 57 se puede observar cómo estar conformados, en primer lugar se tiene la aplicación, además se tiene la configuración de cliente REST debido a que todo el sistema es RESTful, en segundo lugar se tiene la entidad que puede ser propia del microservicio o estar relacionada con el microservicio actuador al que se quiere conectar, en tercer lugar se tiene el controlador que es el punto central del microservicio, encargado de gestionar los datos

tanto para recepción, envío y en caso de algún error, en cuarto lugar se tiene el objeto de acceso a datos utilizado para utilizar las funciones CRUD dentro del microservicio, en quinto lugar se tiene la implementación del servicio como tal, donde se definen todos los métodos utilizados para la comunicación entre los microservicios, y finalmente se tiene las propiedades, donde se encuentra la información acerca del servidor de configuración, como su ruta en el repositorio Git.

Figura 57

Estructura del microservicio conceptual

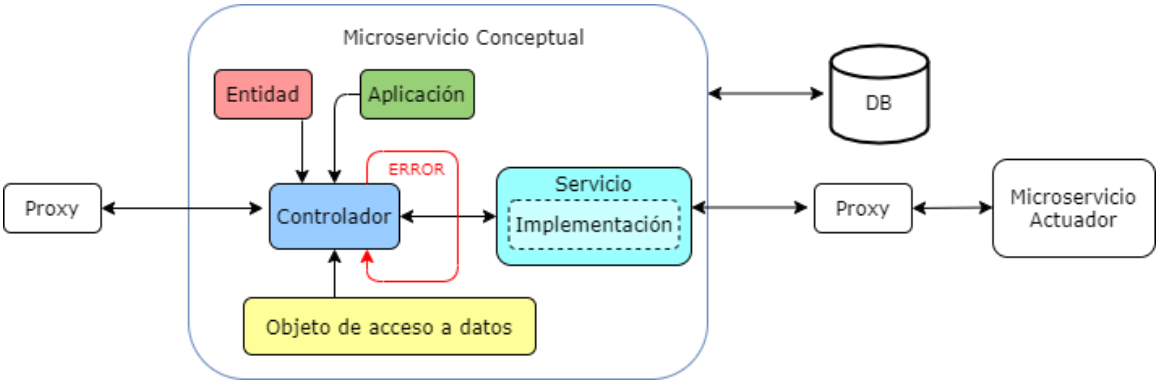


Funcionamiento de los microservicios conceptuales. En cuanto al funcionamiento de un microservicio conceptual, se puede decir qué los componentes que están directamente relacionados en la interacción del envío y recepción de los datos son el controlador y el servicio, el resto son componentes que aportan directamente, pero a la parte de la configuración del controlador para que la aplicación funcione correctamente. El proceso que se realiza internamente en el microservicio se genera a partir de la petición que el servidor proxy envía, a continuación, el controlador

que ya tiene la información de la aplicación, la entidad y el objeto de acceso a datos, recibe la información y gestiona los datos ya sea para guardarlos, enviarlos, o en caso de error acceder a un método de respaldo, finalmente la información pasa a la implementación del servicio, donde se procede a enviarla al proxy y a los microservicios actuadores correspondientes. Es importante mencionar que la recepción de los datos se hace de manera inversa, y que el almacenamiento de los datos se hace en una base de datos propia e independiente del microservicio, en la figura 58 se puede apreciar este proceso de manera gráfica.

Figura 58

Funcionamiento del microservicio conceptual



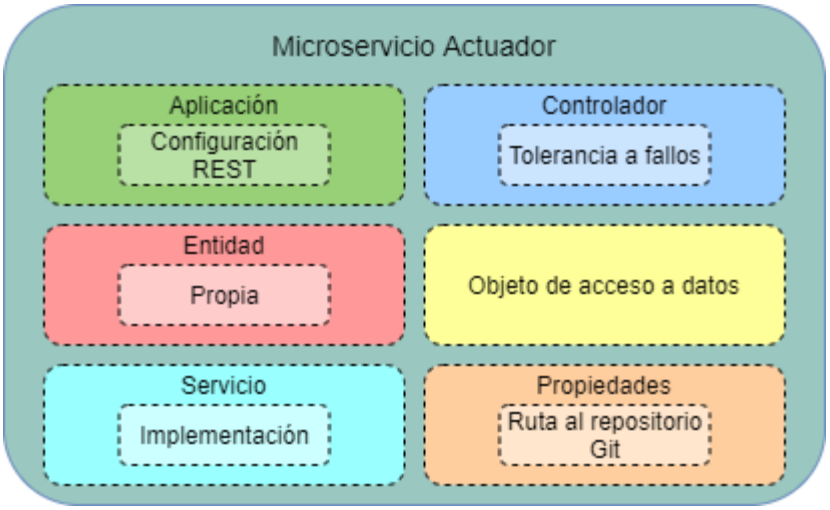
Microservicios actuadores. Después de los microservicios conceptuales, los microservicios actuadores son los encargados de establecer la conexión con el robot, se les denomina “actuadores” ya que están relacionados directamente con los actuadores del robot, que en su mayoría son motores.

Estructura de los microservicios actuadores. Su estructura es muy similar a la de los microservicios conceptuales, en la figura 59 se puede observar cómo estar conformados, en primer lugar se tiene la aplicación, además se tiene la configuración de

cliente REST debido a que todo el sistema es RESTful, en segundo lugar se tiene la entidad que en este caso es únicamente propia, en tercer lugar se tiene el controlador que es el punto central del microservicio, encargado de gestionar los datos tanto para recepción, envío y en caso de algún error, en cuarto lugar se tiene el objeto de acceso a datos utilizado para utilizar las funciones CRUD dentro del microservicio, en quinto lugar se tiene la implementación del servicio como tal, donde se definen todos los métodos utilizados para la comunicación entre los microservicios, y finalmente se tiene las propiedades, donde se encuentra la información acerca del servidor de configuración, como su ruta en el repositorio Git.

Figura 59

Estructura del microservicio actuador

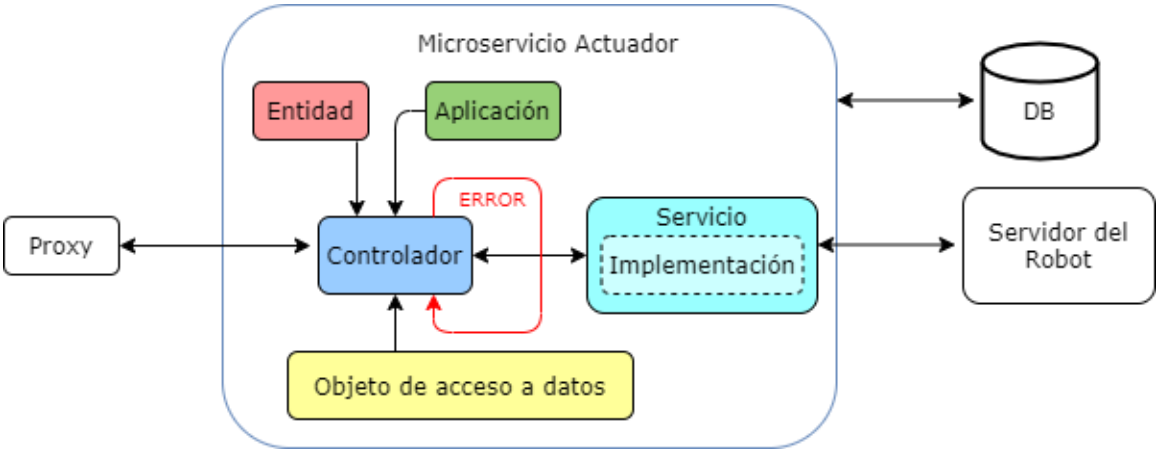


Funcionamiento de los microservicios actuadores. En cuanto al funcionamiento de un microservicio actuador, es muy similar a la de un conceptual, se puede decir qué los componentes que están directamente relacionados en el envío y recepción de los datos son el controlador y el servicio, el resto son componentes que aportan directamente, pero a la parte de la configuración del controlador para que la

aplicación funcione correctamente. El proceso que se realiza internamente en el microservicio se genera a partir de la petición que envía el servidor proxy, a continuación, el controlador que ya tiene la información de la aplicación, la entidad y el objeto de acceso a datos, recibe la información y gestiona los datos ya sea para guardarlos, enviarlos, o en caso de error acceder a un método de respaldo, finalmente la información pasa a la implementación del servicio, donde se procede a enviarla al servidor del robot y a sus actuadores correspondientes. Es importante mencionar que la recepción de los datos se hace de manera inversa y que el almacenamiento de los datos se hace en una base de datos propia e independiente del microservicio, en la figura 60 se puede apreciar este proceso de manera gráfica.

Figura 60

Funcionamiento del microservicio actuador



Servidor del robot

Para poder demostrar la aplicabilidad práctica de toda la arquitectura de software desarrollada es importante contar con otro de los pilares fundamentales en esté trabajo de titulación y es un sistema ciber físico, la mejor opción es un robot que cuente con la disponibilidad para implementar internamente un microcontrolador que a

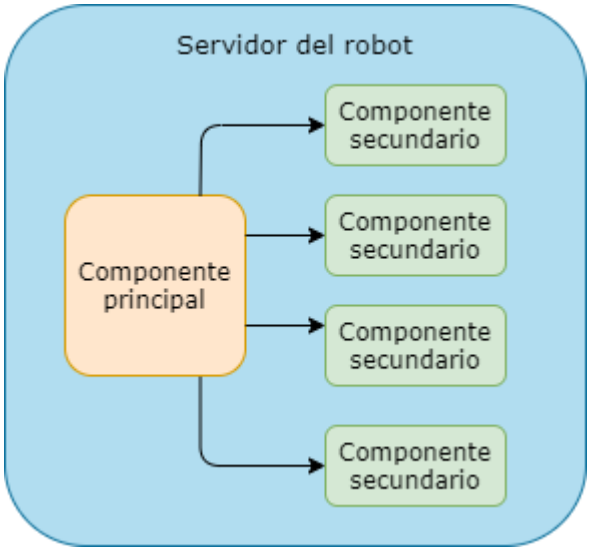
su vez permita levantar un servidor o una aplicación web, adicionalmente deberá contar con los sensores y actuadores necesarios para un despliegue adecuado de todas las funcionalidades que se deseen implementar.

Estructura del servidor del robot

La estructura del servidor del robot consta de varios componentes importantes, en la figura 61 se puede apreciar la distribución de dichos componentes, la parte principal del servidor es la encargada de gestionar los datos que recibe del backend y más específicamente de los microservicios actuadores, en segundo lugar, tenemos los componentes secundarios, los cuales están destinados a recibir la información del componente principal e interactuar directamente con los actuadores del robot. Es importante decir que cada uno de los componentes tiene una configuración propia internamente.

Figura 61

Estructura del servidor del robot

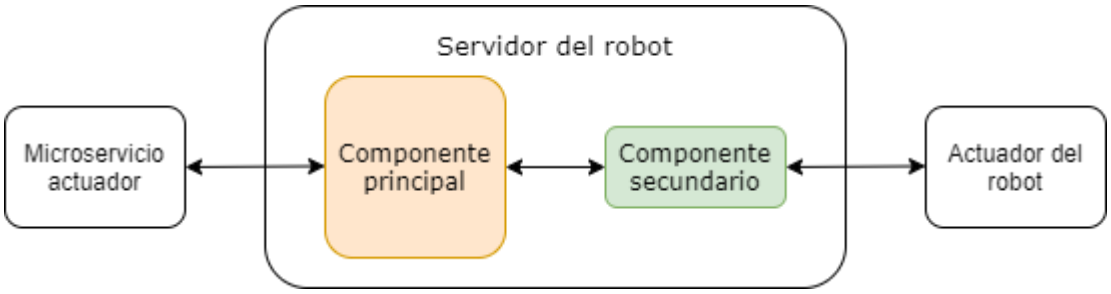


Funcionamiento del servidor del robot

En cuanto al funcionamiento del servidor del robot, en primer lugar se tiene al componente principal, el cual es el encargado de recibir la información por parte de los microservicios actuadores, una vez la información es recibida, se procesa y pasa al componente secundario correspondiente, ya que cada uno está relacionado directamente con los actuadores físicos del robot, una vez la acción es realizada, se devuelve la respuesta, que sigue el mismo camino pero de manera inversa, en la figura 62 se puede apreciar el funcionamiento de este servidor, de manera gráfica.

Figura 62

Funcionamiento del servidor del robot



Capítulo IV

Implementación de la Arquitectura

Como se pudo ver a lo largo del capítulo 3, se trató acerca de las consideraciones de diseño que debe tener la arquitectura que se plantea en el presente proyecto, esto junto a las bases del estudio y la investigación realizada en el capítulo 2, nos ayudan a poder definir de manera adecuada todas las herramientas y tecnologías más óptimas para la implementación de una arquitectura que como el título del presente proyecto indica, debe estar destinada al control telemático de sistemas ciber físicos, ser orientada a servicios y finalmente tolerante a fallos. A lo largo del presente documento se han ido presentando bosquejos de cada una de las etapas de desarrollo y se ha utilizado diagramas similares, esto con el fin de poder visualizar el proceso que se ha ido siguiendo hasta llegar al producto final. Para no separarse de esta línea de diseño e implementación, en el presente capítulo de igual manera se irán reutilizando diagramas, con la diferencia de que en los mismos ya se ira presentando detalles puntuales como código de programación, librerías, herramientas y tecnologías utilizadas, además se detallará de manera muy minuciosa cada uno de los componentes que conforman el sistema implementado.

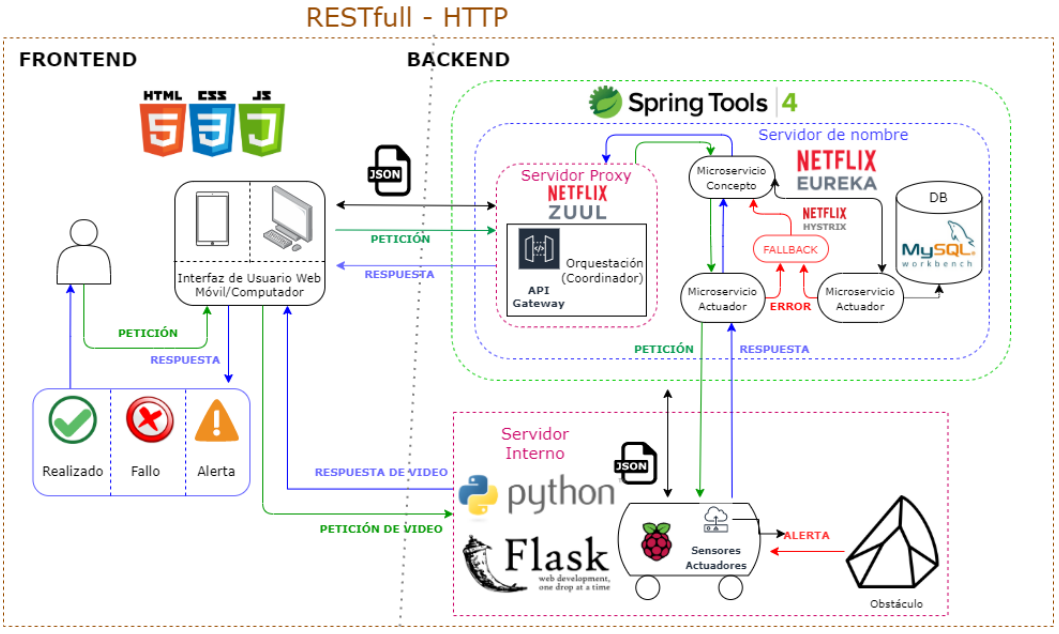
Estructura general de la arquitectura implementada

Para empezar, lo más importante es conocer que herramientas y tecnologías se utilizó para la implementación de la arquitectura, ya se han presentado diagramas de la estructura de la arquitectura, pero solo de manera conceptual. En la figura 63 se puede observar la estructura implementada, en primer lugar, tenemos el frontend, para su implementación se utilizó HTML, CSS y JavaScript, de este modo la aplicación puede

ser *responsive*, es decir que puede ser utilizada desde múltiples dispositivos, después se tiene el backend, para su implementación se utilizó SpringBoot, con su framework Spring Tools 4, dentro de este se utilizó la librería open source Spring Cloud Netflix, una librería para algunos de sus componentes, como por ejemplo Netflix Eureka para el servidor de nombre, Netflix Zuul para el servidor proxy y Netflix Hystrix para la tolerancia a fallos dentro de los microservicios, para gestionar las bases de datos de los microservicios se utiliza MySQL Workbench finalmente para el servidor interno del robot, en la parte del Hardware se utilizó una Raspberry Pi Model 3B+ y en la parte del software que se desarrolló dentro de la Raspberry se utilizó el lenguaje de programación Python, con el cual se utilizó un framework embebido llamado Flask, el cual permite la implementación de aplicaciones o servicios Web. En cuanto al funcionamiento, se abordará posteriormente en el capítulo ya que, antes es necesario detallar algunos otros elementos importantes.

Figura 63

Estructura general de la arquitectura implementada



Hardware

Para una mejor compresión de la implementación de la arquitectura de software es importante comenzar con la descripción del Hardware utilizado para la aplicabilidad practica del sistema desarrollado, debido a qué, conocer con qué fin se diseñó e implemento cada uno de los componentes tanto del frontend como del backend ayudaran a entender por qué y para qué se utilizó cada una de las herramientas y tecnologías en la implementación de la arquitectura final.

Descripción del robot

El robot utilizado es un RaspTank Smart Robot Car, creado y distribuido por Adeept, una empresa de manufactura y desarrollo de Hardware y Software Open Source, cada kit de desarrollo que ellos ofertan está orientado ya sea a Arduino o Raspberry Pi, en el caso del presente proyecto se utiliza una Raspberry como el controlador del robot. Como ellos mismo definen al robot es qué está enfocado para una educación práctica tanto de la electrónica como del desarrollo de software, ya que se aborda estas dos ramas de la ciencia en su producto. En la figura 64 se puede observar cómo luce el robot físicamente.

Estructura externa del robot

Dentro del Kit para la implementación del robot, se encuentran tanto los sensores y actuadores necesarios para su funcionamiento, así como su estructura interna, externa y sus componentes mecánicos para que todo pueda armarse y juntarse adecuadamente. Con lo único que no cuenta directamente el Kit es con la Raspberry Pi, la cual debe ser adquirida independientemente. Para un mayor entendimiento de cada uno de los componentes del robot, es necesario abordarlos individualmente.

Figura 64

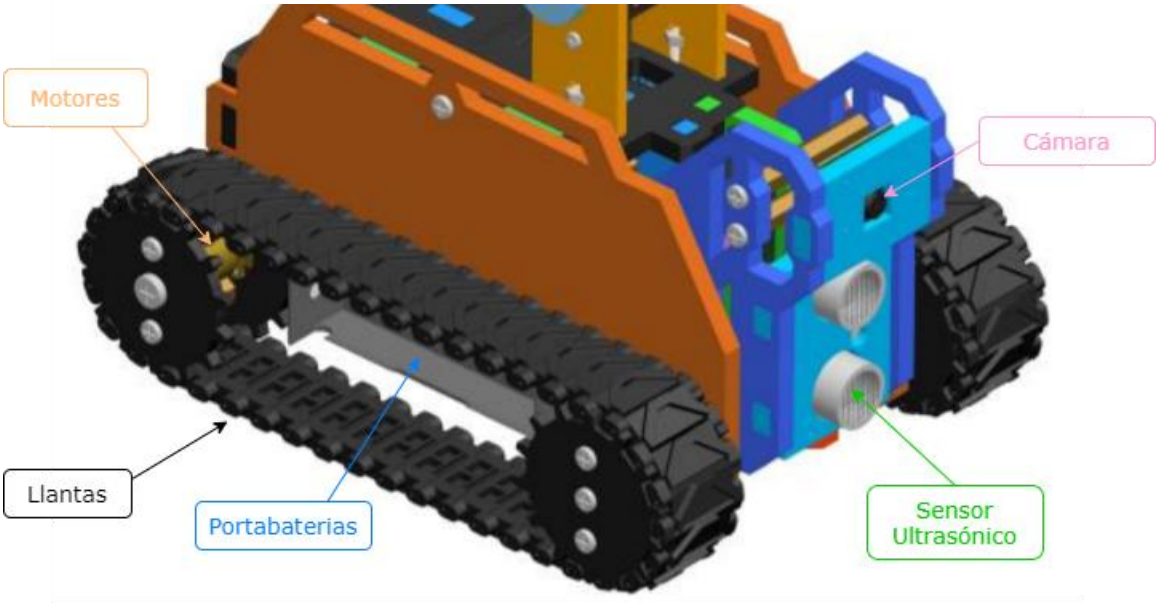
RaspTank Smart Robot Car

Nota: Figura tomada de (Adeept, 2018).

Base del robot. Internamente en la base del robot es donde se encuentra la Raspberry Pi, junto con el resto de componentes importantes para que pueda funcionar correctamente, en la parte externa, además de los recubrimientos, consta de algunos elementos que se pueden visualizar en la figura 65, en primer lugar se tiene las llantas que son orugas para facilitar el movimiento y agarre en cualquier superficie, estas se mueven gracias a los motores qué se encuentran en la parte posterior de la base del robot, en segundo lugar en el espacio intermedio de las llantas se tiene los porta baterías que permiten la portabilidad del robot, y finalmente en la parte frontal s e tiene tanto la cámara como el sensor ultrasónico.

Figura 65

Estructura de la base del robot

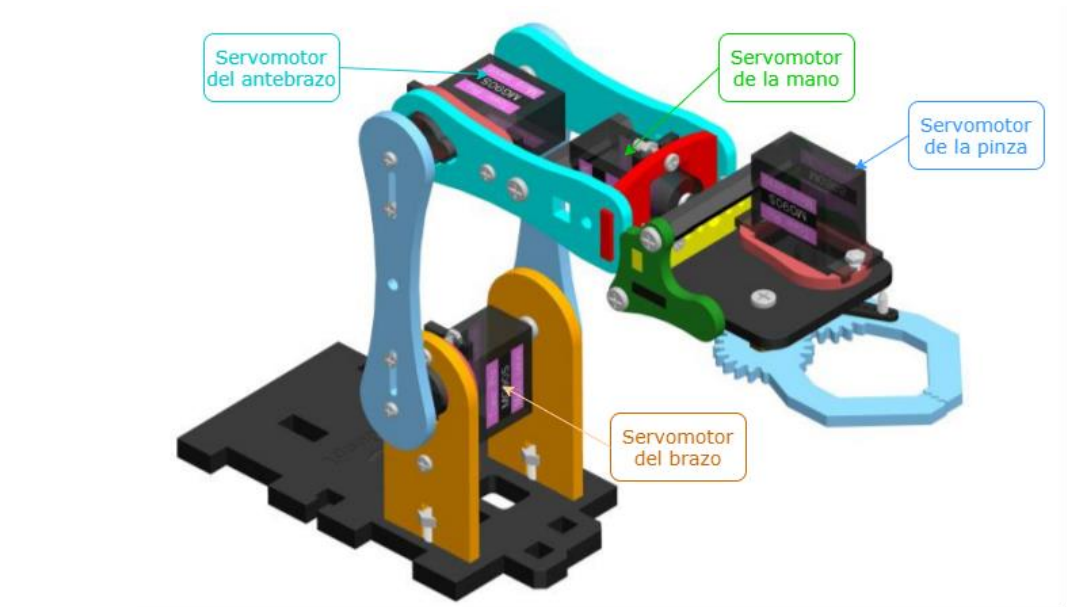


Nota: Figura adaptada y tomada de (Adeept, 2018).

Brazo del robot. Sobre la base del robot se encuentra el brazo del robot, el cual brinda más aplicaciones prácticas sobre todo para entornos de aprendizaje y pruebas, la estructura del brazo puede verse en la figura 66, en primer lugar, se tiene el servo motor del brazo, encargado de la movilidad total de la estructura, en segundo lugar, se tiene el antebrazo y su servomotor, en tercer lugar, se tiene la mano y su servomotor y finalmente se tiene la pinza y su servomotor, cada una de estas partes están limitadas en movimiento por la estructura en sí y por el rango de sus servomotores.

Figura 66

Estructura del brazo del robot



Nota: Figura adaptada y tomada de (Adeept, 2018).

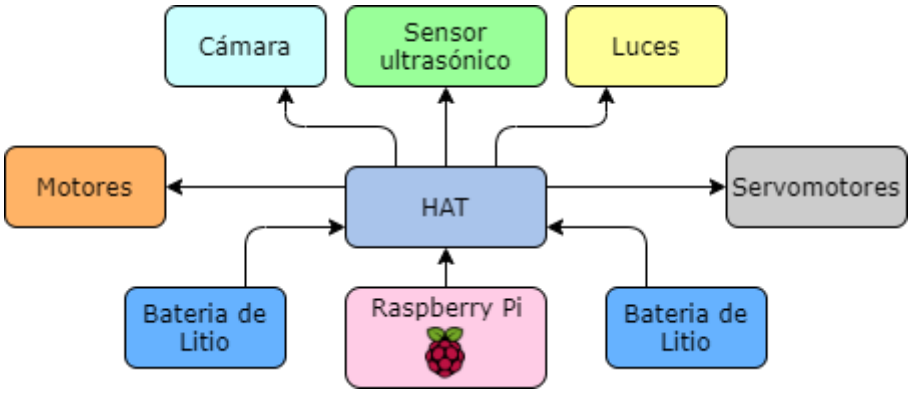
Estructura interna del robot

Como ya se mencionó, internamente en la base del robot se encuentran los componentes electrónicos necesarios para su funcionamiento, ya que el propósito de este capítulo es ahondar en los detalles importantes de funcionamiento, más no en detalles como el ensamblaje del robot, en la figura 67 se puede observar un bosquejo de cómo se encuentran las conexiones de todos los componentes y módulos utilizados. En primer lugar, como núcleo se tiene a la Raspberry Pi, sobre la Raspberry se encuentra un HAT (Hardware adjunto a la parte superior) el cual es encargado de gestionar conexiones, voltajes y amperajes necesarios para el resto de componentes, a continuación se tienen todos los componentes utilizados que van conectados a puertos específicos del HAT, como los servomotores de cada parte, los motores de movimiento,

las luces, la cámara y el sensor ultrasónico, finalmente se tiene la alimentación que es suministrada por dos baterías de litio que se encuentran en los porta baterías.

Figura 67

Estructura interna del robot



Especificaciones técnicas de los componentes del robot. Si bien ahondar en las conexiones no es uno de los objetivos importantes, el conocer cada uno de los elementos o módulos utilizados es pertinente para tener una idea de la energía consumida y de las prestaciones disponibles. En la tabla 27 se presenta un listado y resumen de los componentes electrónicos utilizados con algunas de sus especiaciones técnicas. Es importante mencionar que todos los elementos venían dentro del propio Kit de ensamblaje que la empresa distribuidora oferta, por lo que por lo general se trata dispositivos que requieren una fuente de alimentación similar y que consumen una cantidad de energía acorde a la capacidad máxima que el HAT del robot tiene disponible, en caso de que se quiera usar algún elemento extra seria pertinente verificar el manual de usuario del robot antes.

Tabla 27

Especificaciones técnicas de los componentes electrónicos del robot

Componente electrónico o módulo	Voltaje de alimentación	Corriente consumida	Característica extra
Raspberry PI Model 3B+.	5 V.	2.5 A.	Cantidad: 1.
HAT (Hardware adjunto a la parte superior).	5 V.	1,5 - 2 A.	Cantidad: 1.
Motor DC.	1,5 – 9 V.	280 - 300 mA.	Cantidad: 2.
Servomotores.	4 – 6 V.	150 – 250 mA.	Cantidad: 5. Rotación: 180º.
Módulo de sensor ultrasónico HC SR04.	2 – 5 V.	2 – 5 mA.	Cantidad: 1. Rango: 1 a 10 m.
Raspberry PI Camera Module V2	5 V.	3 – 5 mA.	Cantidad: 1. Resolución: 8 MP. Video: 1080p 30fps.
Modulo WS2812 RGB LED	4.5 V – 5V.	20 mA.	Cantidad: 1. Colores definidos por RGB.
Baterías de Litio 18650	Voltaje que suministra: 3.7 a 5 V	Corriente que suministra: Hasta 3000 mAh.	Cantidad: 2. Requiere 2 porta baterías.

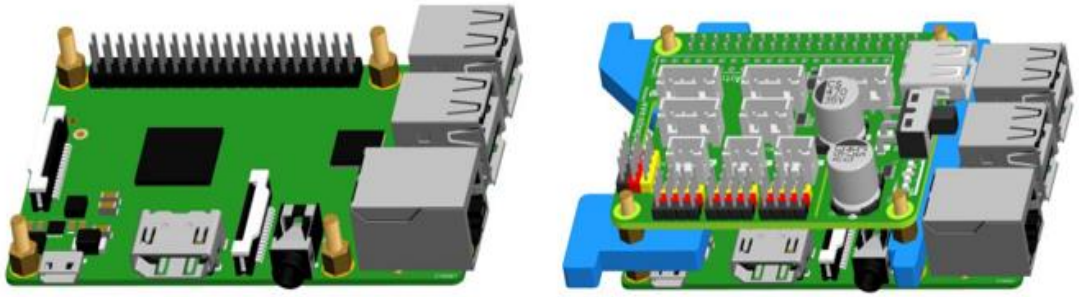
Como dato adicional se puede mencionar que el HAT dispone de un Switch con el que se puede encender o apagar el robot.

Conexiones de los componentes del robot. Con los componentes electrónicos y sus especificaciones técnicas bien definidas, se procede a profundizar en las conexiones que tienen estos componentes dentro del robot y con la Raspberry Pi. Como ya se mencionó, todos los elementos, tanto sensores como actuadores no están conectados directamente con la Raspberry Pi, sino que se tiene un HAT encargado de

las conexiones y de gestionar los voltajes y las corrientes que cada uno de los componentes necesita para funcionar correctamente. En primer lugar, en la figura 68 se puede ver en la parte izquierda a la Raspberry Pi y en la parte derecha al HAT después de ensamblarlo con ciertas piezas del robot.

Figura 68

Raspberry PI y ensamblaje del HAT

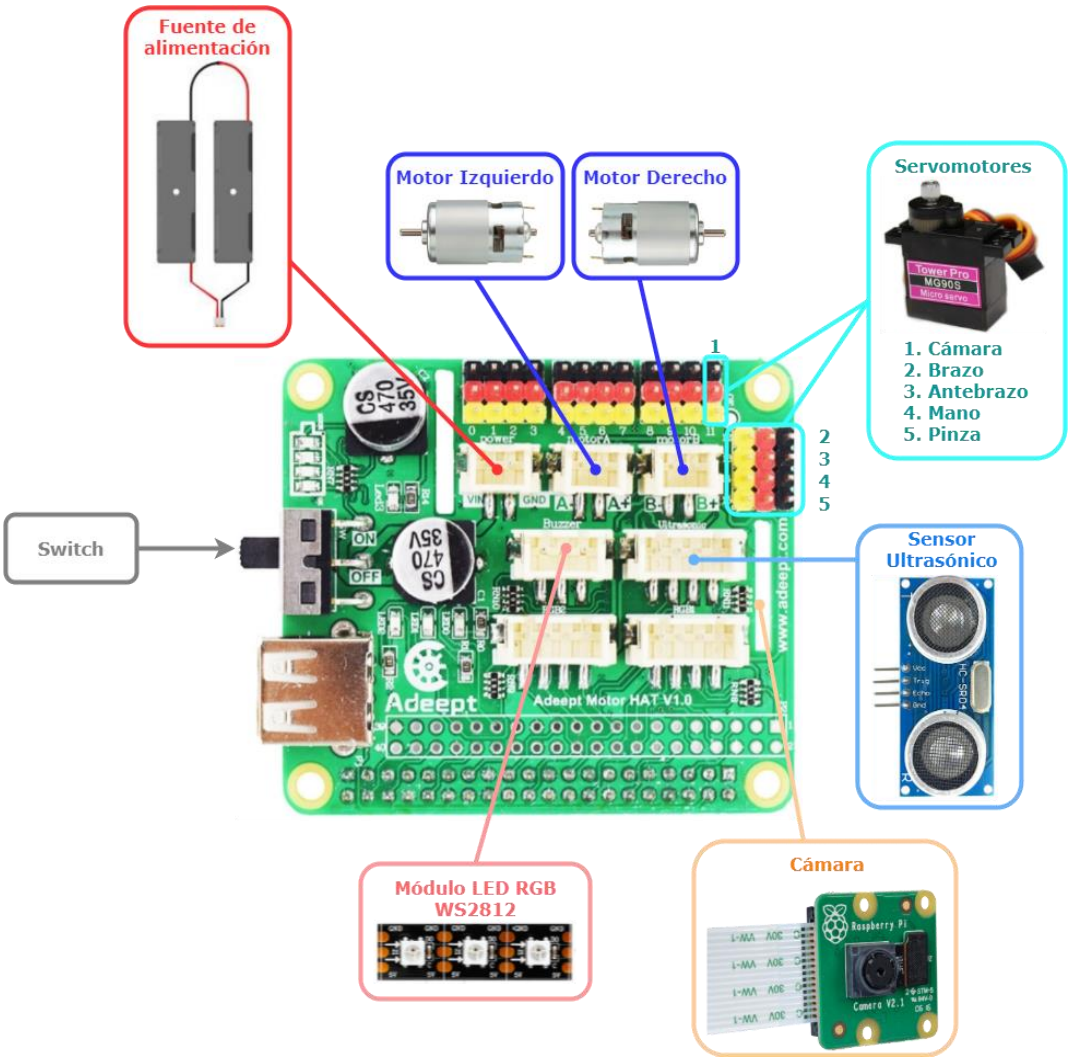


Nota: Figura adaptada y tomada de (Adeept, 2018).

Una vez que el HAT se encuentra ensamblado se puede conectar los componentes utilizados en los pines o puertos específicos que están proporcionados en el manual de ensamblaje del robot, en la figura 69 se puede observar las conexiones de cada uno de los elementos electrónicos y por ejemplo en el caso de los servomotores se puede ver el orden en el que deben ir colocados. Todas las conexiones son realizadas con los cables proporcionados en el Kit del robot y son específicos para cada puerto del HAT. Adicionalmente es importante mencionar que la cámara se conecta directamente al puerto de la Raspberry Pi a través de la ranura señalada en la imagen.

Figura 69

Conexiones de los componentes electrónicos al HAT



Ya conocidas las conexiones que cada componente tiene directamente con el HAT, con fines prácticos y para entender mejor el código de cada uno de los archivos del servidor, es importante hacer mención a los respectivos pines de la Raspberry que se comunican con el HAT para controlar cada uno de los componentes, en la tabla 28 se encuentran detallados los elementos electrónicos que utilizan pines, que se

denominan GPIO en la Raspberry Pi y también el tipo de comunicación, que puede ser PWM o HIGH - LOW que hace referencia a los voltajes altos y bajos.

Tabla 28

Componentes electrónicos con sus pines GPIO y su tipo de comunicación

Componente electrónico	Pines GPIO	Tipo de comunicación
Servomotor antebrazo	13	PWM
Servomotor brazo	12	PWM
Servomotor cámara	11	PWM
Servomotor mano	14	PWM
Servomotor pinza	15	PWM
Motor izquierdo	17, 27 y 18	HIGH - LOW
Motor derecho	4, 14 y 15	HIGH - LOW
Sensor ultrasónico	11 y 8	HIGH - LOW
Luces	12	PWM

Funcionamiento del robot

Una vez que se ha definido cada una de las partes del robot, es importante saber cuál es su funcionalidad y las prestaciones que puede ofrecer, pero ya que esta divido por partes, cada una de estas es encargada de una función en específico, en la tabla 29 se presentan las funciones del robot con cada una de las partes responsables de la acción y una breve descripción.

Tabla 29

Partes encargadas del funcionamiento de las acciones robot.

Acción del robot	Acrónimo	Parte o partes encargadas de la acción	Descripción
Mover adelante	MA	• Motor izquierdo. • Motor derecho. • Sensor ultrasónico.	El movimiento hacia adelante se produce gracias a las llantas y al movimiento hacia adelante de sus respectivos motores, además está limitado por el sensor ultrasónico en caso de haber algún obstáculo al frente, este podrá detectarlo y notificar al respecto.
Mover atrás	MAT	• Motor izquierdo. • Motor derecho.	El movimiento hacia atrás se produce gracias a las llantas y al movimiento hacia atrás de sus respectivos motores.
Mover izquierda	MIZ	• Motor izquierdo. • Motor derecho.	El movimiento hacia la izquierda se produce gracias a las llantas y al movimiento hacia adelante del motor derecho y al movimiento hacia atrás de motor izquierdo.
Mover derecha	MD	• Motor izquierdo. • Motor derecho.	El movimiento hacia la derecha se produce gracias a las llantas y al movimiento hacia adelante del motor izquierdo y al movimiento hacia atrás de motor derecho.
Subir brazo	SB	• Servomotor del brazo.	Para subir el brazo se utiliza el servomotor del brazo y este puede moverse en un rango de 0 a 180º en el sentido que corresponda para subir el brazo.
Bajar brazo	BB	• Servomotor del brazo.	Para bajar el brazo se utiliza el servomotor del brazo y este puede moverse en un rango de 0 a 180º en el sentido que corresponda para bajar el brazo.

Acción del robot	Acrónimo	Parte o partes encargadas de la acción	Descripción
Subir antebrazo	SAB	• Servomotor del antebrazo.	Para subir el antebrazo se utiliza el servomotor del antebrazo y este puede moverse en un rango de 0 a 180º en el sentido que corresponda para subir el antebrazo.
Bajar antebrazo	BAB	• Servomotor del antebrazo.	Para bajar el antebrazo se utiliza el servomotor del antebrazo y este puede moverse en un rango de 0 a 180º en el sentido que corresponda para bajar el antebrazo.
Mano a la izquierda	MNIZ	• Servomotor de la mano.	Para mover la mano a la izquierda se utiliza el servomotor de la mano y este puede moverse en un rango de 0 a 180º en el sentido que corresponda para mover la mano a la izquierda.
Mano a la derecha	MND	• Servomotor de la mano.	Para mover la mano a la derecha se utiliza el servomotor de la mano y este puede moverse en un rango de 0 a 180º en el sentido que corresponda para mover la mano a la derecha.
Abrir pinza	AP	• Servomotor de la pinza.	Para abrir la pinza se utiliza el servomotor de la pinza y este puede moverse en un rango de 0 a 180º en el sentido que corresponda para abrir la pinza.
Cerrar pinza	CP	• Servomotor de la pinza.	Para cerrar la pinza se utiliza el servomotor de la pinza y este puede moverse en un rango de 0 a 180º en el sentido que corresponda para cerrar la pinza.
Encender luces	EL	• Módulo de luces LED RGB.	Para encender las luces se utiliza el módulo de luces y se lo asigna en encendido.

Acción del robot	Acrónimo	Parte o partes encargadas de la acción	Descripción
Apagar luces	AL	• Módulo de luces LED RGB.	Para apagar las luces se utiliza el módulo de luces y se lo asigna en apagado.
Cambiar color luces	CCL	• Módulo de luces LED RGB.	Para cambiar el color de las luces se utiliza el módulo de luces y se asigna el color que se desee.
Transmisión de video	TV	• Cámara V2.	Para captar y transmitir el video se utiliza el módulo de la cámara de la Raspberry Pi, que adicionalmente con configuraciones de software cumple con el propósito establecido.

Software

Una vez que ya se abordó el hardware en su totalidad, y que ya se tiene más claro el funcionamiento de la parte práctica del proyecto y más específicamente del robot, es importante seguir detallando el resto de componentes que conforman la arquitectura y más específicamente el software gracias al cual se logró alcanzar el producto final.

Servidor interno del robot

Como ya se mencionó en la sección anterior, el robot dispone de una Raspberry Pi 3B+ en donde se programa todo lo relacionado con el servidor del robot, el lenguaje de programación utilizado es Python, ya qué es el lenguaje por defecto que manejan todos los dispositivos Raspberry.

Conexión remota con la Raspberry Pi. Existen dos maneras de comunicarse con la Raspberry Pi una vez ya se encuentre configurada tanto con su sistema operativo, con el Wifi encendido y con todas las prestaciones necesarias para su

funcionamiento, la primera es conectando todos los periféricos necesarios para que una computadora funcione, es decir, monitor, teclado, etc.; la segunda y la que se utiliza en el presente trabajo, es mediante el protocolo SSH y con la utilización de algún software específico para conexiones remotas, en este caso se utiliza VNC Viewer. En la figura 70 se puede ver la ventana de inicio de este software, para poder conectarse con la Raspberry Pi es necesario saber la dirección IP que se le ha asignado, para esto se puede usar algún otro programa que escanee la red. Adicionalmente cabe mencionar que como se puede observar en la figura 71, al ingresar a la conexión se nos pide usuario y contraseña, parámetros que de igual manera son establecidos por el usuario en la configuración inicial de la Raspberry.

Figura 70

Ventana inicial de VNC Viewer

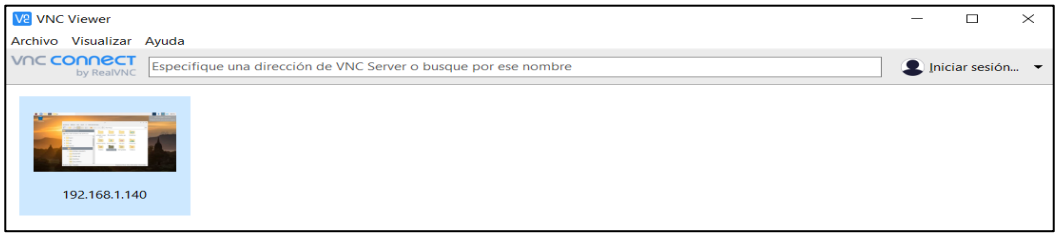
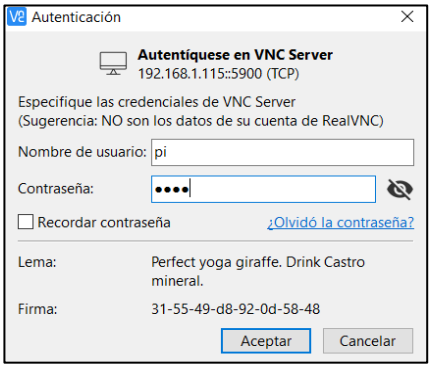


Figura 71

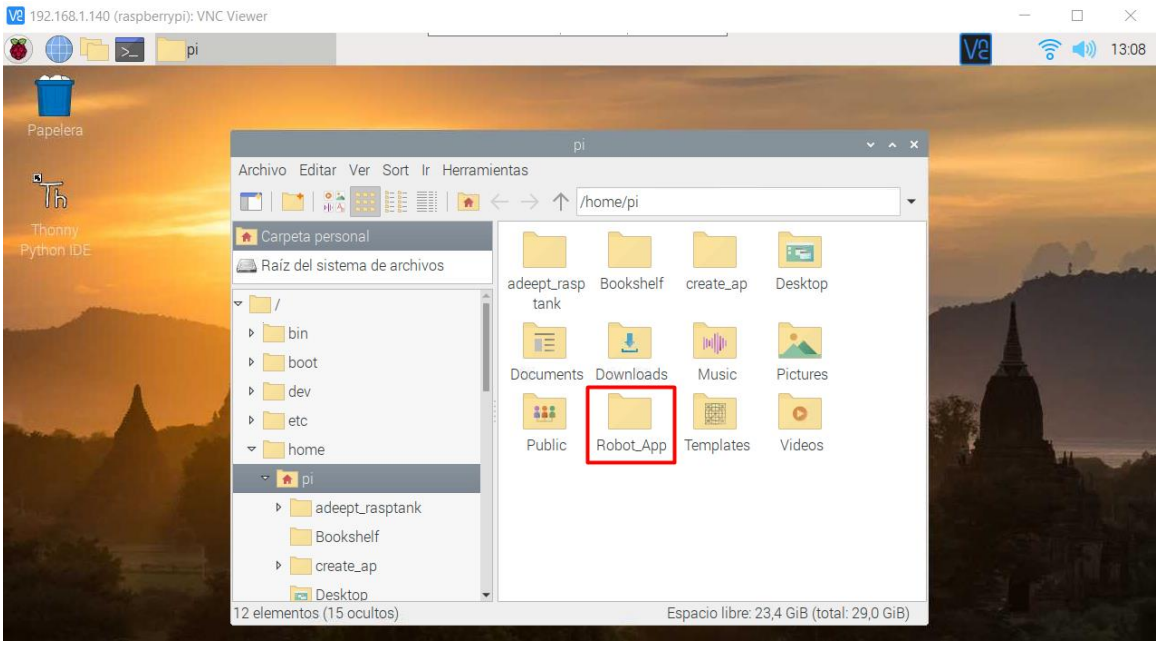
Autenticación de conexión en VNC Viewer



Estructura interna del servidor del robot. Una vez se haya ingresado a la Raspberry Pi, en primer lugar, en la carpeta principal se debe crear un paquete o carpeta donde se van a crear los archivos necesarios para el funcionamiento del robot. Como se puede ver en la figura 72, el nombre que se le dio a la carpeta es "*Robot_App*", a continuación, dentro de la misma se puede generar los archivos, estos pueden ser programados directamente desde el editor de código por defecto de la Raspberry Pi o pueden ser generados en algún editor externo en una computadora diferente y una vez finalizados, copiar y pegarlos dentro de la carpeta de la Raspberry. En la figura 73 se puede ver todos los archivos dentro de la carpeta, los cuales se utilizan para el funcionamiento del robot.

Figura 72

Carpeta principal del servidor interno del robot



Una vez presentados los archivos, es importante mencionar para que funciona cada uno y a que actuador del robot están orientados, entonces en la tabla 30 se

encuentran enlistados cada uno de los archivos y su descripción, es importante mencionar que todos tienen la extensión *".py",* ya que son escritos con Python. La relación entre ellos y su utilización se explicará posteriormente.

Figura 73

Archivos del servidor interno del robot

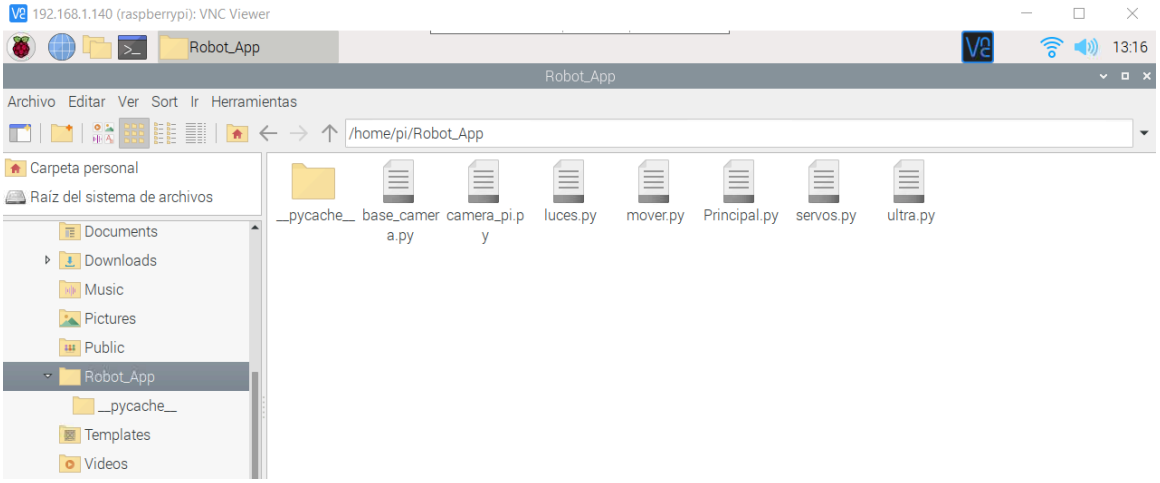


Tabla 30

Descripción de los archivos del servidor interno del robot

Nombre del archivo	Descripción
Principal	Es el archivo principal del servidor, es el encargado del funcionamiento general del robot y del servidor interno, además es el que utiliza el resto de archivos para ir generando los procesos necesarios dependiendo de la petición que se ha realizado.
Mover	Es el encargado del funcionamiento de los motores DC del robot, los cuales están enfocados en el movimiento del robot.

Nombre del archivo	Descripción
Luces	Es el encargado del funcionamiento de las luces del robot.
Servos	Es el encargado del funcionamiento de todos los servomotores del robot.
Ultra	Es el encargado del funcionamiento del sensor ultrasónico del robot, encargado de verificar si no existe ningún obstáculo frente a él.
Camera_pi Base_camera	Estos dos archivos juntos son encargados de la cámara del robot, su función es captar y trasmitir la imagen y el video.

Estructura y funcionamiento de los archivos del servidor del robot.

Es importante conocer las partes importantes de los archivos del servidor del robot para poder entender la lógica que se utilizó para que el robot logré funcionar correctamente, en el código de programación solo se abordará las partes necesarias para la explicación del funcionamiento, pero si se quiere ver el código completo, este se encuentra en los anexos.

Archivo Principal. Este archivo llamado *Principal.py* es el núcleo del servidor del robot, su código completo se encuentra en el Anexo 1.1. Al tratarse de un servidor RESTful, se debe poder realizar y receptar peticiones HTTP, para esto se utiliza el framework Flask para poder levantar y generar las rutas necesarias. Para levantar la aplicación y para que se acepten peticiones externas sin necesidad de CORS, se utiliza las siguientes líneas de código:

```
app = Flask(__name__)
CORS(app, supports_credentials=True)
```

Una vez la aplicación se encuentra inicializada, ya se puede ir generando las rutas, para esto sobre cada uno de los métodos que se quiera utilizar se debe usar la siguiente sentencia:

```
@app.route('Ruta que se desee', methods = ['Métodos HTTP'])
```

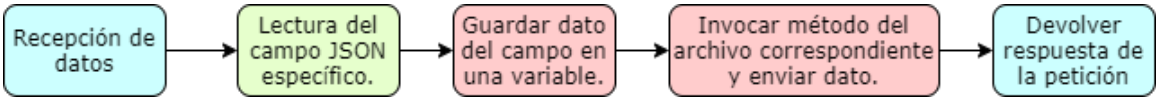
Dentro del archivo principal se definieron los métodos necesarios que vayan relacionados con cada uno de los actuadores del robot, de este modo, usando como ejemplo de actuador el servomotor del brazo, la estructura que los métodos tienen es la siguiente:

```
@app.route('/api/motorbrazo', methods = ['POST'])
def motor_brazo():
    data = request.get_json(force=True)
    movimiento = data.get('movimiento')
    respuesta = servos.servo_brazo(movimiento)
    return jsonify(movimiento=respuesta)
```

Para entender su funcionamiento se puede observar la figura 74, en el diagrama se puede ver que lo que se hace en primer lugar es recibir los datos y al tratarse de tipo JSON, se especifica cual es el campo que se quiere leer, en este caso el campo "*movimiento"*, este dato se lo guarda en una variable que se utiliza como información para invocar el o los métodos del archivo que corresponda, en este ejemplo se invoca el método "*servo_brazo"* del archivo "servos", después de que las acciones se realizan en el archivo correspondiente este devuelve una respuesta que es la que se retorna de igual manera dentro del mismo campo y en formato JSON.

Figura 74

Funcionamiento de un método del archivo principal



Es importante mencionar que hay ciertos métodos que tendrán validaciones internas diferentes que pueden depender de los datos recibidos y que se manejan mediante sentencias condicionales, pero el funcionamiento general es el mismo. Adicionalmente se tiene métodos extra que son utilizados ya sea para configuraciones iniciales o para acciones de interrupción como parar los motores de movimiento.

Ya se ha mencionado la estructura de algunos de los métodos más importantes, pero las sentencias más importantes son las que permiten que la aplicación se inicialice y qué deben ir dentro del método especial main, además al tratarse de una aplicación o servicio Web se necesita especificar la dirección IP y si se quisiera el puerto. Dicho esto, las líneas de código se ven de la siguiente manera:

```
if __name__ == '__main__':
    try:
        inicializar()
        app.run(host="0.0.0.0", threaded=True)
    except KeyboardInterrupt:
        detener()
```

Una vez se ha abordado la estructura y funcionamiento de los métodos más utilizados del archivo Principal, es importante conocer cuáles son, sus respectivas rutas y métodos HTTP, y los métodos de los archivos secundarios que utilizan, por lo que en la tabla 31 se presenta un resumen al respecto.

Tabla 31

Métodos de recepción, sus respectivas rutas HTTP y métodos secundarios

Método	Ruta HTTP	Método HTTP	Archivo secundario	Método del archivo secundario
video_feed()	/video_feed	GET	camera_pi	Camera()
motor_antebrazo()	/api/motorantebrazo	POST	servos	servo_antebrazo()
motor_brazo()	/api/motorbrazo	POST	servos	servo_brazo()
motor_camara()	/api/motorcamara	POST	servos	servo_camara()
motor_mano()	/api/motorantebrazo	POST	servos	servo_mano()
motor_pinza()	/api/motorpinza	POST	servos	servo_pinza()
motor_derecho()	/api/motorderecho	POST	mover	mover_motorderecho()
motor_izquierdo()	/api/motorizquierdo	POST	mover	mover_motorizquierdo()
luces()	/api/luces	POST	luces	colorWipe()

Archivo de los motores. Este archivo se llama *mover.py,* su código completo se encuentra en el Anexo 1.2 y está encargado del funcionamiento de los motores que permiten que el robot se mueva, dentro del archivo existen múltiples métodos que se utilizan para que todos los procesos se desarrollen de manera adecuada, pero los principales métodos son los de recepción de los datos, que tienen una estructura como la siguiente:

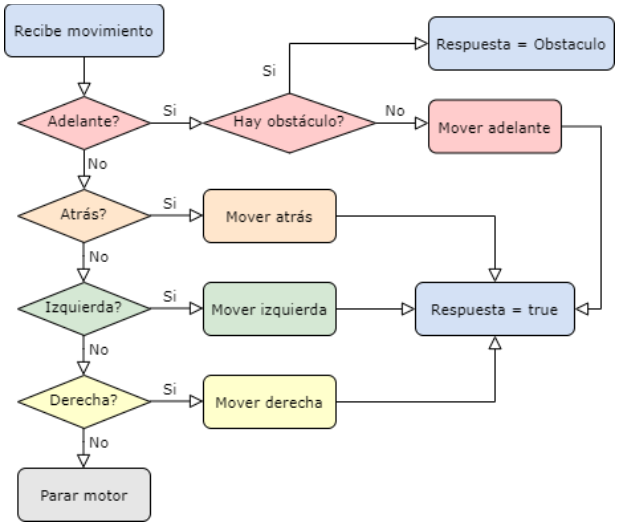
```
def mover_motorizquierdo(speed, movimiento):
    if movimiento == 'Adelante':
        distancia = ultra.checkdist()
        if distancia <= 0.3:
            respuesta = 'Obstaculo'
        else:
            motor_left(Izquierdo_adelante, speed)
```

```
            respuesta = 'true'
    elif movimiento == 'Atras':
        motor_left(Izquierdo_atras, speed)
        respuesta = 'true'
    elif movimiento == 'Izquierda':
        motor_left(Izquierdo_atras, speed)
        respuesta = 'true'
    elif movimiento == 'Derecha':
        motor_left(Izquierdo_adelante, speed)
        respuesta = 'true'
    else:
        motorStop()
    return respuesta
```

Para entender su funcionamiento se tiene el diagrama de flujo presentado en la figura 73, primero se recibe el dato de movimiento y dependiendo de cuál sea, se realiza una acción específica y devuelve la respuesta *"true*", a excepción del movimiento "Adelante" que adicionalmente necesita comprobar el valor del sensor ultrasónico y en caso de que haya un obstáculo, notifica con la respuesta *"Obstáculo".*

Figura 75

Diagrama de flujo del funcionamiento de los motores



Archivo de los servomotores. Este archivo llamado *servos.py,* su código completo se encuentra en el Anexo 1.3 y está encargado del funcionamiento de los servomotores encargados del movimiento de todas las partes móviles del robot, dispone tanto de métodos de inicialización y de configuración, pero los más importantes son los que se encargan de la rotación de los servomotores, es importante mencionar que el rango de movimiento puede ser modificado a criterio del usuario y su estructura es como la siguiente:

```
def servo_brazo(movimiento):
    global angulo_brazo
    global motor_brazo
    if movimiento == 'Subir Brazo':
        if angulo_brazo == 200:
            angulo_brazo = 300
            pwm.set_pwm(motor_brazo, 0, angulo_brazo)
            respuesta = 'true'
        elif angulo_brazo == 300:
            angulo_brazo = 400
            pwm.set_pwm(motor_brazo, 0, angulo_brazo)
            respuesta = 'true'
        elif angulo_brazo == 400:
            respuesta = 'full'
    else:
        if angulo_brazo == 200:
            respuesta = 'full'
        elif angulo_brazo == 300:
            angulo_brazo = 200
            pwm.set_pwm(motor_brazo, 0, angulo_brazo)
            respuesta = 'true'
        elif angulo_brazo == 400:
            angulo_brazo = 300
            pwm.set_pwm(motor_brazo, 0, angulo_brazo)
            respuesta = 'true'
    return respuesta
```

El funcionamiento puede observarse en la figura 74, donde se recibe el dato de movimiento y dependiendo de la variable del ángulo que se encuentra almacenada en ese momento se hace la validación de si el movimiento es posible o si ya se llegó al

límite, si el movimiento es posible, se devuelve la respuesta "*true*" y si ya está al límite, se devuelve la respuesta "*full".*

Archivo de las luces. Este archivo denominado *luces.py* es encargado del funcionamiento de las luces del robot, él código completo se encuentra en el Anexo 1.4 y su funcionamiento puede observarse en la figura 76, consiste en encender, apagar o cambiar de color las luces, dependiendo de la información que se reciba.

Figura 76

Diagrama de flujo del funcionamiento de los servomotores

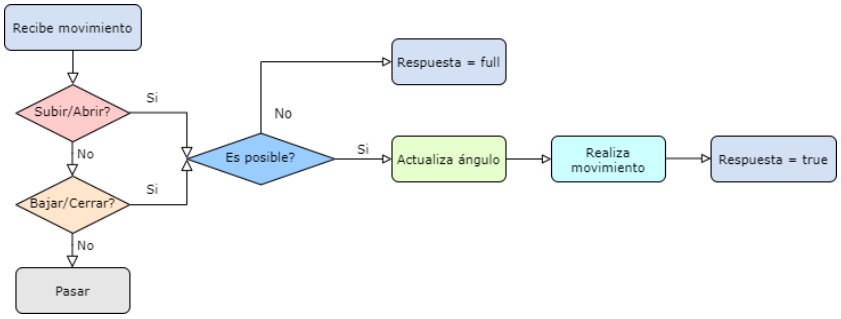
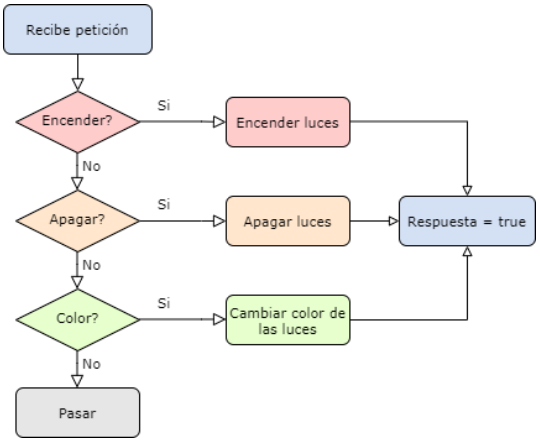


Figura 77

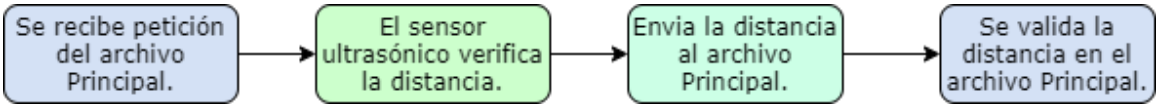
Diagrama de flujo del funcionamiento de las luces



Archivo del sensor ultrasónico. El código de este archivo denominado *ultra.py* puede ser encontrado en el Anexo 1.5 y su funcionamiento básicamente consiste en devolver el valor de la distancia de un obstáculo en caso de que existiese en la parte frontal del robot, en la figura 78 se puede ver un diagrama de flujo de su interacción con el archivo principal, en primer lugar, se recibe la petición de que se verifique la distancia, se devuelve la distancia en ese momento y esta va a sr validada en el archivo principal.

Figura 78

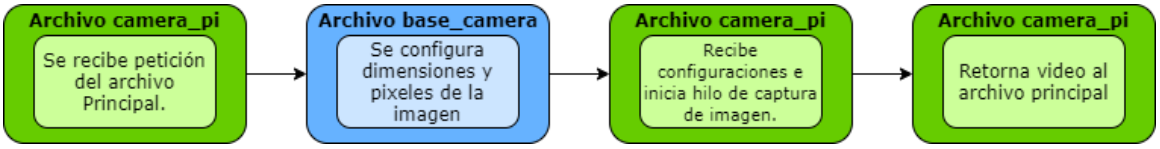
Funcionamiento del archivo del sensor ultrasónico



Archivos de la cámara. Los archivos utilizados para la cámara, llamados *camera_pi.py* y *base_camera.py* y también sus validaciones en el archivo principal, son extraídos de un proyecto Open Source proporcionado por Miguel Grinberg, su proyecto se encuentra en GitHub y su código se encuentra en el Anexo 1.6 y 1.7. El funcionamiento de los archivos está basado en Flask y debido a que en el presente proyecto se utiliza dicho framework, facilitó el trabajo al momento de realizar la captura y la transmisión de video (Grinberg, 2020). En la figura 79 se puede ver el diagrama de flujo de funcionamiento, en primer lugar, el archivo camero_pi recibe la petición de captura de imagen del archivo principal, este se comunica con el base_camera donde se configuran las dimensiones y resolución de la imagen, a continuación, envía los datos de regreso al archivo camera_pi donde se empieza el hilo que capta el video y finalmente se retorna el video al archivo principal.

Figura 79

Funcionamiento de los archivos de la cámara



Despliegue del servidor del robot. Con todos los archivos correctamente configurados y programados, el último paso para poder levantar o desplegar el servidor del robot, se lo realiza desde el terminal de la Raspberry Pi, en la figura 80 se puede visualizar el proceso, en primer lugar, se debe acceder a la carpeta del servidor con el comando:

```
cd Robot_App
```

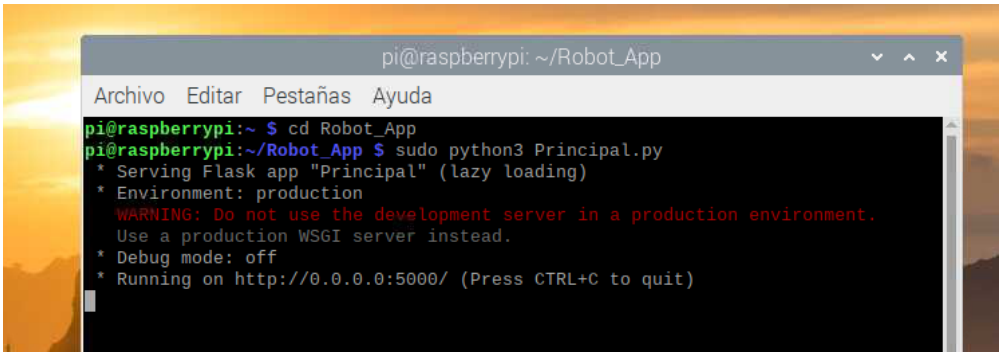
Posteriormente se ejecuta como administrador el archivo principal, con el siguiente comando:

```
Sudo python3 Principal.py
```

Finalmente se puede ver que la aplicación Web se encuentra inicializada y corriendo en la dirección IP y el puerto correspondiente que fueron asignados mediante código en la configuración interna del archivo principal.

Figura 80

Despliegue del servidor del robot



Frontend – Aplicación de Control

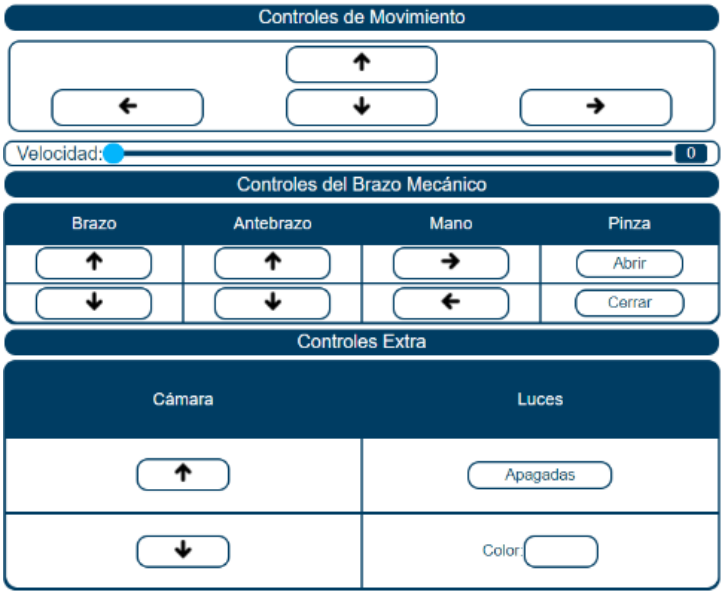
La implementación de la aplicación de control está dividida en dos partes principales, la parte estética y la parte funcional. Para la parte estética se utilizó HTML para generar la estructura con cada una de las divisiones y con todos los elementos necesarios para poder cumplir los fines prácticos del proyecto. Para dar formato, color y animaciones se utilizó CSS, que es encargado de darle todos los acabados y detalles estéticos a la aplicación generada en HTML. El código de la parte estética tanto de HTML y CSS, no se abordarán a profundidad ya que, el diseño estético de la aplicación no es un parte fundamental de los objetivos de esta tesis, sin embargo, el código de ambos archivos puede ser encontrado en los Anexos 2.1 y 2.2 respectivamente.

Por otro lado, lo que si se aborda y es muy importante dentro de los objetivos del presente proyecto, es la parte de funcionalidad y de configuraciones, para esto se utilizó JavaScript, este lenguaje de programación ayuda a darle prestaciones extra a cada uno de los componentes de la aplicación generados por HTML y CSS, entre las características que se les puede otorgar a los elementos de la aplicación, la más importante y la que permite la comunicación con el Backend y el servidor del robot, es el poder generar peticiones HTTP, y consumir todos los servicios REST de la arquitectura. De igual manera el código de JavaScript se encuentra en el Anexo 2.3.

Configuraciones de la aplicación de control. Como ya se mencionó, solo se abordará el código utilizado en JavaScript, pero en primer lugar es importante recordar la estructura general que tiene la aplicación de control, pero sobre todo la de los botones utilizados para generar las peticiones necesarias para enviar al Backend, en la figura 81 se puede ver todos los controles que tiene la aplicación, además hay que tener en cuenta que se tiene el componente de video, que hace su petición internamente.

Figura 81

Controles de la aplicación de control



Ya con los controles en mente, se puede abordar el código. Todos los botones siguen la misma configuración para poder generar las peticiones HTTP, el código tiene la estructura siguiente:

```
let url = "http://localhost:8090/api";
var xmlhttp = new XMLHttpRequest();
var xmlhttp_aux = new XMLHttpRequest();

function mover_atras(){
  document.getElementById('boton_atras').style.backgroundColor = "#abd9f5";
      /*
      Configuraciones y variables que se desee.
      */
      xmlhttp.onerror = function () {
              /*
             Configuraciones de alerta de error, en caso de no establecer conexión con
             el servidor
             */
        }
      xmlhttp.open("POST", url +"/moveratras/enviar_izquierdo");
      xmlhttp_aux.open("POST", url +"moveratras/enviar_derecho");
      xmlhttp.setRequestHeader("Content-Type",
            "application/json;charset=UTF-8");
      xmlhttp_aux.setRequestHeader("Content-Type",
            "application/json;charset=UTF-8");
      xmlhttp.send(JSON.stringify({ "movimiento": "Atras" + velocidad,
      "fecha": fecha + hora }));
      xmlhttp_aux.send(JSON.stringify({ "movimiento": "Atras" + velocidad,
      "fecha": fecha + hora }));
```

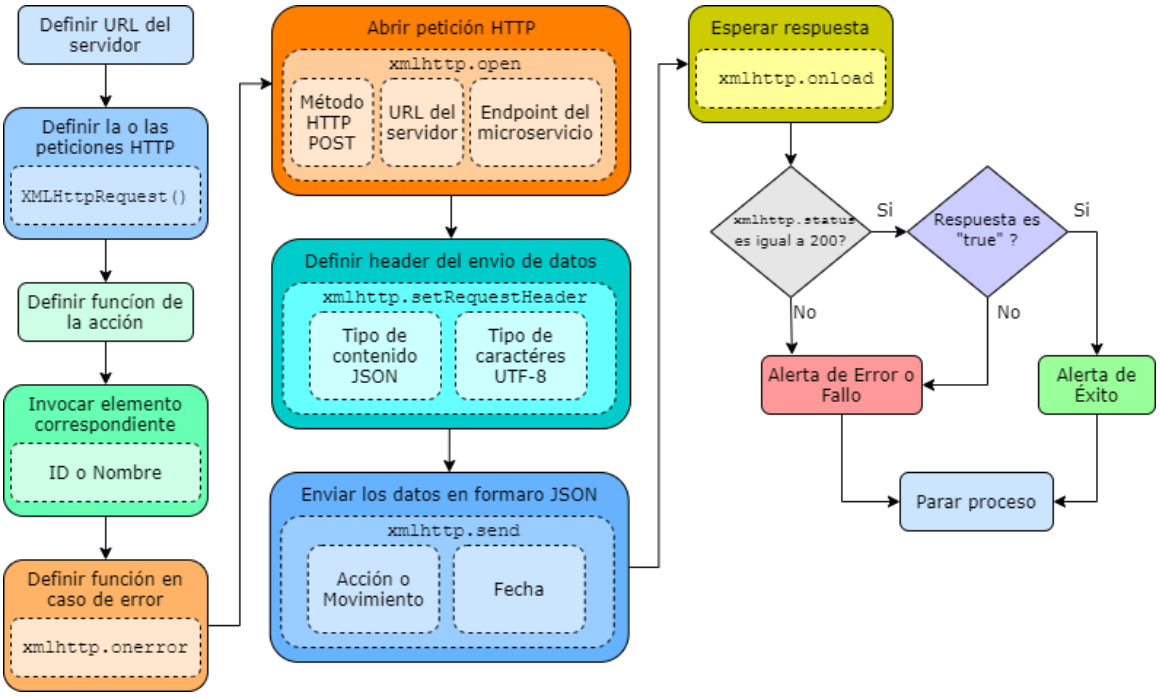
```
    xmlhttp.onload = function () {
        if (xmlhttp.status == 200) {
            respuesta = JSON.parse(this.responseText);
            movimiento = respuesta.movimiento;
        }
        else {
          /*
          Configuración en caso de fallo.
          */
        }
    }
    xmlhttp_aux.onload = function () {
        if (xmlhttp_aux.status == 200) {
            respuesta_aux = JSON.parse(this.responseText);
            movimiento_aux = respuesta_aux.movimiento;
        }
        else {
          /*
          Configuración en caso de fallo.
          */
        }
    }
    setTimeout(() => {
        if (xmlhttp.status == 200 || xmlhttp_aux.status == 200) {
            if (fallo == "true" || fallo_aux == "true") {
                /*
                Alerta en caso de fallo.
                */
            }
            else {
                if (movimiento == "true" && movimiento_aux == "true") {
                        /*
                        Alerta en caso de éxito.
                        */
                }
                else {
                        /*
                        Alerta en caso de fallo
                        */
                }
            }
        };
    }, 500);
    parar_atras();
}
```

Su funcionamiento en primer lugar consiste en definir cuál es la URL del servidor o el Gateway al que nos queremos conectar, a continuación se crea las peticiones HTTP con la clase XMLHttpRequest(),después se crea la función, en este caso para el ejemplo se utiliza el botón de mover hacia atrás (MAT), dentro de la función se llama el botón por el ID o nombre que se definió en el archivo HTML, a continuación con xmlhttp.open() se abre la petición definiendo el método, la url y el endpoint específico del servicio, con xmlhttp.setRequestHeader()

se define el tipo de contenido JSON y el formato de los caractéres UTF-8, finalmente con xmlhttp.send() se hace el envío de los datos de movimiento y fecha y se queda a la espera de una respuesta que se validá con xmlhttp.onload() y depende del xmlhttp.status que debe ser igual a 200 para ser correcto y generar la alerta de éxito, caso contrario se generarían alertas de error o fallo. Es importante mencionar que en este ejemplo se realiza acciones dobles para todo, debido a que en el caso del MAT se debe comunicar con los motores izquierdo y derecho. Para una mayor visión del funcionamiento, en la figura 82 se puede observar un diagrama de flujo de todo el proceso de la petición.

Figura 82

Diagrama de flujo de las peticiones HTTP de la aplicación de control



Finalmente, una vez explicado el funcionamiento de cómo es que funcionan las peticiones HTTP del frontend, es importante saber cuáles son las rutas, los endpoints específicos de los microservicios para cada uno de los controles de la aplicación, y el

movimiento que se envía en formato JSON, de este modo en la tabla 32 se presentan todos estos datos.

Tabla 32

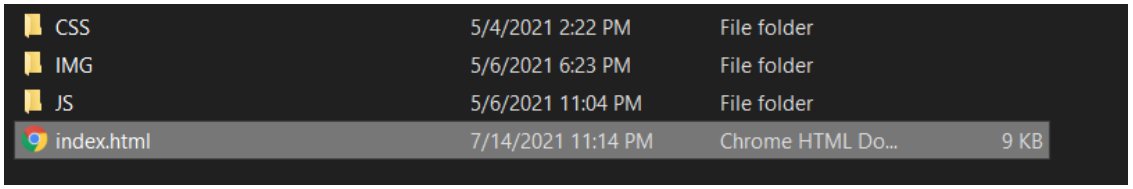
Controles de la aplicación de control y sus atributos

Control de movimiento	Ruta HTTP	Endpoint del microservicio	Mensaje enviado en JSON
Movimiento			
Adelante ↑	http://localhost:8090/api	/moveradelante/enviar_izquierdo /moveradelante/enviar_derecho	Adelante
Atrás ↑	http://localhost:8090/api	/moveratras/enviar_izquierdo /moveratras/enviar_derecho	Atras
Izquierda ←	http://localhost:8090/api	/moverizquierda/enviar_izquierdo /moverderecha/enviar_derecho	Izquierda
Derecha →	http://localhost:8090/api	/moveratras/enviar_izquierdo /moveratras/enviar_derecho	Derecha
Brazo			
Subir ↑	http://localhost:8090/api	/subirbrazo/enviar	Subir Brazo
Bajar ↓	http://localhost:8090/api	/bajarbrazo/enviar	Bajar Brazo
Antebrazo			
Subir ↑	http://localhost:8090/api	/subirantebrazo/enviar	Subir Antebrazo
Bajar ↓	http://localhost:8090/api	/bajarantebrazo/enviar	Bajar Antebrazo
Mano			
Izquierda ←	http://localhost:8090/api	/manoizquierda/enviar	Mano a la Izquierda
Derecha →	http://localhost:8090/api	/manoderecha/enviar	Mano a la Derecha
Pinza			
Abrir	http://localhost:8090/api	/abrirpinza/enviar	Abrir Pinza
Cerrar	http://localhost:8090/api	/cerrarpinza/enviar	Cerrar Pinza
Cámara			
Subir ↑	http://localhost:8090/api	/subircamara/enviar	Subir Cámara
Bajar ↓	http://localhost:8090/api	/bajarcamara/enviar	Bajar Cámara
Luces			
Encendidas	http://localhost:8090/api	/encenderluces/enviar	Encender
Apagadas	http://localhost:8090/api	/apagarluces/enviar	Apagar
Color	http://localhost:8090/api	/cambiarcolorluces/enviar	Código hexadecimal del color

Finalmente, para utilizar la aplicación de control, se puede hacerlo con cualquier navegador si se encuentra en un servidor local, esto se hace abriendo el archivo index.htm como se puede ver en la figura 83, en caso de que ya se encuentre desplegado en algún host, se escribiría la dirección web en el navegador.

Figura 83

Archivo principal de la aplicación de control



Backend – Arquitectura del servidor

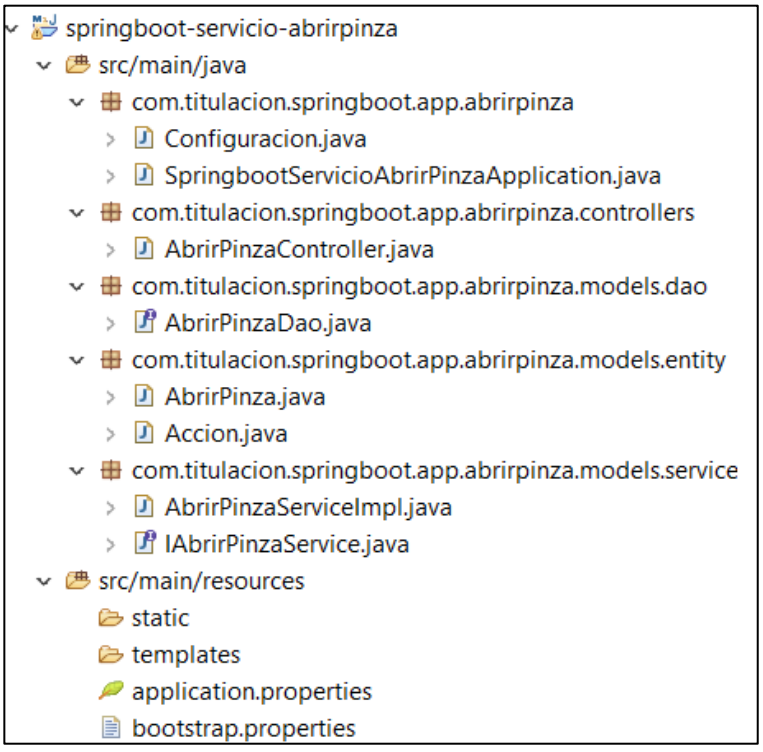
El Backend es el encargado de todo él envió, recepción y gestión de todos los datos, como ya se mencionó en capítulos anteriores, la arquitectura implementada es orientada a servicios y tolerante a fallos. Dentro de los frameworks de desarrollo más utilizados, se escogió Spring Tools 4 por su gran versatilidad al momento de crear aplicaciones o servicios Web, esta herramienta es desarrollada por Eclipse y trabaja con el lenguaje de programación Java junto anotaciones de tipo Spring, que brindan información extra de todos los programas y archivos necesarios para poder implementar un microservicio y configurarlo para que cumpla con las características REST.

Estructura de los microservicios funcionales. Como ya se mencionó, hay dos tipos de microservicios funcionales, los conceptuales que están orientados a un concepto o a una acción del robot como tal y los actuadores, que son los que reciben esta información y la envían al servidor del robot con endpoints específicos de cada uno de sus actuadores.

Estructura de los microservicios conceptuales. Los microservicios conceptuales están distribuidos de la siguiente manera: en el paquete principal hay 3 subpaquetes, la aplicación (nombre del microservicio), los controladores (controllers), y los modelos (models), este último se divide en otros 3 subpaquetes, dao (objeto de acceso a datos), entidad (entity), y servicio (service), dentro de cada uno de estos paquetes hay diferentes clases importantes para el funcionamiento del microservicio. Adicionalmente existen dos archivos o clases de configuración importantes en la carpeta recursos (resources) del microservicio. Para poder visualizar de una mejor manera la estructura de los microservicios conceptuales, en la figura 84 se utiliza de ejemplo al servicio *"abrir_pinza".*

Figura 84 .

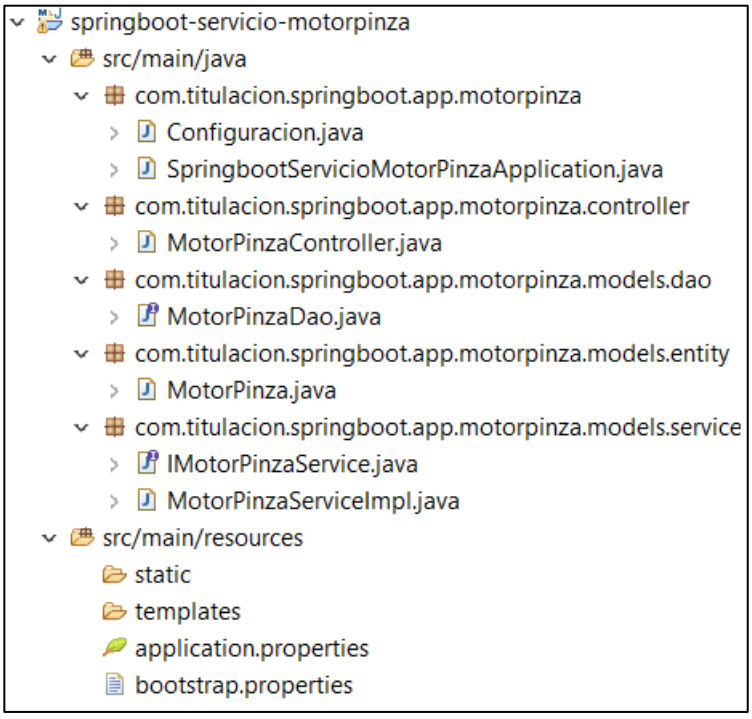
Estructura de los microservicios conceptuales.



Estructura de los microservicios actuadores. La estructura de los microservicios actuadores es muy similar a la de los microservicios conceptuales, con la única diferencia de que en el paquete de entidades no existe la clase *Acción.java,* esto debido a que las acciones solo se reciben en el microservicio conceptual, se envía al actuador y finalmente se reenvía al servidor del robot. Para poder visualizar de una mejor manera la estructura de los microservicios actuadores, en la figura 85 se utiliza de ejemplo al servicio *"motor_pinza",* para utilizar dos microservicios qué se relacionan entre sí.

Figura 85

Estructura de los microservicios actuadores



Configuración de las clases de los microservicios funcionales.

El funcionamiento de los microservicios conceptuales y actuadores es el mismo en cuanto al proceso que siguen, su diferencia es que los conceptuales envían los datos a los actuadores y los actuadores envían los datos al servidor del robot. Antes de

explicar el proceso como tal, es importante abordar cada uno de las clases observables en las figuras 84 y 85, los mismos que conforman el microservicio y tienen configuraciones importantes que deben ser explicadas. Para una mayor comprensión se seguirá utilizando como ejemplo al microservicio "*Abrir_pinza".*

Clases del paquete de la aplicación. En primer lugar, se tiene la clase de la aplicación llamada *SpringbootServicioAbrirPinzaApplication,* su código completo se encuentra en el Anexo 3.1.2, sus funciones principales están definidas por las siguientes anotaciones Spring:

- **@EnableCircuitBreaker:** Al tratarse de una arquitectura tolerante a fallos, permite que se active esta funcionalidad que tiene como fin "abrir" un circuito en caso de un error y redireccionar el proceso a un método alternativo.
- **@EnableEurekaClient:** Al disponer de un servidor de nombre, en este caso Eureka, esta anotación permite definir a esta aplicación o microservicio como un cliente de Eureka, lo que quiere decir que se registrará y localizará en dicho servidor.
- **@SpringBootApplication:** Esta anotación es quizá la más importante, ya que habilita al microservicio como una aplicación de Spring, esto permite qué se pueda usar todas las anotaciones junto con la programación de Java.

En segundo lugar, se tiene la clase llamada *Configuracion,* su código completo se encuentra en el Anexo 3.1.1, sus funciones principales están definidas por las siguientes anotaciones Spring:

- **@Configuration:** Esta anotación sirve como un indicador para que la aplicación como tal sepa que dentro de esta clase se encuentran

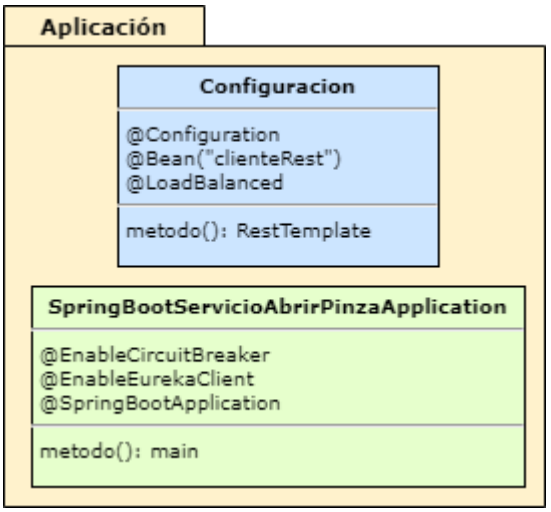
configuraciones tipo @Bean, las cuales son importantes para generar y gestionar peticiones especiales.

- **@Bean:** Esta anotación está orientada para que se puedan reutilizar los métodos o las clases a las que apunta, ya que todo se alberga dentro de un contenedor Spring, también define un nombre dentro del Bean de dicha clase o método, en este caso @Bean(“clienteRest”), ya que se trataba del método RestTemplate para hacer qué el servicio sea REST.
- **@LoadBalanced:** Es una anotación que configura el balanceo de cargas si fuese necesario, pero únicamente es posible si existe más de una instancia del servicio activa.

Con las clases del paquete de la aplicación y todas sus anotaciones Spring importantes bien definidas, se puede ver la estructura de dicho paquete en la figura 86.

Figura 86

Estructura del paquete de la aplicación de un microservicio funcional



Clases del paquete del controlador. Dentro del paquete del controlador se tiene solo una clase llamada *AbrirPinzaController,* su código completo se encuentra en el Anexo 3.2.1, su propósito principalmente es el de manejar todas las peticiones HTTP tanto de envió como de recepción, además y muy importante, es qué maneja la tolerancia a fallos con métodos alternativos en caso de que el patrón de *Circuit Breaker* se llegase a activar. Las principales anotaciones Spring que definen a esta clase son las siguientes:

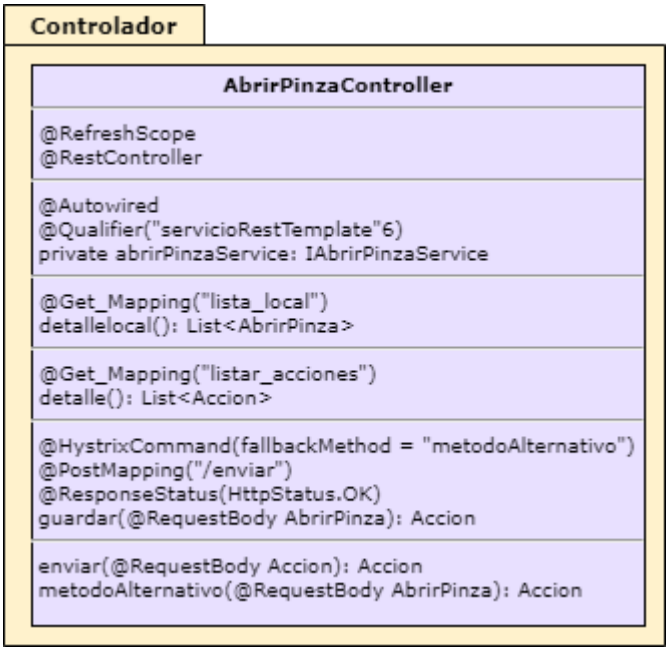
- **@RefreshScope:** Funciona para indicarle a la clase que existe una clase de configuración en alguna parte del proyecto, entonces se le notifica que debe acceder al método o métodos anotados con @Bean que sean necesarios.
- **@RestController:** Habilita a esta clase como el controlador REST, para que pueda utilizar todas las prestaciones RESTful que Spring y Java ofrecen, además permite que se usen todas las anotaciones que existen para poder hacer peticiones.
- **@Autowired:** Indica a la aplicación que el campo que se encuentra anotado debe ser inyectado con dependencias Spring distintas a las de la clase.
- **@Qualifier:** Sirve para diferencias la jerarquía de los métodos que se quiere utilizar, en caso de que se trate del mismo tipo o sean similares.
- **@PostMapping:** Es una anotación orientada a las peticiones HTTP tipo POST, define los endpoints de la clase donde se quieren recibir los datos.
- **@GetMapping:** Es una anotación orientada a las peticiones HTTP tipo GET, define los endpoints de la clase de donde se quieren obtener los datos.
- **@ResponseStatus:** Esta anotación está orientada a la respuesta de la petición HTTP que se haya realizado en ese momento, y funciona para enviar el código de respuesta HTTP que sea necesario.

- **@HystrixCommand:** Esta anotación es la que habilita la tolerancia a fallos dentro del microservicio, ya que dentro de ella permite definir una ruta o un método alternativo en caso de que ocurra un error.
- **@RequestBody:** Esta anotación funciona para solicitar que se extraiga el cuerpo o “body” del mensaje, esto con la finalidad de procesar los datos con cada uno de los campos JSON que se hayan definido.

Una vez definidas cada una de las anotaciones de la clase, en la figura 87 se puede observar la estructura final del paquete y la clase con cada uno de sus métodos importantes.

Figura 87

Estructura del paquete del controlador de un microservicio funcional

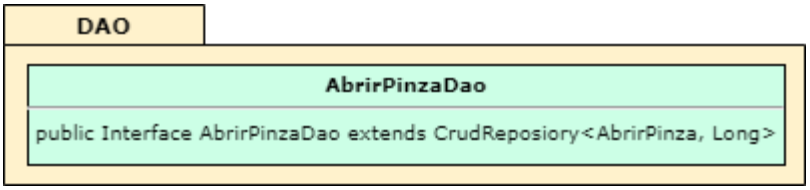


Clases del paquete DAO. La única clase que se encuentra dentro de este paquete, es la llamada *AbrirPinzaDao,* su código s encuentra en el Anexo 3.3.1, esta clase no tiene una estructura compleja, y tampoco tiene anotaciones Spring, solo se

utiliza para implementar los métodos CRUD, en la figura 88 se puede observar la estructura del paquete.

Figura 88

Estructura del paquete DAO de un microservicio funcional



Clases del paquete de entidades. Los archivos de este paquete tienen la tarea de definir las variables de los objetos de datos que se va a recibir y a enviar, en este caso al ser una arquitectura RESTful se utilizará objetos de datos tipo JSON. En primer lugar, se tiene la clase *AbrirPinza,* su código se encuentra en el Anexo 3.4.1 y representa la entidad propia del microservicio y está configurada con las siguientes anotaciones Spring:

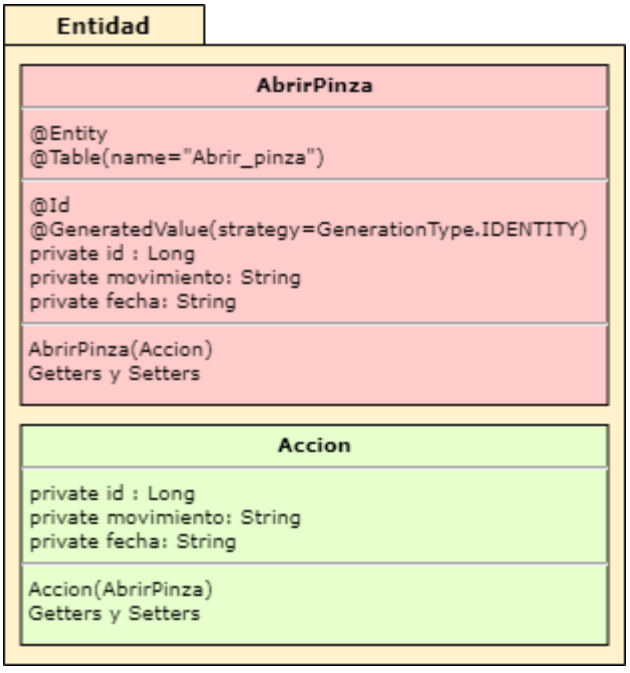
- **@Entity:** Esta anotación es la encargada de indicar a los procesos del microservicio que esta clase es la entidad principal y que sus datos están estructurados según se definan aquí.
- **@Table:** Esta anotación es la encargada de asignar el nombre de la tabla de la base de datos, donde se van a almacenar la información necesaria.
- **@Id:** Es para indicar que cada uno de los datos recibidos van a tener un Id específico que los represente.
- **@GeneratedValue:** Funciona junto con la anotación Id y sirve para ir generado los id de manera ascendente según se vayan recibiendo y guardando los datos.

En segundo lugar, se tiene a la clase *Acción* su código se encuentra en el Anexo 3.4.2*,* y es utilizada para manejar los datos que se van a enviar, en este caso sirve también para diferenciar los datos que se guardan y los qué se envían, esta clase no tiene configuraciones con anotaciones Spring.

Con las configuraciones necesarias explicadas, en la figura 89 se puede observar la estructura del paquete de las entidades con sus respectivas clases, anotaciones Spring y métodos.

Figura 89

Estructura del paquete de entidades de un microservicio funcional



Clases del paquete del servicio. Las clases de este paquete están destinadas a definir los métodos principales del microservicio, tanto para envió, recepción y para almacenar los datos. En primer lugar, se tiene la clase *IAbrirPinzaService,* su código se encuentra en el Anexo 3.5.1 y es una interfaz en la que se especifica los métodos que el servicio va a utilizar con sus respectivos tipos de datos. En segundo lugar, tenemos

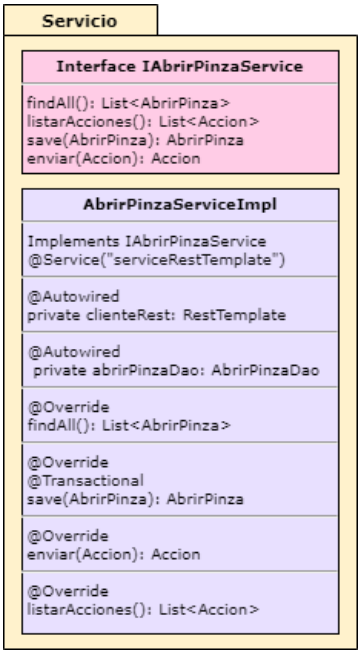
AbrirPinzaServiceImpl, su código se encuentra en el Anexo 3.5.2 y es la clase que implementa los métodos de la interfaz, además está configurada con algunas anotaciones Spring importantes:

- **@Service:** Esta anotación indica al proceso que la clase en donde se encuentra es un servicio implementado.
- **@Autowired:** Esta anotación permite que los componentes de la clase sean inyectados con dependencias Spring externas.
- **@Transactional:** Esta anotación permita que se pueda manejar y compartir un atributo específico entre clases y métodos

Después de haber definido las clases, métodos y anotaciones importantes, la estructura total del paquete del servicio puede ser observada en la figura 90.

Figura 90

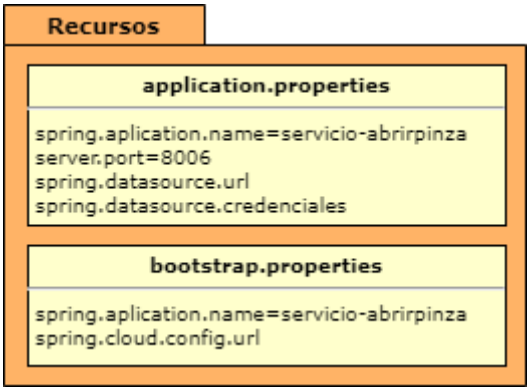
Estructura del paquete del servicio de un microservicio funcional



Archivos del paquete de recursos. Los archivos en este paquete son encargados de las configuraciones del microservicio, la estructura del paquete se observa en la figura 91, el primer archivo es el *application.properties,* su código se encuentra en el Anexo 3.6.1, en el constan el nombre de la aplicación, el puerto, la ruta del servidor Eureka, y la ruta con las credenciales de la base de datos. El segundo archivo es el *bootstrap.properties,* este es un archivo opcional, ya que en el se definen el nombre del archivo de configuración y la ruta del servidor de configuración.

Figura 91

Estructura del paquete de recursos de un microservicio funcional

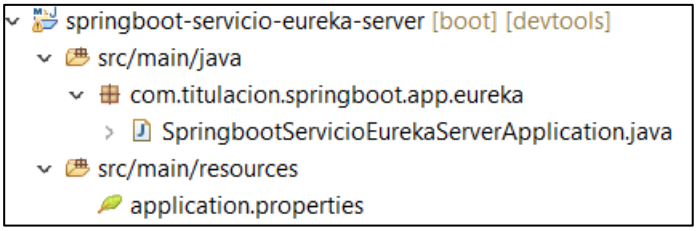


Estructura de los microservicios de configuración. Además de los microservicios funcionales que realizan las acciones necesarias del manejo de la información del sistema, dentro de la arquitectura también existen microservicios de configuración que son muy importantes para que se pueda cumplir con los objetivos propuestos. Estos microservicios son: el servidor de nombre Eureka, el servidor proxy Zuul y el servidor de configuración, a continuación, se detallará su arquitectura.

Estructura del microservicio del servidor Eureka. La estructura de este microservicio, es bastante simple, solo consta del paquete de aplicación y del paquete de los recursos, en la figura 92 se observa esta estructura.

Figura 92

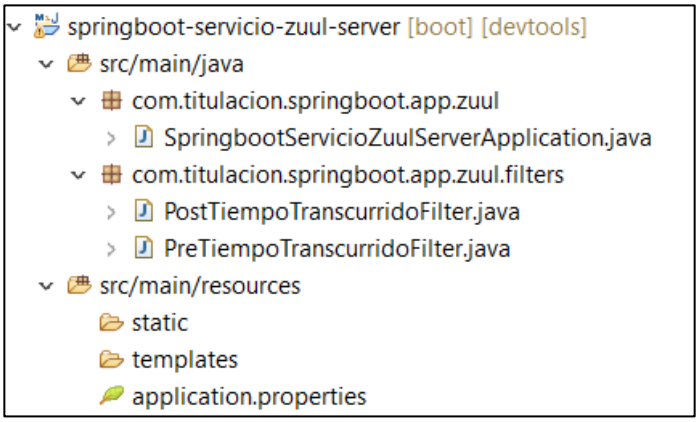
Estructura del microservicio del servidor Eureka



Estructura del microservicio del servidor Proxy Zuul. Este microservicio al igual que el servidor Eureka, consta del paquete de aplicación y el paquete de recursos, pero adicionalmente se agrega el paquete de filtros, orientado a calcular el tiempo de enrutamiento entre las peticiones de los microservicios. En la figura 93 se observa su estructura.

Figura 93

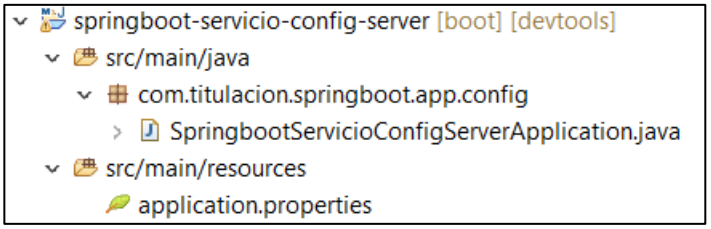
Estructura del microservicio del servidor proxy Zuul



Estructura del microservicio del servidor de configuración. Este microservicio es opcional, pero su estructura es muy similar a los otros dos microservicios de configuración, consta del paquete de la aplicación y del paquete de recursos, en la figura 94 se puede ver la estructura de este microservicio.

Figura 94

Estructura del microservicio del servidor de configuración



Estructura de las clases de los microservicios de configuración.

Una vez conocida la estructura de los microservicios de la configuración, es importante abordar el código de cada una de las clases que constan en cada uno de sus paquetes,

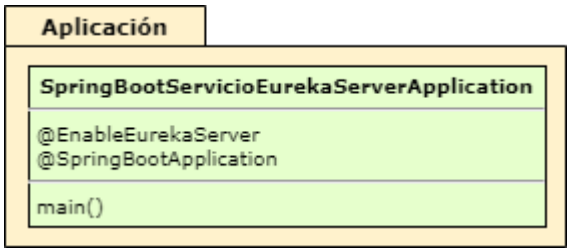
Clases del paquete de aplicación del servidor Eureka. La única clase que se encuentra es *SpringbootServiceEurekaServerApplication,* su código se encuentra en el Anexo 3.7.1 y es encargada de definir al microservicio como el servidor de Eureka y está definido por las siguientes anotaciones Spring:

- **@EnableEurekaServer:** Es la anotación que habilita al microservicio como servidor de Eureka, qué es el encargado de localizar y registrar el resto de los microservicios.
- **@SpringBootApplication:** Esta anotación habilita al microservicio como una aplicación de Spring.

Conociendo las anotaciones importantes, la estructura de este paquete se puede observar en la figura 95.

Figura 95

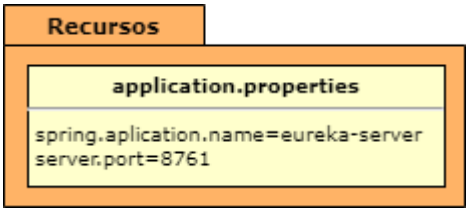
Estructura del paquete de aplicación del servidor Eureka



Archivos del paquete de recursos del servidor Eureka. Este paquete está enfocado en configurar el microservicio del servidor Eureka, la estructura de su único archivo llamado *application.properties,* se puede observar en la figura 96. El puerto por defecto de Eureka es el 8761 y el código completo se encuentra en el Anexo 3.8.1

Figura 96

Estructura del paquete de recursos del servidor Eureka



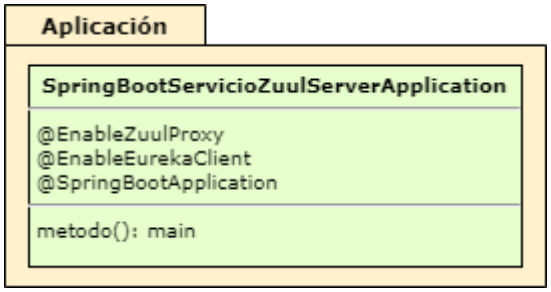
Clases del paquete de aplicación del servidor proxy Zuul. El servidor proxy Zuul es el encargado de redireccionar las peticiones que lleguen al Backend y también de los microservicios entre sí, también funciona como Gateway del backend y orquestador de todos los procesos, también es capaz de calcular el tiempo de cada petición y de seguir su enrutamiento. La única clase que se tiene en este paquete se llama *SpringbootServicioZuulServerApplication su código se encuentra en el anexo 3.11.1* y está definida con las siguientes anotaciones Spring:

- **@EnableEurekaClient:** Es la anotación que habilita al microservicio como cliente de Eureka, para poder registrar el microservicio.
- **@EnableZuulProxy:** Es la anotación que habilita al microservicio como servidor Zuul, un Proxy, orquestador y Gateway del Backend.
- **@SpringBootApplication:** Esta anotación habilita al microservicio como una aplicación de Spring.

Después de abordar la estructura y las anotaciones importantes, la estructura de este paquete puede verse en la figura 97.

Figura 97

Estructura del paquete de aplicación del servidor proxy Zuul



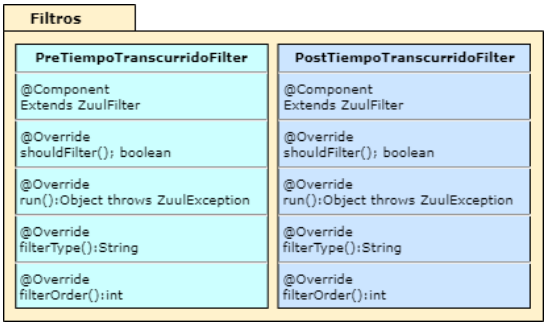
Clases del paquete de filtros del servidor proxy Zuul. Los filtros del servidor Zuul tienen como funcionalidad el calcular el tiempo de las peticiones que llegan al Backend y que se enrutan entre los microservicios internos, además indica la ruta que siguen los datos de un microservicio a otro. La primera clase que se tiene se llama *PreTiempoTranscurridoFilter,* y está encargado de determinar el tiempo inicial de la petición, y de indicar el enrutamiento del origen al destino. La segunda clase es el *PostTiempoTranscurridoFilter,* y está encargado de determinar el tiempo final y con ello calcular el tiempo transcurrido que le tomo a la petición en completar su objetivo. El código de ambas clases se encuentra en los Anexos 3.12.1 y 3.12.2. La única anotación que consta en estas clases, es la siguiente:

- **@Component:** Es la anotación que habilita a la clase como un componente, para que pueda funcionar dentro de un proceso o un camino específico de la clase, en este caso primero Pre, luego Post.

Una vez qué definidas sus características importantes, en la figura 98 se puede ver la estructura de este paquete.

Figura 98

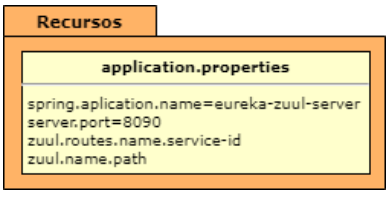
Estructura del paquete de filtros del servidor proxy Zuul



Archivos del paquete de recursos del servidor proxy Zuul. Este paquete está enfocado en configurar el microservicio del servidor proxy Zuul, la estructura de su único archivo llamado *application.properties,* se puede observar en la figura 99. El puerto por defecto de Zuul es el 8090, además en sus propiedades debe contar con los nombres y las rutas del resto de los microservicios que serán utilizados en la arquitectura del sistema. El código completo se encuentra en el Anexo 3.13.1.

Figura 99

Estructura del paquete de recursos del servidor proxy Zuul



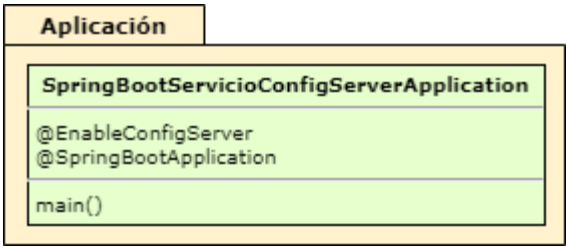
Clases del paquete de aplicación del servidor de configuración. La única clase que se encuentra es *SpringbootServicioConfigServerApplication,* su código se encuentra en el Anexo 3.9.1 y es encargada de definir al microservicio como el servidor de configuración y está definido por las siguientes anotaciones Spring:

- **@EnableConfigServer:** Es la anotación que habilita al microservicio como servidor de configuración, qué es el encargado de definir ciertas propiedades como los puertos y otros atributos de los microservicios.
- **@SpringBootApplication:** Esta anotación habilita al microservicio como una aplicación de Spring.

Una vez qué definidas sus características importantes, en la figura 100 se puede ver la estructura de este paquete.

Figura 100

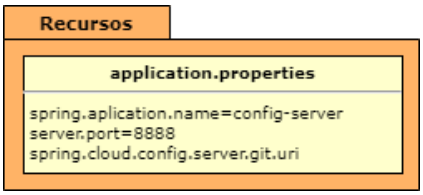
Estructura del paquete de aplicación del servidor de configuración



Archivos del paquete de recursos del servidor de configuración. Este paquete está enfocado en configurar el microservicio del servidor de configuración, la estructura de su único archivo llamado *application.properties,* se puede observar en la figura 101. El puerto por defecto de Zuul es el 8888, además en sus propiedades debe contar con la ruta del repositorio Git donde se albergan los archivos de configuración de los microservicios. El código completo se encuentra en el Anexo 3.10.1.

Figura 101

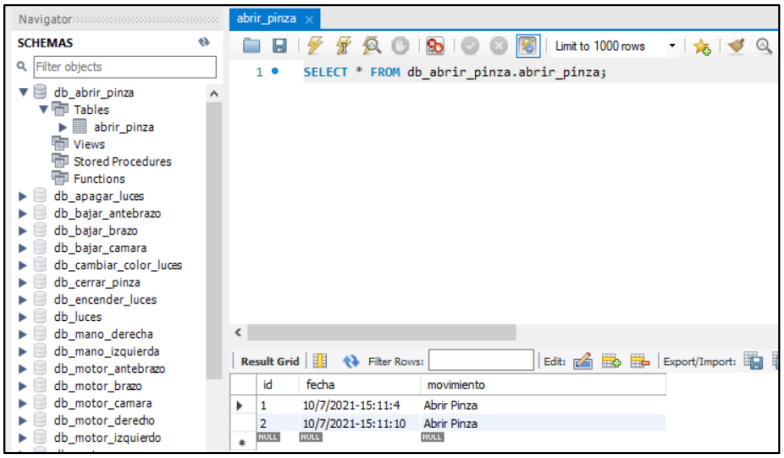
Estructura del paquete de recursos del servidor de configuración



Estructura de las bases de datos de los microservicios. Las bases de datos utilizadas para los microservicios son de tipo relacional y funcionan bajo el lenguaje de programación SQL, la herramienta que se utilizó para el diseño es MySQL Workbench, ya que toda la configuración puede realizarse de manera visual. La estructura final de las bases de datos se encuentra en la figura 102, en primer lugar, a la izquierda se puede ver las bases de datos de algunos microservicios y se encuentra desplegada la del ejemplo "*Abrir_pinza",* dentro de su esquema principal se encuentra la tabla donde se almacenan los datos, denominada de igual manera *"abrir_pinza",* en la parte derecha se tiene la sentencia SQL utilizada para observar los valores dentro de la tabla, utilizada para desplegar los datos de id, fecha y movimiento.

Figura 102

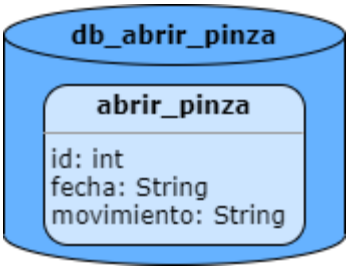
Estructura de las bases de datos de los microservicios.



Con fines prácticos en la figura 103 se muestra la estructura del esquema de una base de datos del microservicio "*Abrir_pinza"*, su nombre principal, nombre de la tabla y los campos de la tabla.

Figura 103

Esquema de una base de datos de los microservicios.

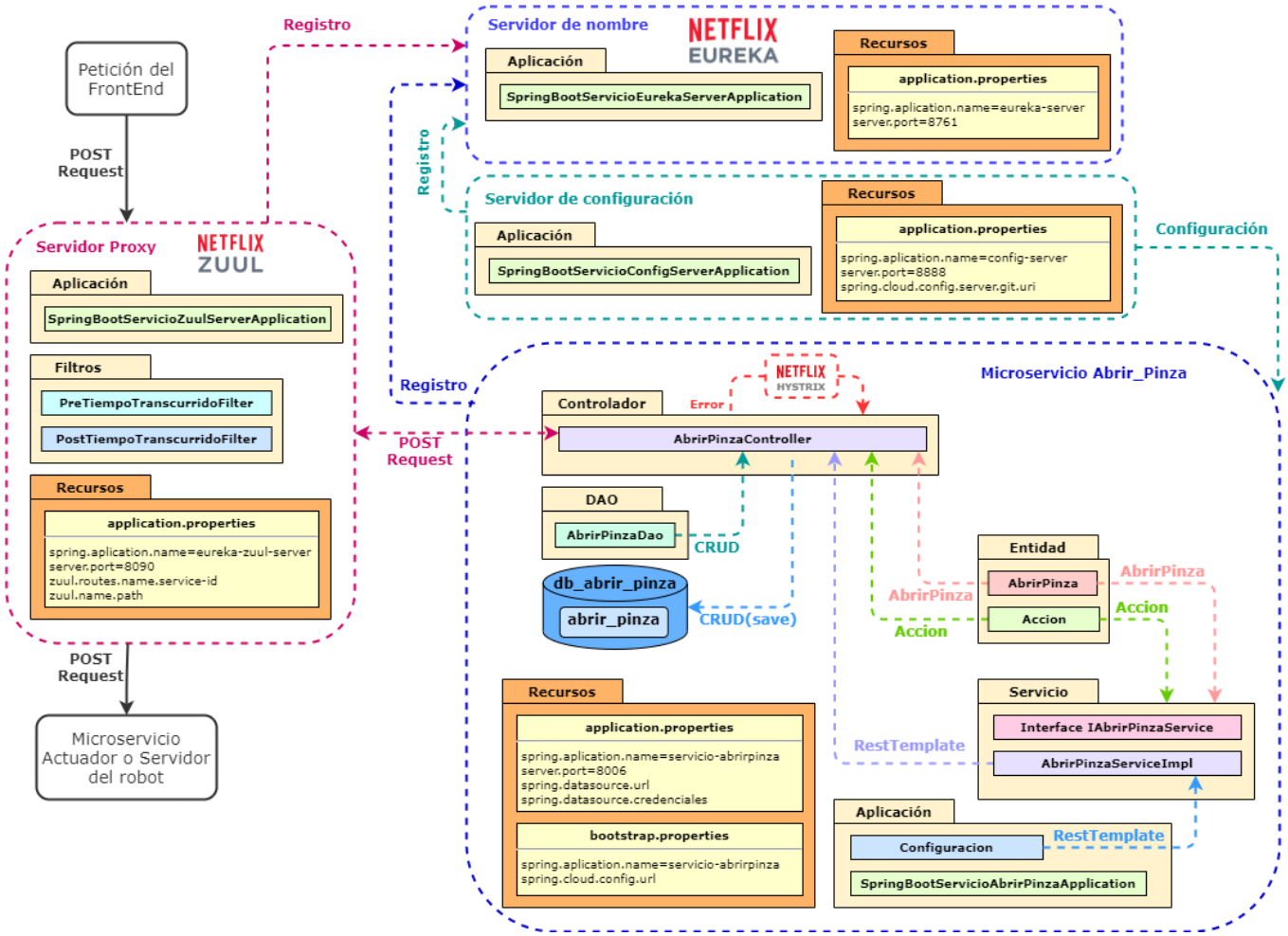


Funcionamiento del Backend. Después de detallar cada una de las partes importantes del Backend, cabe explicar el funcionamiento del mismo, como ya se explicó en el capítulo 3 en el diseño de toda la arquitectura del backend, todo el procesa se da a partir de la petición qué es recibida desde la aplicación de control en el Frontend y termina con el envió de los datos hacia el servidor del robot o en este ejemplo. En la figura 104 se puede ver el funcionamiento del backend utilizando el microservicio "*Abrir_Pinza",* que se ha usado de ejemplo a lo largo de este capítulo, en primer lugar, debe levantarse el servidor Eureka para que el resto de microservicios puedan registrarse y localizarse, el servidor de configuración se utiliza para asignar ciertas propiedades al microservicio. Dentro del microservicio los paquetes comparten atributos y prestaciones entre sí que son importantes para el correcto funcionamiento de la arquitectura, el proceso empieza desde que el servidor Proxy Zuul recibe la petición del Frontend, a continuación, envía la petición POST al controlador del microservicio, este devuelve la respuesta y el servidor Proxy envía la petición al siguiente microservicio que se requiera para completar el proceso o al servidor del robot. Es importante mencionar

que en caso de error el controlador del microservicio abre el circuito y utiliza Hystrix para gestionar la tolerancia a fallos, y también el controlador es el encargado de guardar los datos en la base de datos. Finalmente, como se ve en la figura, cada uno de los componentes del Backend al ser microservicios disponen de un paquete de recursos que contienen todas sus propiedades principales.

Figura 104

Funcionamiento del Backend



Despliegue de la arquitectura final.

Después de haber detallado a profundidad cada uno de los componentes de la arquitectura, para que el proyecto cumpla con uno de los objetivos principales relacionado con el control telemático, el despliegue se lo realiza en internet con la utilización de un VPS, las características del mismo se encuentran detalladas en la tabla 33. En el VPS se encuentra tanto la aplicación de control como la arquitectura del Backend desarrollada en STS 4.

Tabla 33

Características del VPS.

VPS M SSD	
Núcleos	6 vCPU
Memoria RAM	16 GB
Memoria de almacenamiento	400 GB SSD
Velocidad de puertos	400 Mbit/s
Snapshots disponibles	2 Snapshots
Tráfico de entrada y velocidad	32 TB (100 Mbit/s)
Sistema operativo	Windows Server

Despliegue del frontend.

Para el despliegue de la aplicación de control se puede utilizar un panel de control, en este caso se utiliza Plesk, o desplegarlo directamente por medio de la manipulación de Windows Server, de todos modos, se puede obtener una dirección de dominio temporal o utilizar el DNS que el VPS proporciona, de este modo se tiene las siguientes direcciones utilizadas:

- **DNS:** http://vmi632710.contaboserver.net
- **Dominio temporal:** https://jolly-ritchie.144-126-146-110.plesk.page/

Despliegue del backend.

El procedimiento para desplegar el Backend consiste netamente en replicar lo que se realiza en la computadora local, es decir que para que funcione correctamente solo se debe ejecutar cada uno de los microservicios, teniendo en cuenta que se debe corregir la configuración de la dirección de destino del robot, ya que se encuentran en redes distintas. De este modo el procedimiento que se debe seguir en el VPS para que la arquitectura funcione correctamente, es la siguiente:

- Instalar el JDK de java, no importa la versión.
- Instalar Spring Tools Suite 4, version 4.9 RELEASE.
- Implementar todos los microservicios, copiándolos desde la computadora local.
- Cambiar las direcciones de dominio o direcciones IP de los microservicios que se requiera.
- Levantar todos los microservicios.

Una vez se realiza el procedimiento anterior, se debe tener en cuenta que se utiliza un Gateway para acceder al Backend, de este modo se utiliza el DNS del VPS y el puerto del Gateway, quedando la dirección de la siguiente manera:

- **Gateway:** http://vmi632710.contaboserver.net:8090/api

Despliegue del servidor del robot.

Como ya se ha mencionado, el servidor del robot se levanta en una Raspberry Pi, de este modo para poder comunicarse con la misma desde redes externas, se tiene

dos opciones, hacer una redirección de puertos en el caso de qué el proveedor de internet lo permita, o utilizar un servicio de túnel, para redireccionar la dirección IP del servidor local del robot, en este caso se utiliza la segunda, se utiliza NGROK, un servicio gratis, para acceder a este servicio se debe descargar un plugin en la Raspberry Pi y se ingresa el siguiente comando:

```
./ngrok http 5000
```

De este modo se obtiene una dirección URL que representa el túnel y redirecciona la dirección URL del servidor local del robot, quedando de la siguiente manera:

```
http://41fe86992076.ngrok.io = http://localhost:5000
```

Finalmente cabe mencionar que, para utilizar la aplicación de control, al tratarse de una aplicación en desarrollo, es considerada hostil por la mayoría de navegadores por lo que se recomienda utilizarla en Google Chrome y en la configuración del sitio activar el contenido inseguro.

Dicho esto, en la figura 105 se puede ver la arquitectura final desplegada y en la tabla 34 se puede ver la recopilación final de todos los orígenes, destinos, respectivos mensajes JSON que conforman las peticiones HTTP de tipo POST y también las rutas a partir de la ruta principal que recorren las mismos hasta que llegan al servidor del robot y sus actuadores. Adicionalmente cabe mencionar que las peticiones HTTP de tipo GET para obtener las bases de datos de cada microservicio, se pueden obtener con la ruta /lista_local para los microservicios individuales o /listar_acciones para los microservicios que tengan relación entre sí.

Tabla 34

Peticiones HTTP POST de la arquitectura final.

Origen	Destino	Ruta	Mensaje JSON
Botón mover adelante ↑	Servicio mover adelante	http://vmi632710.contaboserver.net:8090 /api/moveradelante/enviar_izquierdo /api/moveradelante/enviar_derecho	{“movimiento”:”Adelante”, “fecha”:”dd/mm/aaaa - hora”}
Botón mover atrás ↓	Servicio mover atras	http://vmi632710.contaboserver.net:8090 /api/moveratras/enviar_izquierdo /api/moveratras/enviar_derecho	{“movimiento”:”Atras”, “fecha”:” …”}
Botón mover izquierda ←	Servicio mover izquierda	http://vmi632710.contaboserver.net:8090 /api/moverizquierda/enviar_izquierdo /api/moverizquierda/enviar_derecho	{“movimiento”:”Izquierda”, “fecha”:” …”}
Botón mover derecha →	Servicio mover derecha	http://vmi632710.contaboserver.net:8090 /api/moverderecha/enviar_izquierdo /api/moverderecha/enviar_derecho	{“movimiento”:”Derecha”, “fecha”:” …”}
Botón subir brazo ↑	Servicio subir brazo	http://vmi632710.contaboserver.net:8090 /api/subirbrazo/enviar	{“movimiento”:”Subir Brazo”, “fecha”:” …”}
Botón bajar brazo ↓	Servicio bajar brazo	http://vmi632710.contaboserver.net:8090 /api/bajarbrazo/enviar	{“movimiento”:”Bajar Brazo”, “fecha”:” …”}
Botón subir antebrazo ↑	Servicio subir antebrazo	http://vmi632710.contaboserver.net:8090 /api/subirantebrazo/enviar	{“movimiento”:”Subir Antebrazo”, “fecha”:” …”}
Botón bajar antebrazo ↓	Servicio bajar antebrazo	http://vmi632710.contaboserver.net:8090 /api/bajarantebrazo/enviar	{“movimiento”:”Bajar Antebrazo”, “fecha”:” …”}
Botón mano izquierda ←	Servicio mano izquierda	http://vmi632710.contaboserver.net:8090 /api/manoizquierda/enviar	{“movimiento”:”Mano a la Izquierda”, “fecha”:” …”}
Botón mano derecha →	Servicio mano derecha	http://vmi632710.contaboserver.net:8090 /api/manoderecha/enviar	{“movimiento”:”Mano a la Derecha”, “fecha”:” …”}
Botón abrir pinza	Servicio abrir pinza	http://vmi632710.contaboserver.net:8090 /api/abrirpinza/enviar	{“movimiento”:”Abrir Pinza”, “fecha”:” …”}

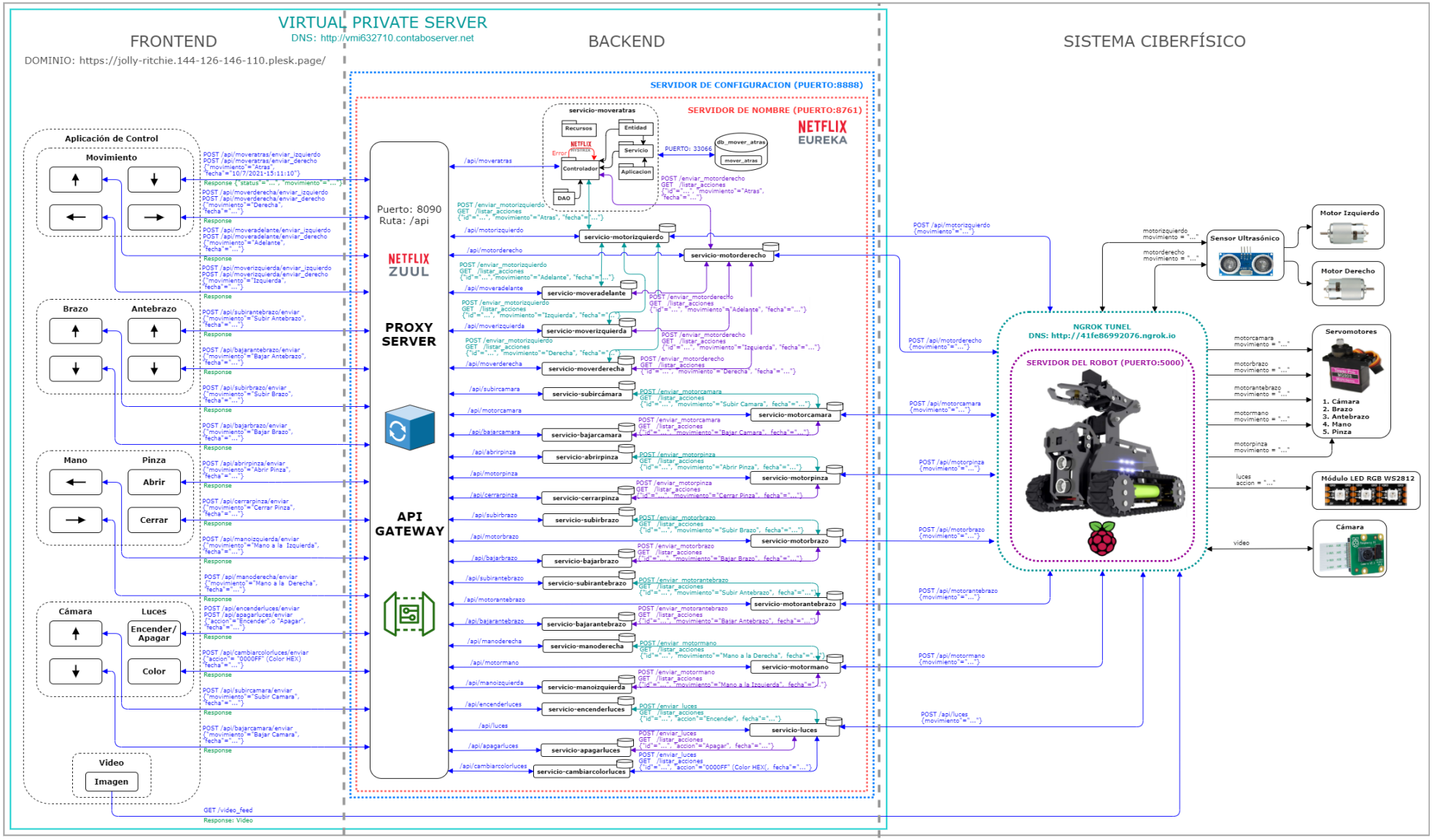
Origen	Destino	Ruta	Mensaje JSON
Botón cerrar pinza	Servicio cerrar pinza	http://vmi632710.contaboserver.net:8090 /api/cerrarpinza/enviar	{“movimiento”:”Cerrar Pinza”, “fecha”:” …”}
Botón subir cámara ↑	Servicio subir cámara	http://vmi632710.contaboserver.net:8090 /api/subircamara/enviar	{“movimiento”:”Subir Camara”, “fecha”:” …”}
Botón bajar cámara ↓	Servicio bajar cámara	http://vmi632710.contaboserver.net:8090 /api/bajarcamara/enviar	{“movimiento”:”Bajar Camara”, “fecha”:” …”}
Botón encender o apagar luces	Servicio encender luces y apagar luces	http://vmi632710.contaboserver.net:8090 /api/encenderluces/enviar /api/apagarluces/enviar	{“accion”:” Encender”, “fecha”:” …”} {“accion”:” Apagar”, “fecha”:” …”}
Botón color luces	Servicio cambiar color luces	http://vmi632710.contaboserver.net:8090 /api/cambiarcolorluces/enviar	{“accion”: “0000ff” (Color HEX), “fecha”:” …”}
Servicio mover adelante	Servicio motor izquierdo y Servicio motor derecho	http://servicio-motorizquierdo/enviar http://servicio-motorderecho/enviar	{“movimiento”:”Adelante”, “fecha”:” …”}
Servicio mover atrás	Servicio motor izquierdo y Servicio motor derecho	http://servicio-motorizquierdo/enviar http://servicio-motorderecho/enviar	{“movimiento”:”Atras”, “fecha”:” …”}
Servicio mover derecha	Servicio motor izquierdo y Servicio motor derecho	http://servicio-motorizquierdo/enviar http://servicio-motorderecho/enviar	{“movimiento”:”Derecha”, “fecha”:” …”}
Servicio mover izquierda	Servicio motor izquierdo y Servicio motor derecho	http://servicio-motorizquierdo/enviar http://servicio-motorderecho/enviar	{“movimiento”:”Izquierda”, “fecha”:” …”}
Servicio subir brazo	Servicio motor brazo	http://servicio-motorbrazo/enviar	{“movimiento”:”Subir Brazo”, “fecha”:” …”}
Servicio bajar brazo	Servicio motor brazo	http://servicio-motorbrazo/enviar	{“movimiento”:”Bajar Brazo”, “fecha”:” …”}

Origen	Destino	Ruta	Mensaje JSON
Servicio subir antebrazo	Servicio motor antebrazo	http://servicio-motorantebrazo/enviar	{“movimiento”:”Subir Antebrazo”, “fecha”:” …”}
Servicio bajar antebrazo	Servicio motor antebrazo	http://servicio-motorantebrazo/enviar	{“movimiento”:”Bajar Antebrazo”, “fecha”:” …”}
Servicio mano izquierda	Servicio motor mano	http://servicio-motormano/enviar	{“movimiento”:”Mano a la Izquierda”, “fecha”:” …”}
Servicio mano derecha	Servicio motor mano	http://servicio-motormano/enviar	{“movimiento”:”Mano a la Derecha”, “fecha”:” …”}
Servicio abrir pinza	Servicio motor pinza	http://servicio-motorpinza/enviar	{“movimiento”:”Abrir Pinza”, “fecha”:” …”}
Servicio cerrar pinza	Servicio motor pinza	http://servicio-motorpinza/enviar	{“movimiento”:”Cerrar Pinza”, “fecha”:” …”}
Servicio subir cámara	Servicio motor cámara	http://servicio-motorcamara/enviar	{“movimiento”:”Subir Camara”, “fecha”:” …”}
Servicio bajar cámara	Servicio motor cámara	http://servicio-motorcamara/enviar	{“movimiento”:”Bajar Camara”, “fecha”:” …”}
Servicio encender luces	Servicio luces	http://servicio-luces/enviar	{“accion”:” Encender”, “fecha”:” …”}
Servicio apagar luces	Servicio luces	http://servicio-luces/enviar	{“accion”:” Apagar”, “fecha”:” …”}
Servicio cambiar color luces	Servicio luces	http://servicio-cambiarcolorluces/enviar	{“accion”: “0000ff” (Color HEX), “fecha”:” …”}
Servicio motor izquierdo	Motor izquierdo del robot	http://41fe86992076.ngrok.io /api/motorizquierdo	{“movimiento”:”Adelante”} {“movimiento”:”Atras”} {“movimiento”:”Izquierda”} {“movimiento”:”Derecha”}
Servicio motor derecho	Motor derecho del robot	http://41fe86992076.ngrok.io /api/motorderecho	{“movimiento”:”Adelante”} {“movimiento”:”Atras”} {“movimiento”:”Izquierda”} {“movimiento”:”Derecha”}

Origen	Destino	Ruta	Mensaje JSON
Servicio motor brazo	Motor del brazo del robot	http://41fe86992076.ngrok.io /api/motorbrazo	{“movimiento”:”Subir Brazo”} {“movimiento”:”Bajar Brazo”}
Servicio motor antebrazo	Motor del antebrazo del robot	http://41fe86992076.ngrok.io /api/motorantebrazo	{“movimiento”:”Subir Antebrazo”} {“movimiento”:”Bajar Antebrazo”}
Servicio motor mano	Motor del mano del robot	http://41fe86992076.ngrok.io /api/motormano	{“movimiento”:”Mano a la Izquierda”} {“movimiento”:” Mano a la Derecha”}
Servicio motor pinza	Motor de la pinza del robot	http://41fe86992076.ngrok.io /api/motorpinza	{“movimiento”:”Abrir Pinza”} {“movimiento”:” Cerrar Pinza”}
Servicio motor cámara	Motor de la cámara del robot	http://41fe86992076.ngrok.io /api/motorcamara	{“movimiento”:”Subir Camara”} {“movimiento”:”Bajar Camara”}
Servicio luces	Luces del robot	http://41fe86992076.ngrok.io /api/luces	{“accion”: “Encender”} {“accion”: “Apagar”} {“accion”: “0000ff” (Color HEX)}

Figura 105

Arquitectura final implementada y desplegada.



Capítulo V

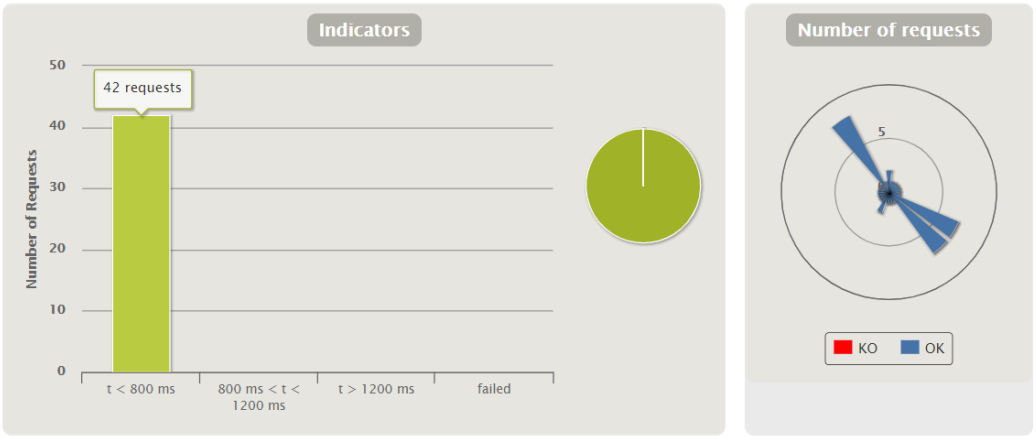
Pruebas de Validación

Pruebas de funcionamiento

Una vez la arquitectura se encuentra completamente implementada y desplegada, para las pruebas de funcionamiento se utiliza la herramienta Gatling para enviar una secuencia de peticiones los más rápido posible, ya que si bien la aplicación de control ayudaría a evaluar el funcionamiento de toda la arquitectura del sistema, existe la limitante de que las notificaciones no permiten maximizar la cantidad de peticiones por segundo, es por ello que en el script utilizado en Gatling se configuran todas las peticiones posibles para evaluar el funcionamiento total del sistema, de este modo se obtienen 42 peticiones en total que se pueden ver en la figura 106 que lograron completarse con éxito y con un tiempo de respuesta menor a los 800ms. y en la figura 107 se puede ver la lista de todas las peticiones realizadas con mayor detalle. De este modo se concluye que el funcionamiento de la arquitectura total del sistema es óptimo.

Figura 106

Peticiones y tiempo de respuesta de la prueba de funcionamiento.



Nota: Figura obtenida de la carta de datos generada por Gatling.

Figura 107

Peticiones de la prueba de funcionamiento en Gatling.

STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	42	42	0	0%	1.75	366	433	491	659	747	767	466	95
mano_derecha	2	2	0	0%	0.083	482	625	696	753	764	767	625	143
mover_ad..._derecho	1	1	0	0%	0.042	563	563	563	563	563	563	563	0
mover_ad...zquierdo	1	1	0	0%	0.042	492	492	492	492	492	492	492	0
mover_atras_derecho	1	1	0	0%	0.042	425	425	425	425	425	425	425	0
mover_at...zquierdo	1	1	0	0%	0.042	381	381	381	381	381	381	381	0
mover_iz..._derecho	1	1	0	0%	0.042	458	458	458	458	458	458	458	0
mover_iz...zquierdo	1	1	0	0%	0.042	429	429	429	429	429	429	429	0
mover_de..._derecho	7	7	0	0%	0.292	381	387	459	653	706	719	453	115
mover_de...zquierdo	7	7	0	0%	0.292	366	397	425	464	473	475	406	36
subir_brazo	1	1	0	0%	0.042	661	661	661	661	661	661	661	0
subir_antebrazo	1	1	0	0%	0.042	587	587	587	587	587	587	587	0
abrir_pinza	1	1	0	0%	0.042	567	567	567	567	567	567	567	0
mano_izquierda	2	2	0	0%	0.083	433	494	525	549	554	555	494	61
bajar_brazo	1	1	0	0%	0.042	432	432	432	432	432	432	432	0
bajar_antebrazo	1	1	0	0%	0.042	447	447	447	447	447	447	447	0
cerrar_pinza	1	1	0	0%	0.042	445	445	445	445	445	445	445	0
subir_camara	1	1	0	0%	0.042	557	557	557	557	557	557	557	0
bajar_camara	1	1	0	0%	0.042	394	394	394	394	394	394	394	0
encender_luces	1	1	0	0%	0.042	631	631	631	631	631	631	631	0
cambiar_color_luces	8	8	0	0%	0.333	376	418	438	472	480	482	423	31
apagar_luces	1	1	0	0%	0.042	487	487	487	487	487	487	487	0

Nota: Figura obtenida de la carta de datos generada por Gatling.

Pruebas de carga

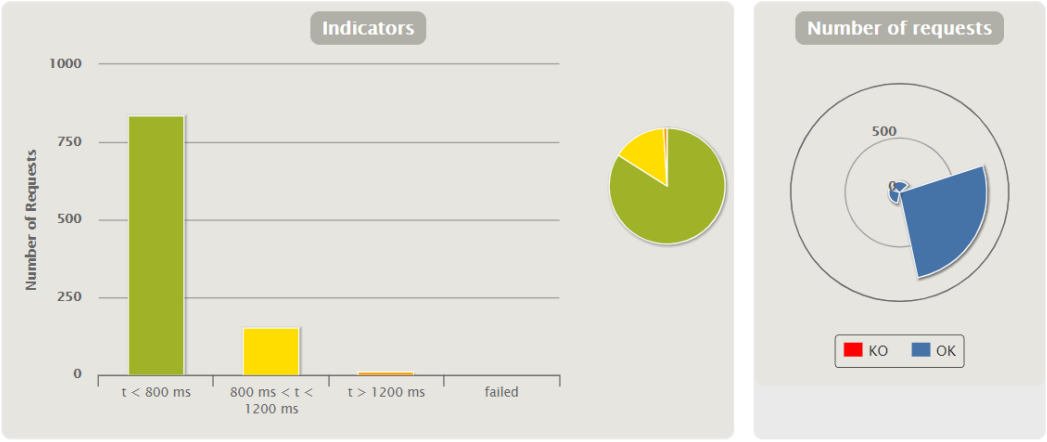
Al igual que para la prueba de funcionamiento, para las pruebas de carga se utiliza Gatling por las prestaciones que brinda tanto para configurar usuarios y peticiones por segundo, de este modo se configuran 10 peticiones totales para 100, 200, 400, 800, 1000 y 5000 usuarios en un lapso de 60 segundos.

Prueba de carga con 100 usuarios.

Los resultados obtenidos para 100 usuarios se pueden observar en la figura 108, donde se puede ver que las 1000 peticiones en total lograron completarse en su totalidad y con todas debajo de 1200ms de respuesta, adicionalmente en la figura 109 se puede ver qué el pico de peticiones por segundo fue de 20 peticiones con 17 usuarios activos.

Figura 108

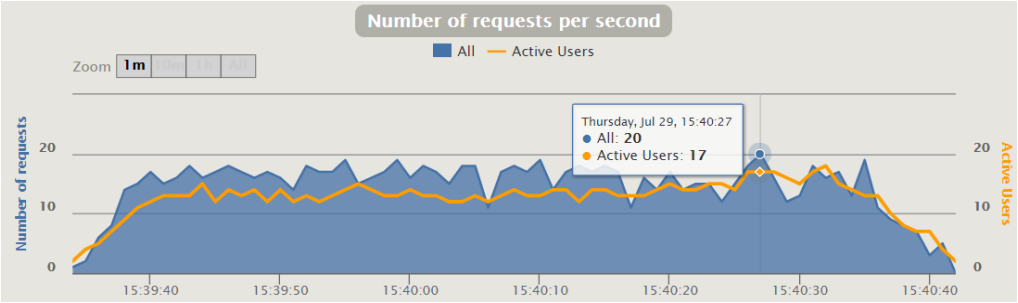
Prueba de carga con 100 usuarios.



Nota: Figura obtenida de la carta de datos generada por Gatling.

Figura 109

Pico de peticiones para 100 usuarios.



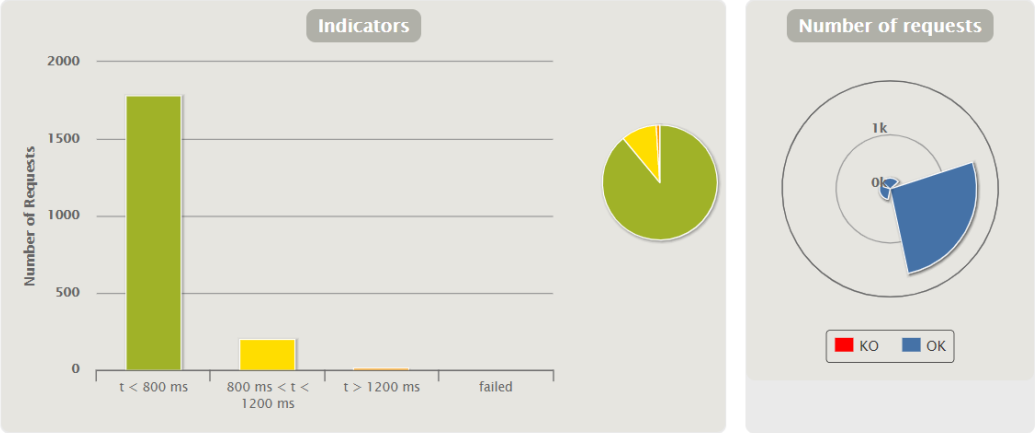
Nota: Figura obtenida de la carta de datos generada por Gatling.

Prueba de carga con 200 usuarios

Los resultados obtenidos para 200 usuarios se pueden observar en la figura 110, donde se puede ver que las 2000 peticiones en total lograron completarse en su totalidad y con todas debajo de 1200ms de respuesta, adicionalmente en la figura 111 se puede ver qué el pico de peticiones por segundo fue de 65 peticiones con 25 usuarios activos.

Figura 110

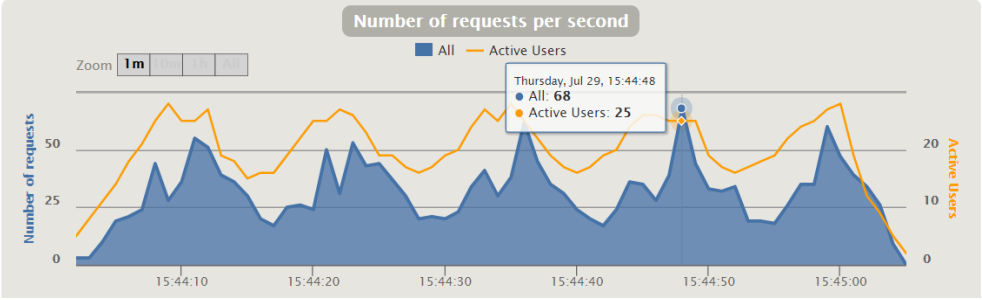
Prueba de carga con 200 usuarios.



Nota: Figura obtenida de la carta de datos generada por Gatling.

Figura 111

Pico de peticiones para 200 usuarios.



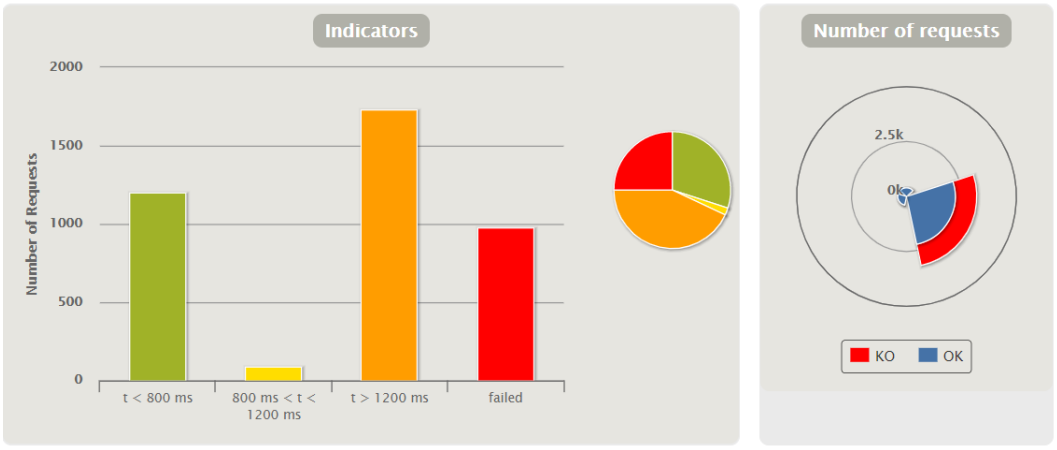
Nota: Figura obtenida de la carta de datos generada por Gatling.

Prueba de carga con 400 usuarios

Los resultados obtenidos para 400 usuarios se pueden observar en la figura 112, donde se puede ver que, de las 4000 peticiones en total, solo lograron completarse 75% y el 43% sobre 1200ms de respuesta, adicionalmente en la figura 113 se puede ver qué el pico de peticiones por segundo fue de 158 peticiones con 94 usuarios activos.

Figura 112

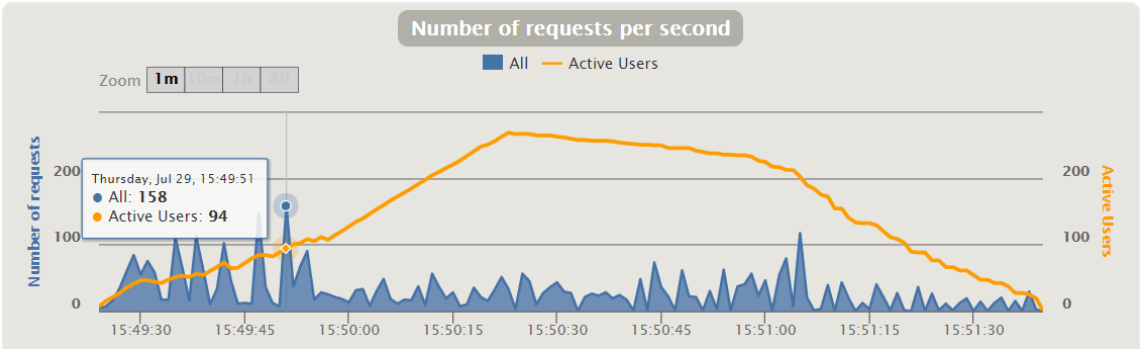
Prueba de carga con 400 usuarios.



Nota: Figura obtenida de la carta de datos generada por Gatling.

Figura 113

Pico de peticiones para 400 usuarios.



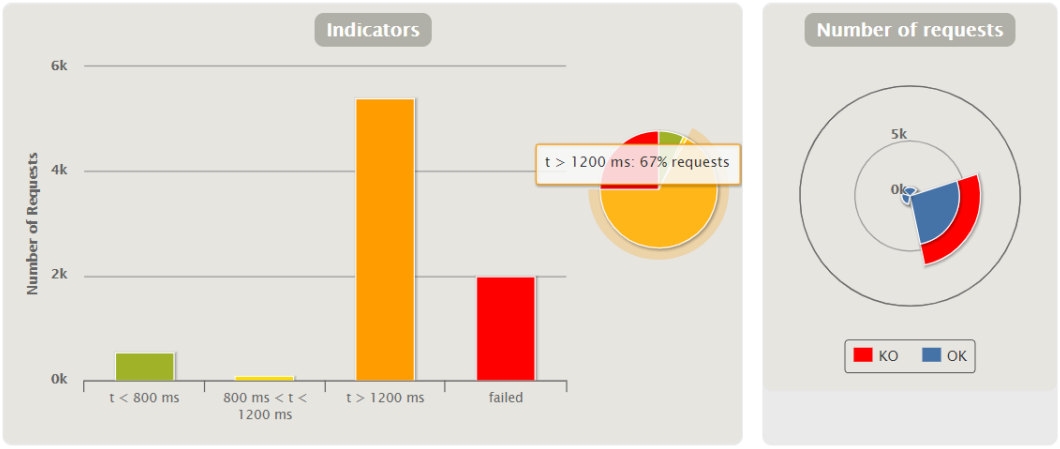
Nota: Figura obtenida de la carta de datos generada por Gatling.

Prueba de carga con 800 usuarios

Los resultados obtenidos para 800 usuarios se pueden observar en la figura 114, donde se puede ver que de las 8000 peticiones en total lograron completarse 75% de las peticiones, pero con un 67% con tiempo de respuesta mayor a 1200ms, adicionalmente en la figura 115 se puede ver qué el pico de peticiones por segundo fue de 146 peticiones con 68 usuarios activos.

Figura 114

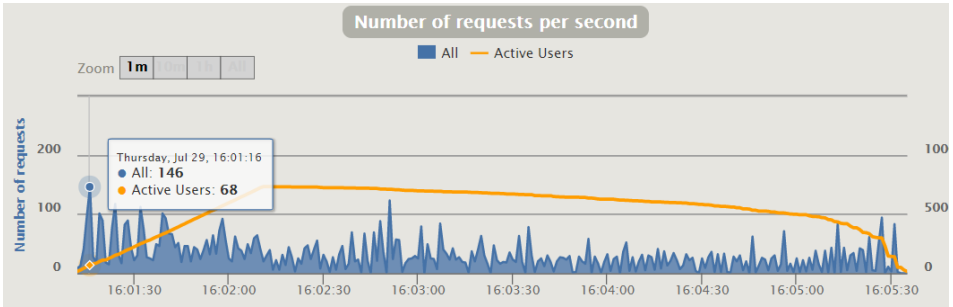
Prueba de carga con 800 usuarios.



Nota: Figura obtenida de la carta de datos generada por Gatling.

Figura 115

Pico de peticiones para 800 usuarios.



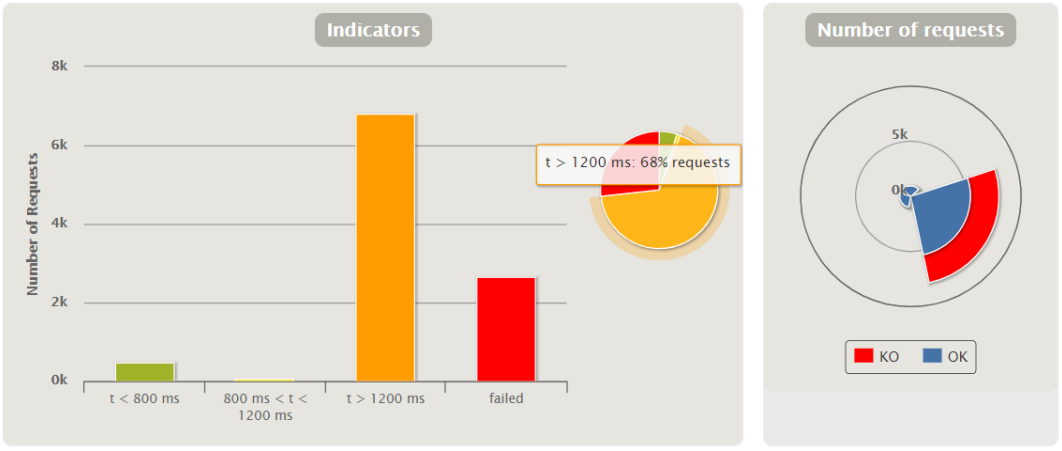
Nota: Figura obtenida de la carta de datos generada por Gatling.

Prueba de carga con 1000 usuarios

Los resultados obtenidos para 10000 usuarios se pueden observar en la figura 116, donde se puede ver que de las 10000 peticiones en total lograron completarse 73% de las peticiones, pero con un 68% con tiempo de respuesta mayor a 1200ms, adicionalmente en la figura 117 se puede ver qué el pico de peticiones por segundo fue de 162 peticiones con 95 usuarios activos.

Figura 116

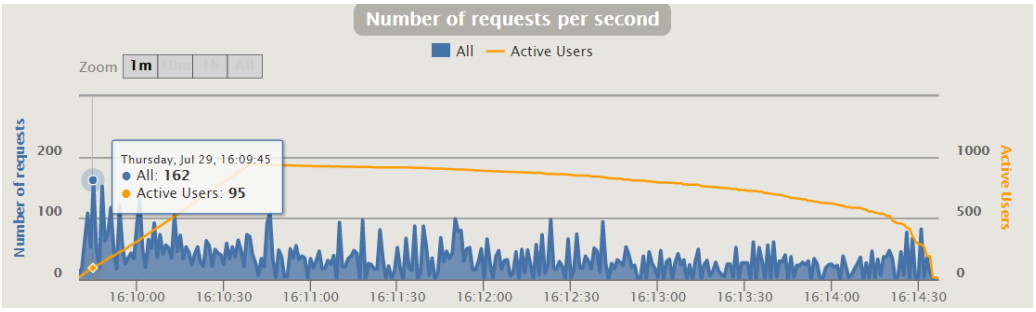
Prueba de carga con 1000 usuarios.



Nota: Figura obtenida de la carta de datos generada por Gatling.

Figura 117

Pico de peticiones para 1000 usuarios.



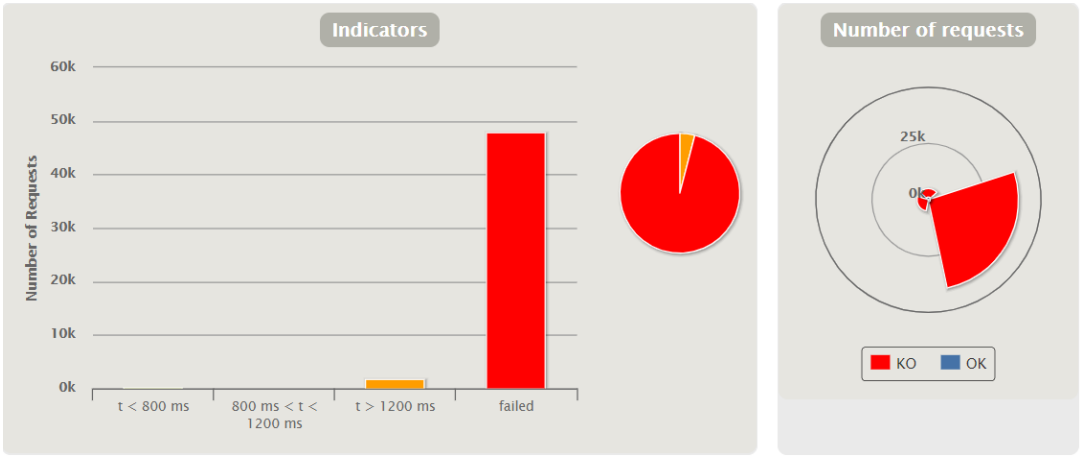
Nota: Figura obtenida de la carta de datos generada por Gatling.

Prueba de carga con 5000 usuarios

Los resultados obtenidos para 5000 usuarios se pueden observar en la figura 118, donde se puede ver que de las 50000 peticiones en total lograron completarse 4% de las peticiones y el 96% fallaron completamente, adicionalmente en la figura 119 se puede ver qué el pico de peticiones por segundo fue de 501 peticiones con 4950 usuarios activos.

Figura 118

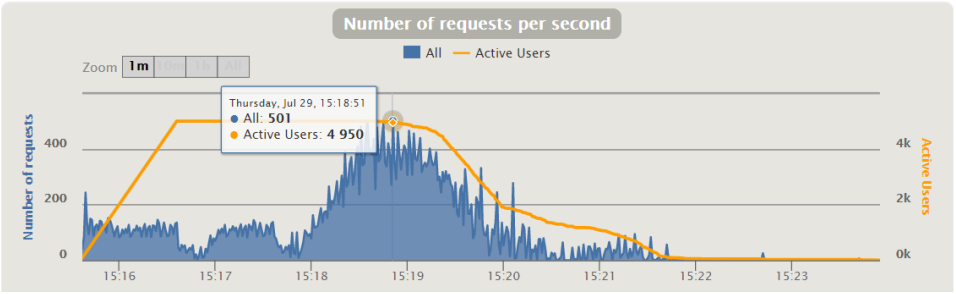
Prueba de carga con 5000 usuarios.



Nota: Figura obtenida de la carta de datos generada por Gatling.

Figura 119

Pico de peticiones para 5000 usuarios.



Nota: Figura obtenida de la carta de datos generada por Gatling.

Finalmente se puede mencionar que las pruebas de carga cumplen con lo esperado, ya que al tratarse de un motor de compilación Apache Maven, se espera que el máximo de peticiones por segundo sean 160, de este modo al momento de acercarse a este valor, se puede ver como el sistema empieza a bajar su rendimiento, hasta que con un número mucho mayor el sistema simplemente no logra responder. En la tabla 35 se puede observar el resumen de las pruebas de carga y el comportamiento del sistema respecto a las peticiones por segundo.

Tabla 35

Datos recopilados de las pruebas de carga

Usuarios	Peticiones Totales	Máximas Pet/s	% Éxito	% Fallo	%Respuesta t < 800ms	%Respuesta t < 1200ms
100	1000	20	100%	0%	100%	0%
200	2000	68	100%	0%	100%	0%
400	4000	158	75%	25%	32%	43%
800	8000	146	75%	25%	7%	67%
1000	10000	162	73%	27%	6%	68%
5000	50000	501	4%	96%	0%	100%

Pruebas de usabilidad

Para poder determinar qué tan amigable es para el usuario la aplicación de control del sistema, se utiliza el sistema de escalas de usabilidad (System Usability Scale, SUS). Se evaluó a 10 usuarios que pudieron utilizar la aplicación y probar el robot remotamente, de este modo se obtuvo las respuestas presentadas en la tabla 36, donde ya se encuentran ponderadas las respuestas junto a las respectivas consideraciones que el test debe seguir para obtener un valor adecuado, como se

puede ver después de obtener cada uno de los resultados y promediarlos, se obtiene el valor de 90.15/100, lo que muy gratamente indica que la aplicación es sumamente útil, fácil de usar y amigable con cualquier tipo de usuario.

Tabla 36

Resultados del test de usabilidad.

Preguntas / Usuarios	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	Puntaje SUS
U1	5	1	5	1	5	1	5	1	5	1	100
U2	5	1	5	2	5	5	5	1	5	1	87,5
U3	3	1	5	1	5	1	5	5	5	1	85
U4	4	3	5	1	4	2	5	1	5	1	87,5
U5	4	5	5	1	5	1	5	1	5	1	87,5
U6	5	2	4	1	4	1	5	1	5	1	92,5
U7	5	1	5	1	5	1	5	5	5	1	90
U8	5	1	5	2	5	1	5	1	5	1	97,5
U9	5	2	5	2	5	1	5	1	5	1	95
U10	5	2	4	5	5	1	5	3	5	1	80
									Promedio:		90.25

Finalmente se puede mencionar qué la pregunta que más variación tuvo respecto a sus respuestas es la segunda, esto quizá sea porqué los usuarios entendieron la pregunta refiriéndose al sistema en general y no únicamente a su experiencia con la aplicación de control.

Capítulo VI

Conclusiones y Recomendaciones

Conclusiones

Tras el trabajo investigativo de las múltiples herramientas, tecnologías y tendencias más actuales, se logró diseñar, implementar y desplegar una arquitectura que cumple con los objetivos propuestos, ya que está orientada al control telemático de sistemas ciber físicos, está estructurada bajo una metodología orientada a servicios, los cuales son tolerantes a fallos, independientes y totalmente escalables. Además, se puso en práctica un caso de prueba para el control de un robot móvil usando una aplicación de control a la cual se puede acceder desde cualquier dispositivo.

Se pudo implementar en su totalidad una arquitectura RESTful, valiéndose de todas las prestaciones que brindan cada una de las tecnologías utilizadas, así como JavaScript en el frontend, Spring Boot en el backend y Python-Flask en el servidor interno del robot. En cada una se pudo utilizar peticiones HTTP características de REST que permiten que los mensajes de tipo JSON sean enviados y recibidos sin problema entre cada uno de los componentes.

Se pudo acoplar cada una de las peticiones HTTP dependiendo del requerimiento, de este modo se pudo interconectar elementos importantes dentro de la arquitectura y brindar una versatilidad adecuada ya sea para obtener información con peticiones de tipo GET o para enviar información con peticiones de tipo POST, además se configuró correctamente cada uno de los componentes de la arquitectura necesarios para qué no exista ningún inconveniente al momento de usar este protocolo de comunicación.

La utilización de JavaScript en el Frontend permite una gran versatilidad al momento de abrir peticiones HTTP, ya que dependiendo de sus configuraciones permite

que la aplicación de control pueda comunicarse con cualquier tipo de arquitectura de Backend y gestionar de manera adecuada el envío de datos y las respuestas. Sin embargo, siempre se necesita tener una conexión a internet por la utilización de los paquetes de JavaScript, tanto para la generación de las peticiones HTTP, como para las características visuales de las notificaciones.

Utilizar las herramientas y tecnologías open source ofrecidas por Netflix, como el servidor de nombre Eureka y el Gateway-Proxy Zuul, permitió diseñar una arquitectura de microservicios, donde cada uno de estos es totalmente escalable, independiente, desacoplado y además ofrece una gran velocidad en cuanto a tiempos de respuesta tanto en envío y recepción de datos. Sin embargo, una desventaja es el alto consumo de los recursos del computador que ocupa y requiere cada microservicio.

Dentro de los microservicios se logró implementar la tolerancia a fallos gracias a Netflix Hystrix. Esta tecnología open source permite que cada uno de los microservicios puedan aislar puntos de falla en el caso de darse y enviar una señal de fallback que hace que el estado de la aplicación regrese a su estado anterior de ser necesario y también almacene todos los errores en las bases de datos respectivas de cada servicio.

Se implementó el servidor interno del robot utilizando Python junto con su Framework Flask, permitiendo un despliegue simple y rápido, además de la facilidad de poder utilizar las prestaciones de este lenguaje de programación para acoplar las diferentes, clases, archivos y librerías, relacionadas directamente con el servicio web y también con las características para poder manejar los sensores y actuadores del robot.

Para poder comunicarse a través de internet con el servidor interno del robot, se utilizó un servicio de túnel para redireccionar su dirección IP hacia el internet, logrando utilizar los sensores y actuadores del robot de manera adecuada, sin embargo, debido a la latencia, la transmisión de video se ve afectada y se refleja con delay en el FrontEnd.

La mejor alternativa para poder acceder al robot mediante el internet, es redireccionar los puertos del router local, sin embargo, esto puede verse limitado por el ISP.

Se diseño y desarrollo una aplicación multifuncional que puede ser utilizada en cualquier dispositivo siempre y cuando se le otorguen los permisos necesarios, además es la encargada de ofrecer una experiencia de usuario adecuada, ya que gestiona todas las respuestas enviadas desde el backend y el robot, brindando una alta confiabilidad y una interacción lo más fluida posible con el resto de la arquitectura del sistema.

Después de las pruebas de validación realizadas, se observó y concluyó que la arquitectura desarrollada cumple con todos los objetivos planteados, ya que su aplicación de control es amigable con el usuario, el backend gestiona una cantidad de peticiones por segundo aceptable para tratarse de un prototipo y el robot cumple con todas las características funcionales que se determinó en un principio.

Finalmente es importante mencionar que el despliegue de la arquitectura en el VPS se lo realizo sin ningún por menor utilizando las mismas versiones tanto de STS4 y el JDK con las que se empezó a diseñar y desarrollar el proyecto. Esto debido a que existen librerías o tecnologías que pueden ser incompatibles o pueden generar algún error con versiones más actuales.

Recomendaciones

Al momento de hacer el despliegue y en caso de no poder utilizar un servicio gratuito, se recomienda utilizar un servidor virtual, ya que sus prestaciones facilitan en gran medida el poder replicar y poner en marcha cualquier tipo de aplicación, de esté modo se logra qué la comunicación entre componentes de la arquitectura pueda darse de manera remota y no únicamente dentro de una red local.

Es muy importante seguir una metodología de investigación adecuada, ya qué para poder diseñar, desarrollar, implementar y desplegar una arquitectura, previamente ya se debe tener una visión general del comportamiento que se busca con el producto final, de este modo se pueden ahorrar en tiempos de programación, en recursos económicos y se puede evitar errores innecesarios.

Se debe tener en cuenta que, al tratarse de una arquitectura en desarrollo, al momento de usar la aplicación de control se debe otorgar los permisos necesarios para contenido inseguro, ya que los navegadores por lo general pueden considerarla hostil por la falta de certificaciones que obtiene una aplicación una vez que se la quiere empezar a producir y distribuir.

Finalmente, si se quiere replicar el trabajo presentado en este documento se debe utilizar las mismas versiones de las herramientas utilizadas. Siendo así la versión 4.9 RELEASE de STS 4, JDK de Java de 1.8 en adelante, MySQL Workbench 8.0, Python 3, Vanilla Javascript, HTML 5 y CSS3.

Trabajos Futuros.

Para los trabajos futuros se propone:

- Crear una aplicación enfocada especialmente en dispositivos móviles, esto con el fin de brindar un mejor soporte para los usuarios que prefieren usar un celular o una Tablet y no un computador necesariamente.
- Si bien el Backend funciona de manera adecuada, lo óptimo seria encapsular el proyecto dentro de un ejecutable que permita levantar todos los microservicios sin necesidad de entrar a ninguna herramienta de compilación como STS4.

- Si se lo lleva a producción, se debe implementar todos los protocolos de seguridad adecuados y habilitar todas las certificaciones para qué la aplicación no sea considerada como hostil para softwares de terceros como por ejemplo los navegadores de internet.
- Implementar protocolos de autenticación para brindar seguridad a cada uno de los usuarios, además de poder gestionar y enfocar aplicaciones específicas para cada uno de ellos, ya que la información permanecería segura y encriptada.
- En los microservicios se propone utilizar versiones actualizadas o inclusive tecnologías diferentes para poder mejorar el rendimiento y la capacidad de la arquitectura del sistema, de este modo evaluar qué sería lo más óptimo para un producto final lo más actualizado posible.

.

Acrónimos

- CRUD: Crear, Leer, Actualizar, Eliminar.
- CPS: Sistema Ciber-Físico
- CSS: Hoja de estilos en cascada.
- DAO: Objeto de acceso a datos.
- HTML: Lenguaje de marco de hipertexto.
- HTTP: Protocolo de trasferencia de hipertexto.
- IoT: Internet de las cosas.
- JS: JavaScript.
- Py: Python.
- REST: Transferencia de estado representacional.
- SLR: Revisión sistemática de la literatura.
- SMS: Mapeo sistemático de la literatura.
- SOA: Arquitectura orientada a servicios.
- SOAP: Protocolo de acceso de objeto simple.
- STS4: Spring Tools Suite 4.
- VPS: Servidor virtual privado.
- WoT: Web de las cosas.

Referencias

330Ohms. (10 de Abril de 2017). *330Ohms*. Obtenido de 330Ohms: https://blog.330ohms.com/2017/04/10/diferencias-entre-una-fotorresistencia-y-un-sensor-de-luz/

Adafruit. (23 de Septiembre de 2019). *Adafruit*. Obtenido de Adafruit PIR Motion Sensor: https://learn.adafruit.com/pir-passive-infrared-proximity-motion-sensor

Adafruit. (28 de Marzo de 2019). *Adrafruit*. Obtenido de Adafruit: https://www.adafruit.com/product/4095#:~:text=Dimensions%3A%2041%20x%2021%20mm

Adeept. (12 de 07 de 2018). *Adeept Rasptan*. Obtenido de Adeept: https://www.adeept.com/video/static1/itemsfile/Tutorial_12-11.pdf

Alam, M., Rufino, J., Ahmed, S., Ferreira, J., Ahmed, S., Shah, N., & Chen, Y. (2018). *Orchestration of Microservices for IoT Using Docker and Edge Computing.* New Jersey: IEEE Communications Magazine.

Alzaghoul, E. (12 de Febrero de 2021). *Wikimedia Inc*. Obtenido de Wikipedia la enciclopedia libre: https://en.wikipedia.org/wiki/Web_Services_Discovery

Arjonilla, R. (02 de Febrero de 2017). *Rafa Arjonilla*. Obtenido de Backend: https://rafarjonilla.com/que-es/backend/

Baboi, M., Iftene, A., & Gîfu, D. (2019). *Dynamic Microservices to Create Scalable and Fault Tolerance Architecture.* Bucharest: 23rd International Conference on Knowledge-Based and Intelligent Information & Engineering.

Badii, C., Pierfrancesco, B., Difino, A., Nesi, P., Pantaleo, G., & Paolucci, M. (2019). *MicroServices Suite for Smart City Applications.* Florencia: MDPI.

Bailey, J., Budgen, D., Turner, M., Kitchenham, B., Brereton, P., & Linkman, S. (2007). *Evidence relating to Object-Oriented software design: A survey.* Durham: IEEE.

Benayache, A., Bilami, A., Barkat, S., Lorenz , P., & Taleb, H. (2019). *MsM: A microservice middleware for smart WSN-based IoT application.* Colmar: ELSEVIER.

Bixio, L., Delzanno, G., Rebora, S., & Rulli, M. (2020). *A Flexible IoT Stream Processing Architecture.* Genova: MDPI.

Blancarte, O. (23 de Julio de 2014). *Oscar Blancarte Software Architect*. Obtenido de Oscar Blancarte Software Architect: https://www.oscarblancarteblog.com/2014/07/23/que-es-service-oriented-architecture-soa/

Blancarte, O. (17 de Marzo de 2017). *Oscar Blancarte Software Architect*. Obtenido de Reactive Programming: https://reactiveprogramming.io/blog/es/estilos-arquitectonicos/monolitico

Blancarte, O. (8 de Junio de 2018). *Oscar Blancarte Software Architect*. Obtenido de Reactive Programming: https://reactiveprogramming.io/blog/es/estilos-arquitectonicos/microservicios

Bliss, H. (17 de Julio de 2018). *FixterGeek*. Obtenido de FixterGeek: https://medium.com/fixtergeek/porqu%C3%A9-alguien-usar%C3%ADa-javascript-para-el-backend-e48442d9b561

Bricolabs Wiki . (11 de Mayo de 2018). *Bricolabs Wiki* . Obtenido de Bricolabs Wiki : https://bricolabs.cc/wiki/guias/control_de_motores#:~:text=Un%20motor%20de%20corriente%20continua,acci%C3%B3n%20de%20un%20campo%20magn%C3%A9tico.

Brito, A., Fetzer, C., Köpsell, S., Pietzuch, P., Pasin, M., Felber, P., . . . Fehér, M. (2019). *Secure end-to-end processing of smart metering data.* Campina Grande: Journal of Cloud Computing: Advances, Systems and Applications.

Butzin, B., Golatowski, F., & Timmermann, D. (2016). *Microservices Approach for the Internet of Things.* Rostock: IEEE.

Castro, J. D. (25 de 02 de 2021). *Platzi*. Obtenido de Platzi: https://platzi.com/blog/que-es-html-css-javascript/

Chakray. (2020). *¿QUÉ DIFERENCIAS HAY ENTRE REST Y SOAP?* Sevilla: Edge of the Web.

Christensen, B. (29 de febrero de 2012). *The Netflix Tech Blog*. Obtenido de Netflix Technology Blog: https://netflixtechblog.com/fault-tolerance-in-a-high-volume-distributed-system-91ab4faae74a

Cisco. (20 de Noviembre de 2019). *Cisco*. Obtenido de Cisco IoT: https://www.cisco.com/c/en/us/solutions/internet-of-things/overview.html

Cobos Domínguez, A. (2016). *Diseño e implementación de una arquitectura IoT basada en tecnologías Open Source.* Sevilla: Universidad de Sevilla.

Colomer, J. (2018). *Estudio de los sensores para la detección de obstáculos aplicables a robots móviles.* Cataluña: UOC.

Crespo, R. (16 de Agosto de 2016). *Roberto Crespo*. Obtenido de Roberto Crespo Blog Personal: http://www.robertocrespo.net/kaizen/microservicios-implementar-circuit-breaker-hystrix/

Departamento de la Ciencia de Computación e Inteligencia Artficial. (2014). *Orquestación de Servicios: BPEL.* Alicante: Universidad de Alicante.

Dewesoft. (13 de Marzo de 2020). *Dewesoft d.o.o*. Obtenido de Dewesoft: https://dewesoft.com/es/daq/que-es-un-sensor

Dipsis, N., & Stathis, K. (2019). *A RESTful middleware for AI controlled sensors, actuators and smart.* Londres: Journal of Ambient Intelligence and Humanized Computing.

Dragoni, N., Lanese, I., & Mazzara, M. (2018). *Microservices: How To Make Your Application Scale.* Sterling: Springer Verlag.

Eassa, A. (2018). *NoSQL Injection Attack Detection in Web Applications Using RESTful Service.* Sterling: Programming and Computer Software.

eCityclic. (20 de Abril de 2020). *eCityclic*. Obtenido de eCityclic: https://www.ecityclic.com/es/noticias/que-es-soa-o-arquitectura-orientada-a-servicios

Eismann, S., Kistowski, J., Grohmann, J., Bauer, A., Schmitt, N., & Kounev, S. (2019). *TeaStore - A Micro-Service Reference Application.* Wurzburg: FAS*W.

Expressif Systems. (1 de Septiembre de 2016). *Expressif Systems*. Obtenido de Expressif Systems: https://www.espressif.com/en/products/socs/esp32

Fan, P., Liu, J., Yin, W., Wang, H., Chen, X., & Sun, H. (2020). *2PC*: a distributed transaction concurrency control protocol of multi-microservice based on cloud computing platform.* Shanghai: SpringerOpen.

Gatling. (29 de 07 de 2013). *Gatling*. Obtenido de Gatling: https://gatling.io/open-source/

Gifu, D., & Pop, E. (2020). *Services Integration in the Cyber World.* Bucarest: IEEE.

Gorbenko, A., Romanovsky, A., & Tarasyuk, O. (2019). *Fault tolerant internet computing: Benchmarking and modelling trade-offs between availability, latency and consistency.* Leeds: ELSEVIER.

Grinberg, M. (19 de Mayo de 2020). *miguelgrinberg/flask-video-streaming*. Obtenido de GitHub Web Site: https://github.com/miguelgrinberg/flask-video-streaming

Guinard, D., Wilde, E., & Trifa, V. (2010). *A resource oriented architecture for the Web of Things.* Tokyo: IEEE Internet of Things (IOT).

Hoorn, A., Aleti, A., Dullmann, T., & Pitakrat, T. (2018). *ORCAS: Efficient Resilience Benchmarking of Microservice Architectures.* Stuttgart: ResearchGate.

Huang, L., Zhuang, W., Sun, M., & Zhang, H. (2020). *Research and Application of Microservice in Power Grid Dispatching Control System.* Nanjing: IEEE 4th Information Technology,Networking,Electronic and Automation Control Conference.

ISO/IEC JTC 1. (01 de Marzo de 2014). *ISO/IEC JTC 1*. Obtenido de ISO/IEC JTC 1: https://www.iso.org/files/live/sites/isoorg/files/developing_standards/docs/en/internet_of_things_report-jtc1.pdf

ITTGWEB. (26 de Febrero de 2021). *Ingeniería de Software*. Obtenido de Seguridad de Software: https://ittgweb.wordpress.com/2016/05/29/4-1-seguridad-de-software/

Kitchenham, B., & Charters, S. (2007). *Guidelines for performing Systematic Literature Reviews in Software Engineering.* Wellington: ResearchGate.

Kitchenham, B., Budgen, D., & Brereton, P. (2011). *Using mapping studies as the basis for further research – A participant-observer case study.* Staffordshire: ELSEVIER.

Kramer, M., Frese, S., & Kuijper, A. (2019). *Implementing secure applications in smart city clouds using.* Darmstadt: Elsevier.

Krivic, P., Skocir, P., Kusek, M., & Jezic, G. (2018). *Microservices as Agents in IoT Systems.* Roma: KES International Symposium.

Lakhan, A., & Li, X. (2019). *Transient fault aware application partitioning computational offloading algorithm in microservices based mobile cloudlet networks.* Nanjing: Springer.

Laleh, T., Paquet, J., Mokhov, S., & Yan , Y. (2018). *Constraint verification failure recovery in web service composition.* Montreal: Elsevier.

Llamas, L. (03 de Marzo de 2021). *Luis Llamas*. Obtenido de Luis Llamas, Ingeniería, Informática y Diseño: https://www.luisllamas.es/detectar-obstaculos-con-sensor-infrarrojo-y-arduino/

Lyu, M., Biennier, F., & Ghodous, P. (2019). *Control as a Service Architecture to Support Cloud-based and Event-driven Control Application Development.* Lyon: ICIOT.

Ma, S.-P., Fan, C.-Y., Chuang, Y., Liu, I.-H., & Lan, C.-W. (2019). *Graph-based and scenario-driven microservice analysis, retrieval, and testing.* Taipei: Elsevier.

Malyuga, K., Perl, O., Slapoguzov, A., & Perl, I. (2020). *Fault Tolerant Central Saga Orchestrator in RESTful Architecture.* Auckland: CONFERENCE OF FRUCT ASSOCIATION.

McNally, M., Chaplin, J., Martínez, G., & Ratchev, S. (2020). *Towards Flexible, Fault Tolerant Hardware Service Wrappers for the Digital Manufacturing on a Shoestring Project.* Nottingham: ELSEVIER.

Moguel Márquez, J. (2018). *Una arquitectura orientada a servicios y dirigida por eventos para el control inteligente de UAVs multipropósito.* Universidad de Extremadura, Tecnologías Informáticas. Badajoz: Universidad de Extremadura.

Morales, M. J. (26 de Febrero de 2021). *Modusoperantic*. Obtenido de Descubrimiento de los Servicios: https://www.modusoperantic.com/es/descubrimiento-de-los-servicios/

Moubarak, P. (2018). *Adaptive Manipulation of a Hybrid Mechanism Mobile Robot.* Washington: IEEE.

Mundowin. (26 de Febrero de 2021). *Mundowin*. Obtenido de Mundowin: https://mundowin.com/5-mejores-soluciones-de-balanceo-de-carga-para-una-distribucion-estable-del-trafico-de-la-red/

Naylamp Mechatronics. (24 de Marzo de 2018). *Naylamp Mechatronics*. Obtenido de TUTORIAL MPU6050, ACELERÓMETRO Y GIROSCOPIO: https://naylampmechatronics.com/blog/45_tutorial-mpu6050-acelerometro-y-giroscopio.html

NEO. (02 de febrero de 2014). *NEO: Networking and Emerging Optimization*. Obtenido de Herramientas Web para la enseñanza de protocolos de comunicación: https://neo.lcc.uma.es/evirtual/cdd/tutorial/aplicacion/http.html

Nestrategia. (25 de 02 de 2021). *Nestragia, tu agencia de posicionamiento web en Madrid*. Obtenido de NESTRATEGIA: https://nestrategia.com/desarrollo-web-back-end-front-end/

Netflix. (18 de Noviembre de 2018). *GitHub*. Obtenido de GitHub: https://github.com/Netflix/Hystrix

NSF. (1 de Junio de 2020). *National Science Foundation*. Obtenido de NSF, Where Discoveries Begin: https://www.nsf.gov/funding/pgm_summ.jsp?pims_id=503286

OpenJS Foundation. (25 de 02 de 2021). *OpenJS Foundation*. Obtenido de Node JS: https://nodejs.org/es/about/

Oracle. (2012). *Oracle9i Application Server Web Services Developer's Guide.* Austin: Oracle.

Oracle. (24 de 02 de 2020). *Java*. Obtenido de Java: https://www.java.com/es/about/whatis_java.jsp

Ortega, M. (2006). *Acelerómetro.* México: Monytex.

Parmar, P., & Champaneria, T. (2017). *Study and Comparison of Transportation System Architectures for Smart City.* Ahmedabad: I-SMAC.

Pautasso, C., Zimmermann, O., & Leymann, F. (2008). *RESTful Web Services vs. "Big" Web Services - Making the Right Architectural Decisions.* Beijing: Conference: Proceedings of the 17th International Conference on World Wide Web, WWW.

Perry, S. (5 de Noviembre de 2017). *IBM*. Obtenido de IBM: https://developer.ibm.com/es/languages/java/tutorials/j-spring-boot-basics-perry/

Petersen, K., Feldt, R., Mujtaba, S., & Mattsson, M. (2008). *Systematic mapping studies in software engineering.* Viena: ACM-Digital Library.

Pop, E., Gifu, D., Sacala , I., Caramihai, S., Moisescu, M., & Repta, D. (2019). *Services Integration for Cyber Physical Systems.* Bucarest: CSCS.

Proyectos con Arduino. (29 de Enero de 2019). *Proyectos con Arduino*. Obtenido de Proyectos con Arduino: https://proyectosconarduino.com/sensores/sensor-de-distancia-hc-sr04/

Quality Devs. (25 de Febrero de 2021). *Quality Devs*. Obtenido de Quality Devs: https://www.qualitydevs.com/2019/09/16/que-es-angular-y-para-que-sirve/

Raggett, D. (5 de Junio de 2016). *W3C*. Obtenido de Web of things interest group: https://www.w3.org/blog/wotig/2016/06/05/application-platforms-for-the-web-of-things/

Raiola Networks. (25 de Febrero de 2021). *Raiola Networks*. Obtenido de Raiola Networks: https://raiolanetworks.es/contacto/

RaspberriPi Foundation. (14 de Abril de 2015). *RaspberryPi*. Obtenido de RaspberryPi: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

Raspberry PI. (27 de Mayo de 2017). *Raspberry PI Foundation*. Obtenido de Raspberry PI Camera Modules: https://www.raspberrypi.org/documentation/hardware/camera/

Redacción TicJob. (14 de Enero de 2019). *MuyComputerPRO*. Obtenido de zona ticjob: https://www.muycomputerpro.com/zona-ticjob/competencia-ticjob-rest-vs-restful/

Richter, D., Konrad, M., Utecht, K., & Polze, A. (2017). *Highly-Available Applications on Unreliable Infrastructure; Microservice Architectures in Practice.* Berlin: IEEE International Conference on Software Quality.

Rodríguez, A. (27 de Noviembre de 2015). *Paradigma Digital, S.L*. Obtenido de Paradigma Digital: https://www.paradigmadigital.com/dev/quien-es-quien-en-la-arquitectura-de-microservicios-spring-cloud-22/

Rosa Moncayo, J. M. (17 de Mayo de 2018). *OpenWebinars*. Obtenido de OW: OpenWebinars: https://openwebinars.net/blog/que-es-rest-conoce-su-potencia/

RS Components. (1997). *Light Depending Resistor.* Corby: RS DATASHEET.

Ruby. (24 de 02 de 2020). *Ruby*. Obtenido de Ruby El Mejor amigo de un desarrollador: https://www.ruby-lang.org/es/about/

Sampaio, A., Rubin, J., Beschastnikh, I., & Rosa, N. (2019). *Improving microservice-based applications with runtime placement adaptation.* Recife: SpringerOpen.

Santana, C., Alencar, B., & Prazeres, C. (2018). *Microservices: A Mapping Study for Internet of Things Solutions.* Cambridge: IEEE International Symposium on Network Computing and Applications.

Santos, M., Hoque, S., & Magedanz, T. (2017). *A Service Orchestration Architecture for Fog-Enabled Infrastructures.* Berlin: FMEC.

Seeed. (15 de Marzo de 2017). *Seeed Studio Bazar*. Obtenido de Seeed The IoT Hardware Enabler: https://www.seeedstudio.com/ESP32-CAM-Development-Board-with-camer-p-3153.html

Shi, J., Wan, J., Yan, H., & Suo, H. (2009). *A survey of Cyber-Physical Systems.* Nanjing: 2009 IEEE International Conference on Intelligence and Security Informatics.

SL Componentes. (2017). *Sensor de Luminosidad AD4070.* 14: Agosto.

Soler&Palau. (15 de Octubre de 2018). *S&P*. Obtenido de S&P: https://www.solerpalau.com/es-es/blog/sensores-movimiento/

SpringBoot. (4 de Marzo de 2021). *Spring*. Obtenido de Spring: https://spring.io/projects/spring-boot

Taha, W. (2016 de Julio de 2016). *ResearchGate*. Obtenido de ResearchGat: Descubra el conocimiento científico y manténgase conectado con el mundo de la ciencia: https://www.researchgate.net/post/What-is-the-difference-between-cyber-physical-systems-CPS-and-Internet-of-Things-IoT-systems

Taherizadeh, S., Stankovski, V., & Grobelnik, M. (2018). *A Capillary Computing Architecture for Dynamic Internet of Things: Orchestration of Microservices from Edge Devices to Fog and Cloud Providers.* Basilea: Sensors.

Tao, F., & Qi, Q. (2019). *New IT Driven Service-Oriented Smart Manufacturing: Framework and Characteristics.* Beijing: SMACS.

The PHP Group. (24 de 02 de 2020). *My PHP.net*. Obtenido de PHP: https://www.php.net/manual/es/intro-whatcando.php

Thramboulidis, K., Vachtsevanou, D., & Solanos , A. (2018). *Cyber-Physical Microservices: An IoT-based Framework for Manufacturing Systems.* Patras: IEEE.

Toquica, J., Benavides, D., & Motta, J. (2019). *WEB COMPLIANT OPEN ARCHITECTURE FOR TELEOPERATION OF INDUSTRIAL ROBOTS.* Brasilia: CASE.

Trunov, A., Voronova, L., Voronov, V., Sukhachev, D., & Strelnikov, V. (2018). *Legacy Applications Model Integration to Support Scientific Experiment.* Moscú: IEEE.

Universidad Internacional de Valencia. (21 de Marzo de 2018). *VIU*. Obtenido de Tres tipos de seguridad informática que debes conocer: https://www.universidadviu.com/int/actualidad/nuestros-expertos/tres-tipos-de-seguridad-informatica-que-debes-conocer#:~:text=La%20seguridad%20de%20software%20se,proporcionar%20integridad%2C%20autenticaci%C3%B3n%20y%20disponibilidad.

UXpañol. (25 de 02 de 2017). *UXpañol - Discusiones sobre Experiencia de Usuario*. Obtenido de UXpañol: https://uxpanol.com/teoria/sistema-de-escalas-de-usabilidad-que-es-y-para-que-sirve/

Vujović, V., & Maksimović, M. (2015). *Raspberry Pi as a Sensor Web node for home automation.* Ámsterdam: ELSEVIER.

Web Empresa. (14 de 11 de 2015). *WebEmpresa*. Obtenido de WebEmpresa: https://www.webempresa.com/hosting/vps-que-es.html

Wikipedia. (2 de noviembre de 2020). *Fundación Wikimedia Inc.* Obtenido de Wikipedia, La enciclopedia libre: https://es.wikipedia.org/wiki/Tolerancia_frente_a_fallos

Wikipedia. (27 de noviembre de 2020). *Fundación Wikimedia Inc.* Obtenido de Wikipedia la enciclopedia libre: https://es.wikipedia.org/wiki/Netflix

Wikipedia. (21 de noviembre de 2020). *Fundación Wikimedia, Inc*. Obtenido de Wikipedia la enciclopedia libre: https://es.wikipedia.org/wiki/JSON

Wikipedia. (2020). *Transferencia de Estado Representacional.* Huntsville: Wikimedia Inc.

Wikipedia. (21 de 02 de 2021). *Wikimedia Inc*. Obtenido de Wikipedia la enciclopedia libre: https://es.wikipedia.org/wiki/Python

Wikipedia. (26 de Febrero de 2021). *Wikimedia Inc*. Obtenido de Wikipedia la enciclopedia libre: https://es.wikipedia.org/wiki/Dise%C3%B1o_de_tolerancia_a_fallos#Replicaci%C3%B3n

World Wide Web Consortium (W3C). (2007). *Simple Object Access Protocol (SOAP) 1.1.* Massachusetts: W3C Organization.

Yue, J., Wu, X., & Xue , Y. (2020). *Microservice Aging and Rejuvenation .* Xi'an: World Conference on Computing and Communication Technologies.

Zang, X., Zou, J., Huang, L., Chen, W., Zhou, S., & Liang , R. (2018). *A fault diagnosis method for microservices based on multi-factor self-adaptive heartbeat detection algorithm.* Beijing: IEEE.

Zheng, T., Zhang, Y., Zheng, X., Fu, M., & Liu, X. (2017). *BigVM: A Multi-Layer-Microservice-Based Platform for Deploying SaaS.* Melbourne: Fifth International Conference on Advanced Cloud and Big Data.

Anexos