ESCUELA PROFESIONAL DE INGENIERÍA DE SOFTWARE

ARQUITECTURA DE SOFTWARE BASADA EN MICROSERVICIOS PARA EL USO EN DISPOSITIVOS DE INTERNET DE LAS COSAS

Tesis presentada por:
Christian Víctor, Loza Peralta

Para optar por el Título Profesional de :
Ingeniero de Software

Arequipa - Perú

2020

Dedicatoria

A mis padres por todo el apoyo, comprensión y paciencia que me dieron durante mi vida.

Agradecimientos

Agradecer profundamente a todas aquellas personas que me apoyaron durante todo este tiempo, el cual fue muy valiosa su ayuda, sus consejos, su motivación, a todos ellos muchas gracias.

Índice general

Índice de Abreviaturas y Siglas

ACL Anti-Corruption Layer

API Application Programming Interface

ATAM Architecture Trade-off Analysis Method

AWS Amazon Web Service

BLE Bluetooth Low Energy

CI Continuous Integration

CoAP Constrained Application Protocol

CORBA Common Object Request Broker Architecture

CS Client Side

DDD Domain-Driven Design

DevOps Development Operations

ESB Enterprise Service Bus

GPS Global Positioning System

HTTP Hypertext Transfer Protocol

IoT Internet of Things

IP Internet Protocol

JPA Java Persistence API

JSON JavaScript Object Notation

JWT JSON Web Token

LAN Local Area Network

M2H Machine to Human

M2M Machine to Machine

MQTT Message Queuing Telemetry Transport

NFC Near Field Communication

OWL Web Ontology Language

RDF Resource Description Framework

REST REpresentational State Transfer

RFID Radio Frequency Identification

RMI Remote Method Invocation

RPC Remote Procedure Call

SMS Short Message Service

SOA Service Oriented Architecture

SOAP Simple Object Access Protocol

SS Server Side

TCP Transmission Control Protocol

TLS Transport Layer Security

URI Uniform Resource Identifier

WoO Web of Objects

XML eXtensible Markup Language

XMPP Extensible Messaging and Presence Protocol

Índice de Tablas

Índice de Figuras

Glosario de Términos

Arquitectura de software: Es una estructura organizativa de un sistema o componente.

Backlog: Lista de requisitos o tareas concernientes a determinados requisitos de usuario.

Diagrama: Es un conjunto finito de nodos y aristas que representa un concepto o idea.

Diseño arquitectónico: Es el proceso de definir una colección de componentes de hardware y software e interfaces para establecer un marco para el desarrollo de un sistema de software.

Dispositivo IoT: Es un microcontrolador Arduino que utiliza un conjunto de componentes electrónico e interactúa con dispositivos físicos por medio de sensores y actuadores.

Escenario: Espacio físico donde se ejecuta el dispositivo IoT e interactúa con uno o varios microservicios.

Hardware: Es el equipo físico compuesto de elementos electrónicos que es utilizado para el funcionamiento de un componente o un sistema.

Implementación: Es el proceso de traducir un diseño en componentes tangibles de hardware y software.

Internet de las Cosas: Es una interconexión entre objetos físicos e internet que permite la manipulación y control del objeto.

Microservicio: Es un conjunto de elementos que cumple como arquitectura en por medio de subconjuntos de componentes pequeños.

Middleware: Es un sistema que funciona como intermediario e intercambia información entre sistemas.

Tolerancia a fallos: Es un componente colocado para asegurar el funcionamiento de una función o componente y recuperarse de forma segura en caso de falla.

Virtualización o Contenerización: Es el conjunto de procedimientos para empaquetar un componente.

Resumen

El avance tecnológico es una de las brechas que existen en los diversos sectores de la sociedad que buscan adecuarse a la cuarta revolución industrial ó industria 4.0, esto implica una mayor demanda de sistemas de software que sean adaptables, flexibles y cambiantes en base a los objetivos de negocio; un punto crítico es el software que pueda ajustarse a la transformación digital y auge de dispositivos inteligentes con capacidades comunicativas a objetos físicos.

La arquitectura basada en microservicios es un tema relativamente nuevo aplicado a suplir las crecientes demandas de ciclos de software, por su granularidad, bajo acoplamiento y la minimiza la dependencia que puede tener con otros sistemas; por lo tanto lo hacen candidato para ser usados con lo dispositivos IoT y para nuevas necesidades tecnológicas emergentes que requieran cambio o adaptación.

El objetivo de la presente tesis es diseñar e implementar una arquitectura de software basada en microservicios para el uso en dispositivos IoT, el mismo que es ejecutado en tres escenarios de forma independiente que funcionan como servicios web. El software se encuentra hecho bajo la metodología ágil Scrum y diseñado bajo el enfoque de diseño dirigido por el dominio, implementado con las tecnologías de Jakarta EE, MySQL y Docker; en el caso de hardware se utiliza Arduino UNO y otros componentes electrónicos, permitiendo que el dispositivo pueda comunicarse con los servicios web, manteniendo alta cohesión entre los diversos dispositivos utilizados, servicios y protocolos de comunicación.

La arquitectura basada en microservicios es evaluada a través de los atributos de calidad tales como la modificabilidad, disponibilidad y portabilidad. Mientras que los dispositivos IoT son evaluados por medio de un conjunto de pruebas para garantizar la fiabilidad de su funcionamiento y comunicación, obteniendo resultados significativos.

Abstract

Technological advancement is one of the gaps that exist in many different sectors of society that seek to adapt to the fourth industrial revolution or revolution 4.0, this implies a greater demand for software systems that are adaptable, flexible and changeable based on business objectives; a critical point is the software that can adjust to the digital transformation and the rise of smart devices with communication capabilities to physical objects.

Micro services-based architecture is a relatively new topic applied to replace the increasing demands of software cycles, due to its granularity, low coupling and it minimizes dependence on other systems; therefore, they make it a candidate for use with IoT devices and for new emerging technological needs that require change or adaptation.

The objective of this thesis is to design and implement a software architecture based on microservices for use in IoT devices, which is executed in three independent scenarios that function as web services. The software is made under the agile Scrum methodology and designed under the domain-directed design approach, implemented with the technologies of Jakarta EE, MySQL and Docker; In the case of hardware, Arduino UNO and other electronic components are used, allowing the device to communicate with web services, maintaining high cohesion between the various devices used, services and communication protocols.

The architecture based on microservices is evaluated through quality attributes, and IoT devices are evaluated through a set of tests to guarantee the reliability of their operation and communication, obtaining significant results.

Palabras clave

Arquitectura de Software, Microservicios, Internet de la Cosas.

Introducción

El rápido avance de la digitalización, virtualización de la información y la demanda de sistemas adaptables a diferentes dominios como la industria, educación, agronomía y hogar, requiere una demanda de construcción de software flexible y adaptable a cambios que puedan estar interconectados digitalmente con dispositivos IoT.

La presente tesis propone un arquitectura de software basada en microservicios para el uso en dispositivo de Internet de las Cosas, que permitan el fácil acceso a la gestión de información por medio de servicios web y ser utilizados por diversos sistemas, de esta manera permitirá adaptarse a nuevos requerimientos y oportunidades emergentes que incrementen la productividad.

La tesis consta del capítulo I que describe la problemática del proyecto, se presentan los antecedentes investigativos y la definición del problema. El capítulo II muestra el planteamiento del proyecto, exponiendo los fundamentos teóricos, objetivo principal, objetivos específicos, justificación y viabilidad. En el tercer capítulo se hace una explicación de la metodología de desarrollo ágil escogida, el diseño e implementación de la arquitectura de software basada en microservicios y los dispositivos IoT con herramientas que permitan la integración y entrega continua. Seguidamente el capítulo IV muestra los resultados obtenidos de la arquitectura de software basada en microservicios el cual mediante la validación y análisis de arquitectura ATAM, proporciona la verificación y cumplimiento de los atributos de calidad de: modificabilidad, facilidad de instalación, portabilidad, tolerancia a fallos, disponibilidad y seguridad. Mientras que la comunicación con los dispositivos IoT en cada escenario propuesto tienen una fiabilidad de comunicación con los microservicios de 99 %, 82.96 % y 77 % respectivamente. Finalmente se exponen las conclusiones, recomendaciones y los enfoques de trabajos futuros relacionados con el tema de tesis.

Capítulo 1

Problemática del proyecto

1.1. Estrategia para la elaboración de los antecedentes investigativos

Para la construcción del estado del arte se ha considerado seguir un proceso que contribuya a obtener una mejor calidad de resultados con respecto al tema de la presente tesis, la figura 1.1 muestra el proceso de selección de papers.

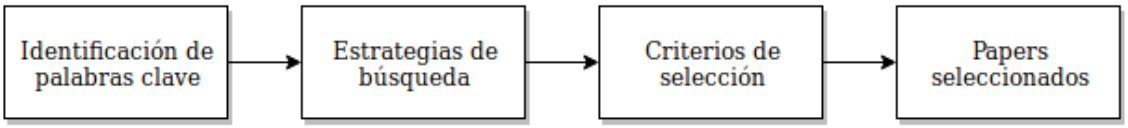


Figura 1.1. Proceso de selección de papers.

En la figura 1.1 muestra el proceso para la elaboración del estado de arte, inicia desde la identificación y selección de palabras clave, seguidamente se aplican las estrategias de búsqueda para explorar trabajos similares en distintas bases de datos científicas a los cuales puedan aplicarse los criterios de selección, obteniendo finalmente papers seleccionados con las consideraciones adecuadas para la presente tesis.

1.1.1. Cadena de Búsqueda y Estrategias de Búsqueda

Para la definición de la cadena de búsqueda se identifican las palabras claves que permite indexar documentos relacionados en conjunto con las estrategias de búsqueda:

((micro-service* OR microservice* OR "micro service"*) AND ("internet of things" or iot))

Las estrategias de búsqueda se añaden a la cadena de búsqueda para que el conjunto de documentos extraídos consultadas en bases de datos científicas sean más objetivas, estas estrategias se pueden denotar por los términos añadidos como son:

- **Operadores booleanos**: Permite la inclusión o exclusión de términos o palabras clave
- **Comillas**: Delimita un conjunto de palabras de forma literal.
- **Asterisco**: Utilizado como comodín para la extracción de contenidos formada desde una raíz.

Una vez delimitado la cadena de búsqueda en conjunto con las estrategias de búsqueda ya definidas se inserta en bases de datos científicas consideradas en el punto 1.1.2, posteriormente se emplea los criterios de inclusión, exclusión y eliminación para que proporcionen una mejor base de antecedentes investigativos del proyecto de investigación propuesto.

1.1.2. Bases de datos científicas

Se identifican las bases de datos científicas para proporcionar la búsqueda necesaria con respecto a la cadena de búsqueda ya definida, se consideran las siguientes bases de datos:

- Springer Link[1]
- IEEE Xplore Digital Library[2]
- Scopus[3]
- ScienceDirect[4]

1.1.3. Criterios de Inclusión

- Investigaciones publicadas con implementación de arquitecturas, *middlewares*, *frameworks*, modelos y técnicas con enfoque a microservicios e Internet of Things (IoT).
- Investigaciones publicadas únicamente en inglés.
- Investigaciones publicadas con un margen mínimo de 6 años de antigüedad máxima.
- Investigaciones publicadas en conferencias, congresos, *proceedings*, revistas y publicaciones en base de datos fiables expuestas en la sección 1.1.2.

1.1.4. Criterios de exclusión

- Publicaciones con metodologías aplicadas a microservicios e IoT.
- Investigaciones publicadas con propuestas de implementación de arquitecturas.

[1]https://link.springer.com/
[2]https://ieeexplore.ieee.org
[3]https://www.scopus.com
[4]https://www.sciencedirect.com

1.1.5. Criterios de eliminación

- Los criterios de eliminación se aplicaron a las investigaciones duplicadas en base al titulo, autor y año.
- Investigaciones con modelos matemáticos aplicados a microservicios y dispositivos IoT.
- Investigaciones que solo contengan una sola palabra del conjunto principal de la cadena de búsqueda.

En relación a la aplicación de criterios de selección se obtiene un total de 30 papers, considerando solo 11 papers que aplican intrínsecamente al uso de una arquitectura basada en microservicios y dispositivos IoT, de esta manera proporciona una mayor calidad a la presente investigación.

1.2. Antecedentes de la Investigación

Se realiza una estrategia para la elaboración de los antecedentes investigativos, que consta de: identificar las palabras clave, estrategias de búsqueda, criterios de selección y finalmente la selección de papers, este proceso se detalla.

1.2.1. Explotación de Microservicios interoperables en Objetos de la Web habilitados con Internet de las Cosas

La investigación de [1] menciona que todo objeto IoT tiene la capacidad de conectarse a internet, intercambiando información mediante la integración de aplicaciones en múltiples dominios con diferentes tecnologías de implementación, esta investigación propone el uso de una arquitectura basada en microservicios y una arquitectura Web of Objects (WoO) con una semántica interoperable que provee una plataforma de servicios y virtualización de objetos IoT, permitiendo la intercomunicación entre servicios y aplicaciones de IoT en dominios cruzados con gobernabilidad descentralizada como demuestra la figura 1.2.

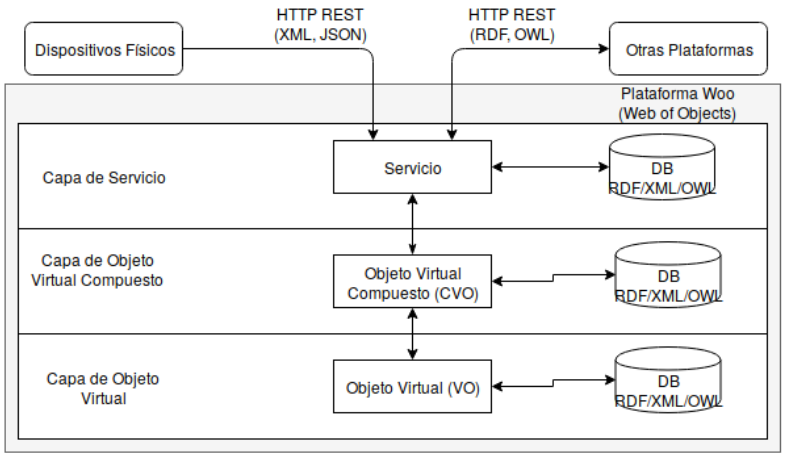


Figura 1.2. Arquitectura de la plataforma Web of Objects.

Fuente: [1].

La figura 1.2 muestra el uso de diferentes capas que tiene la plataforma WoO que los dispositivos IoT acceden e intercambian entre si. Cada capa de la arquitectura tiene un coordinador de mensaje único que permite la comunicación entre las otras capas, además poseen dos bases de datos: Resource Description Framework (RDF) similar a un modelo entidad relación combinado con eXtensible Markup Language (XML) y Web Ontology Language (OWL) que permite la publicación de datos mediante ontologías que controlan el registro histórico.
La Capa de Objeto Virtual se enfoca en la virtualización del dispositivo físico IoT, donde existen microservicios individuales por cada tarea que realiza este dispositivo. Este y otros dispositivos virtualizados son intervenidos por la capa de Objeto Virtual Compuesto que ejecuta características del servicio y por último la Capa de Servicio que proporciona una interfaz de acceso (autorización, autenticación, usuarios, descomposición de servicios, etc.) que controla mediante diferentes protocolos la comunicación con los dispositivos IoT.
La implementación de la investigación muestra el soporte de la arquitectura propuesta en diversos espacios inteligentes que reciben servicios de dispositivos IoT de terceros. Estos espacios inteligentes están dentro de dominios diferentes (vehículo, hogar, trabajo) que hacen posible intraoperabilidad e interoperabilidad entre dominios.

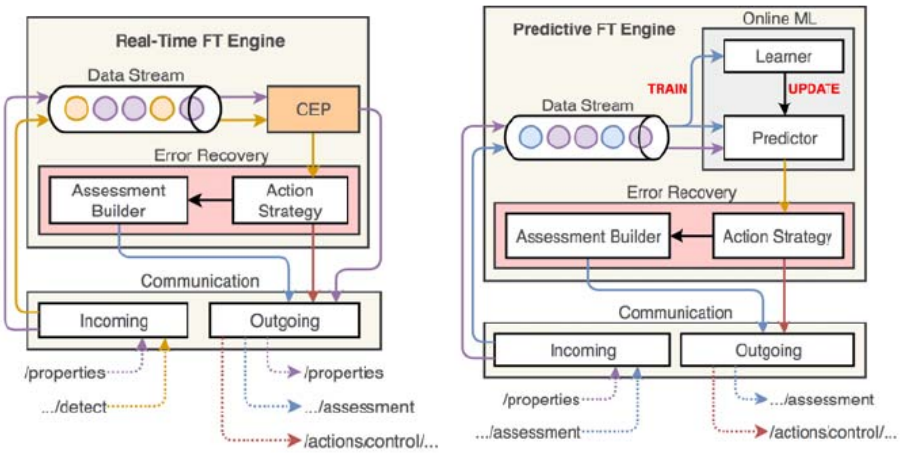
1.2.2. Arquitectura de microservicios para la tolerancia a errores reactiva y proactiva en sistemas de Internet de las Cosas

El autor [2] propone el uso de una arquitectura de soporte a tolerancia a fallos que pueda adaptarse y evolucionar con sistemas de dispositivos IoT, siendo posible mediante dos microservicios principales:

- **Microservicio de tolerancia a fallos reactiva**, inicia una estrategia de recuperación de errores después de que se ha detectado un error. Esto requiere una rápida detección y toma de decisiones, siendo posible mediante el uso de un microservicio auxiliar de procesamiento de eventos complejos (*complex event processing*), el cual analiza propiedades y errores.

- **Microservicio de tolerancia a fallos proactiva**, inicia una estrategia de recuperación de errores antes de que se haya detectado un error. El concepto esta diseño para prevenir fallas que afecten a una aplicación, es posible mediante el uso de un microservicio auxiliar de *Machine Learning*.

La comunicación de los microservicios propuestos se hace mediante el estilo arquitectónico RESTful intercambiando datos con formato JavaScript Object Notation (JSON) que es admitido por cualquier proveedor de servidor en la nube.
La figura 1.3 realizado por [2] muestra las arquitecturas utilizadas para tolerancia a fallos compuestos por diferentes microservicios para el manejo de flujo de datos y la recuperación de errores, las flechas de color púrpura indican el seguimiento de los datos, las amarillas detectan errores , las azules la evaluación del error y las rojas la acción de recuperación.



(a) Arquitectura tiempo real (b) Arquitectura predictiva

Figura 1.3. Arquitecturas de microservicios de tolerancia a fallos.

Fuente: [2].

La figura 1.3a muestra un componente *CEP (complex event processing)* funcionando un manejador inteligente que permite definir estrategias para recuperación basada en historiales, mientras que en la figura 1.3b muestra un microservicio auxiliar *Machine Learning* que permite predecir futuras fallas ambos funcionan en conjunto.

El caso de prueba realizada en este trabajo mostró que la arquitectura de microservicios a tiempo real detectó, evaluó e identificó cada error, estos errores son enviados a la arquitectura predictiva que evalúa e idéntica cada patrón.

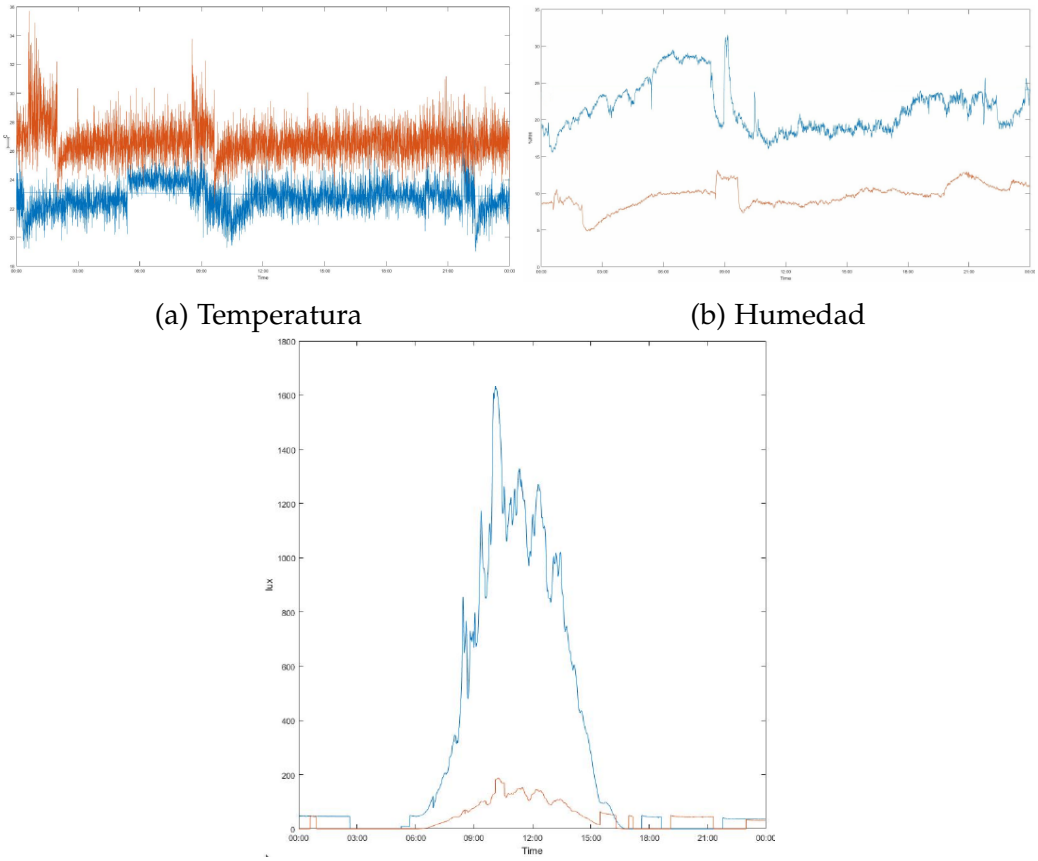
1.2.3. IoT basado en microservicios para edificios inteligentes

El trabajo de [3] se orienta en el diseño de un prototipo de una plataforma para soportar múltiples aplicaciones concurrentes en edificios inteligentes, que utiliza una red de sensores, así como una arquitectura de microservicios distribuida mediante el lenguaje de programación Jolie[5].

Para la construcción del prototipo se hizo el uso de sensores (temperatura, humedad, luminosidad y movimiento), cámaras, dispositivos RaspBerry[6] Pi con Bluetooth Low Energy (BLE) y el protocolo de comunicación Z-Wave para comunicación inalámbrica.

Estos dispositivos fueron colocados en diferentes habitaciones de la universidad de Innopolis (habitaciones de estudio, dormitorios y laboratorios) que están equipadas con artefactos eléctricos que proveen ventilación fría, ventilación caliente e iluminación del cual se utilizó para la recolección de datos y probar el prototipo, obteniendo los resultados mostrados en la figura 1.4.

[5]https://www.jolie-lang.org/
[6]https://www.raspberrypi.org/



(a) Temperatura (b) Humedad

(c) Luminosidad

Figura 1.4. Gráficos de comparación entre dos ambientes usando el prototipo y un sistema manual.

Fuente: [3].

Los resultados de comparación mostrados en la figura 1.4 demuestra que esta plataforma controla el flujo de gasto de energía según la cantidad de personas en las habitaciones. La figura 1.4a utiliza el prototipo el cual muestra que existe un menor uso de calefacción (color azul) que del sistema manual (color rojo). Mientras que en la figura 1.4b el prototipo mantiene la humedad de 30 % (color azul) en comparación al sistema manual (color rojo) y por último la figura 1.4c la comparación entre las dos habitaciones una de ellas no supera los 200 lux (color rojo) mientras que la otra habitación tiene un promedio de 600 lux (color azul) debido a la iluminación solar.
En la investigación destaca que el uso de dispositivos IoT y el uso de microservicios son base fundamental para los edificios inteligentes, que pueden ser capaces de adaptarse y autoconfigurarse según los factores y condiciones ambientales los cuales permiten el ahorro de energía.

1.2.4. Microservicios como agentes en sistemas IoT

La investigación propuesta por [4], muestra como objetivo el uso de microservicios en una arquitectura de agentes en sistemas IoT, estos dispositivos se comunican por medio del puerto de comunicación Machine to Machine (M2M) para el

intercambio de información mediante el uso de dispositivos físicos XBee[7], este último es utilizado como medio inalámbrico para la interconexión y comunicación entre dispositivos con protocolo IEEE 802.15.4. La implementación de la propuesta fue construida bajo una arquitectura monolítica con dos componentes principales: el primer componente es el núcleo principal regido bajo el modelo de capas (datos, lógica, presentación) y el segundo componente es la comunicación M2M como muestra la figura 1.5.
Posteriormente esta arquitectura fue descompuesta a nivel modular generando una arquitectura basado en agentes el cual tiene como base principal microservicios, el autor define que un agente es un microservicio el cual puede procesar solicitudes de otros microservicios mediante la comunicación M2M y protocolo Hypertext Transfer Protocol (HTTP), por lo que cada microservicio tiene su propia interfaz RESTful.

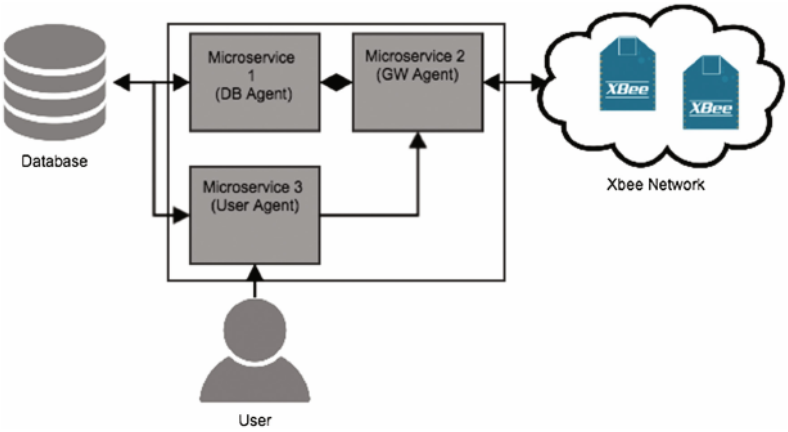


Figura 1.5. Arquitectura de microservicios como agentes para IoT.

Fuente: [4].

La figura 1.5 muestra los microservicios individuales donde son considerados como agentes con una responsabilidad única pero que trabajan de forma colaborativa, estos servicios están encargados de la siguientes tareas:

- **Microservicio 1 (*DB Agent*):** almacena los datos recibidos de una comunicación entrante.
- **Microservicio 2 (*Gateway Agent):*** registra los datos de comunicación al **Microservicio 1**, posteriormente descubre la red de dispositivos físicos XBee.
- **Microservicios 3 (*User Agent*):** este agente interactúa con el usuario final donde puede administrar los servicios y dispositivos.

Con la implementación de esta arquitectura se utiliza los microservicios para gestionar los dispositivos que bajo la comunicación M2M genera un ahorro de energía, así como la facilidad de controlar diferentes dispositivos IoT activos de forma remota.

[7]https://www.digi.com/xbee

Ambas arquitecturas (monolito y microservicios) son comparadas con respecto al dominio IoT como se muestra en la tabla 1.1.

Tabla 1.1. Propiedades de comparación entre arquitecturas de software.

Propiedad	**Arquitectura de Microservicios**	**Arquitectura Monolítica**
Escalabilidad	La replicación en servidores es dependiente de la demanda.	La replicación se hace en servidores múltiples.
Desarrollo	Uso de diferentes lenguajes de programación en el sistema.	Todos los servicios se escriben en un mismo lenguaje.
Cambiable	El cambio se hace en un microservicio con tiempos mínimos de despliegue del sistema.	El cambio causa revisión y demora en tiempo de despliegue del sistema.
Integración	Genera desafío cuando son grandes cantidades de módulos distribuidos.	Módulos en un mismo conjunto.
Comunicación	Se puede definir comunicación mediante la interfaz entre microservicios.	En memoria.
Mantenimiento	Pequeños módulos y código fuente modular.	Un solo código fuente para todo el sistema.
Actualización	Es posible agregar nuevos microservicios.	Actualización de todo el sistema.

Fuente: Elaboración propia.

La tabla 1.1 muestra las principales ventajas de la arquitectura basada en microservicios como: la escalabilidad, desarrollo, integración, comunicación, mantenimiento y actualización con respecto a dispositivos IoT.

1.2.5. Microservicios ciber físicos, un marco basado en IoT para sistemas de fabricación

La investigación realizada en [5] propone un *framework* llamado microservicios ciber físicos que aprovecha los beneficios de una arquitectura basado en microservicios en el dominio de sistemas de fabricación, utiliza la ingeniería impulsada por modelos para la semi automatización de microservicios y tecnologías IoT para el uso en la industria manufacturera. Además, este *framework* trabaja bajo el enfoque de modelo mecatrónico integrado que es considerado como una extensión de este sistema como se ve en la figura 1.6.

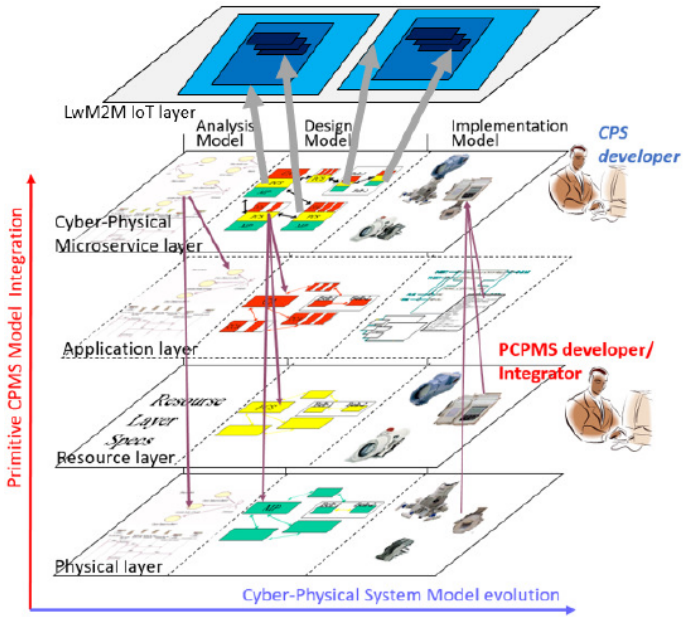


Figura 1.6. Arquitectura basada en microservicio y compatible con IoT para sistemas de fabricación ciberfísico.

Fuente: [5].

En la figura 1.6 muestra la evolución del modelo de sistema ciber físico (modelo de análisis, modelo de diseño e implementación) que se desarrolla en paralelo y en conjunto con la integración primitiva de microservicios ciber físicos en las diferentes capas.

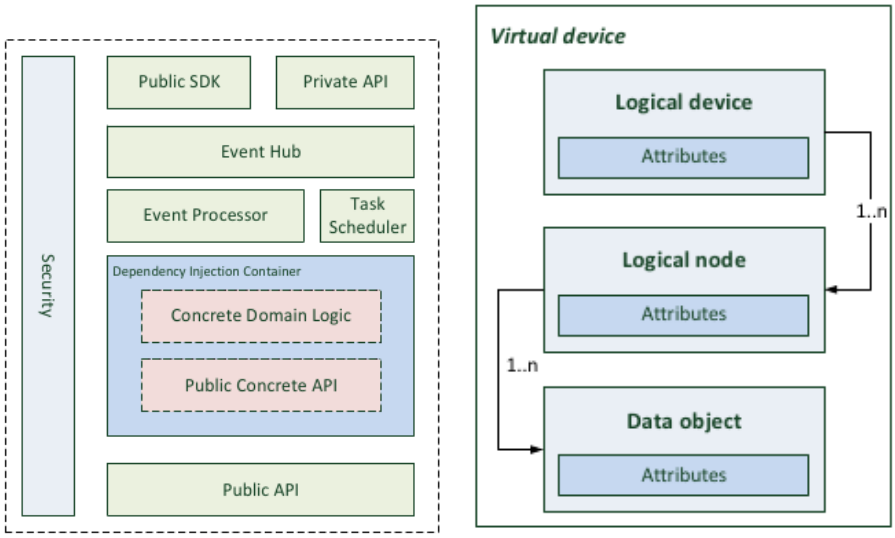
La principal capa de microservicios ciber físicos esta a cargo de dos desarrolladores (evolución del modelo de sistema ciber físico e integración primitiva de microservicios ciber físicos) que esta integrado estrechamente con la mecánica, la electrónica y el software con el objetivo de realizar funcionalidades específicas del entorno, teniendo así conexión con las demás capas (aplicación, recurso, física), pero principalmente con la capa IoT. Además, también existen interfaces de software, electrónicas y mecánicas definidas, lo que lo diferencia de los microservicios tradicionales del dominio de software, todas estas interfaces se definen utilizando software de construcción SysML[8] y AutomationML[9].

La investigación fue aplicada en dos casos de estudio, en una planta de licor y en un sistema de ensamblaje de sillas *Gregor*, ambos basados en el enfoque de microservicios ciber físicos, siendo necesario la aplicación de patrones de orquestación para definir cada proceso de ensamblaje. Las mediciones hechas en la investigación muestran una alta latencia en el nivel del componente ciber físico, el cual esta maquinaria inteligente no es aceptable en la industria, pero esta arquitectura basada en microservicios es una tecnología prometedora para el contexto *Industry 4.0* o también llamados sistemas de fabricación inteligentes, considerado clave para el desarrollo modular de sistemas flexibles.

[8]http://www.omgsysml.org/
[9]https://www.automationml.org/

1.2.6. Arquitectura de una plataforma IoT interoperable basada en microservicios

La investigación de [6] propone una arquitectura basada en microservicios como *middleware* que puedan tener conexión con diferentes tipos de dispositivos, servicios y protocolos de comunicación IoT de un dominio global, manteniendo así la escalabilidad e interoperabilidad de productos de software como también obtener un uso eficiente de los recursos IoT, la figura 1.7 explica la arquitectura.



(a) Microservicios a alto nivel (b) Dispositivo virtual

Figura 1.7. Arquitectura de microservicios e IoT en alto nivel.

Fuente: [6].

La figura 1.7 demuestra una arquitectura de microservicios 1.7a separado por capas donde se tiene expuesto una Application Programming Interface (API) que proporciona para la coordinación de servicios remotos y la virtualización del dispositivo IoT 1.7b, donde tiene diferentes componentes lógicos (dispositivo, nodo, objeto) con información sobre el dispositivo virtualizado que proporciona estados, descripción, datos y recursos del modelo.
La comunicación entre los microservicios y el dispositivo físico virtualizado es posible mediante una interfaz REST que permite la gestión de datos con el uso de la comunicación M2M.
Durante la experimentación de la investigación los microservicios pudieron ser probados en escenarios de automatización del hogar en los siguientes puntos:

- **Microservicio meteorológico:** Encargado de la adquisición de datos de microclima, historia de datos de servicio obtenido de la medición de temperatura ambiental que frente al microservicio Modbus TCP implementado, que se centra en la implementación de la pila de protocolos de comunicación Modbus, generando la adquisición de datos del microclima para una ubicación geográfica. preconfigurada.

- **Microservicio *Machine Learning*:** Este microservicio esta encargado de estimar el consumo de energía en el futuro mediante el uso de una regresión lineal, dando como resultante un modelo de regresión entrenado que se puede usar para realizar predicciones.

El trabajo enfatiza la necesidad de conectar dispositivos heterogéneos dentro de un entorno global único teniendo en cuenta la escalabilidad de las aplicaciones es un requisito previo para el uso eficiente de los recursos de IoT. Además, la arquitectura propuesta basada en microservicios mantiene principios de protocolos de comunicación, conservando la escalabilidad del sistema.

1.2.7. Diseño de una plataforma inteligente IoT con arquitectura de microservicios

El trabajo propuesto por [7] muestra la implementación de una arquitectura de microservicios con IoT aplicado a una plataforma *smart city* para diferentes aplicaciones involucradas en el aumento de la eficiencia de una ciudad a nivel distrital. Esta plataforma opera bajo el sistema de Modelado de Información de Distrito para Reducción de Energía (DIMMER), el cual analiza la información a tiempo real de sensores y comentarios de los usuarios para analizar la utilización de los edificios y proporcionar retroalimentación en tiempo real sobre el uso de la energía y comportamientos relacionados.

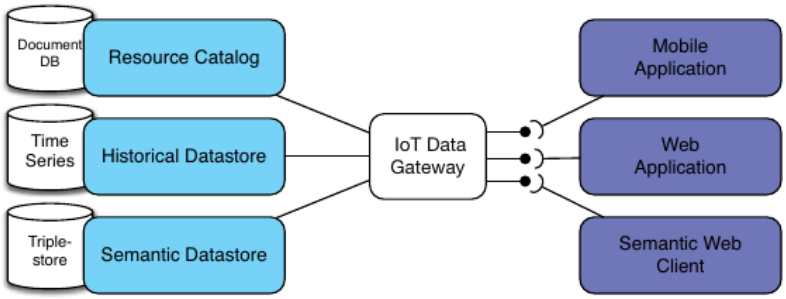


Figura 1.8. Arquitectura de microservicios, parte de la plataforma DIMMER.

Fuente: [7]

La figura 1.8 muestra la arquitectura de microservicios el cual tiene comunicación mediante una API que tiene conexión con dispositivos IoT, esta API tiene una interfaz de comunicación el cual permite que diversas plataformas(Móvil, aplicación Web, Web semántica) obtienen información. Cada microservicio esta a cargo de:

- ***Resource Catalog***: Encargado de proporcionar información básica sobre la configuración de los dispositivos, la implementación y los protocolos de comunicación compatibles.
- ***Historial Datastore***: A cargo del almacenamiento de datos del sensor, mensajería y otros.

- ***Semantic Datastore***: Es una abstracción de nivel superior, que describe los dispositivos IoT con atributos y relaciones con otras entidades y servicios utilizando las tecnologías de la Web Semántica con lenguaje SPARQL.

Esta información es registrada en las bases de datos de cada microservicio, después es consumida mediante peticiones HTTP, que sirve como *middleware* para el sistema DIMMER, de esta manera contribuye a trabajar de forma independiente y descentralizada, permitiendo que cada microservicio se concentre en su trabajo mientras mantiene la compatibilidad general del sistema. Además, permite que cada interfaz de servicio, pueda conectarse con otras implementaciones.
La investigación concluye que el uso de microservicios simplifica el diseño y la implementación de servicios individuales, pero tiene el costo de una mayor complejidad de los sistemas distribuidos. Sin embargo, a medida que se desarrollen más aplicaciones y servicios, se debe llevar a cabo una evaluación más exhaustiva.

1.2.8. Enfoque de microservicios para Internet de las Cosas

Esta investigación propuesta por [8] proporciona una descripción y comparación de diseño, patrones y mejores prácticas utilizadas bajo el enfoque de microservicios con IoT, demostrado en la tabla 1.2. Estos últimos han generado mucha demanda y se debe considerar su funcionamiento de una manera adecuada, no obstante las aplicaciones de IoT podrían adoptar varias de estas decisiones de diseño para mejorar la capacidad de crear aplicaciones de valor agregado.

Tabla 1.2. Tabla de comparación de características entre microservicio e IoT.

Característica	Microservicios	IoT
Auto contención	Envuelve el dominio de negocios y reduce las dependencias, bibliotecas empaquetadas con la aplicación.	Bibliotecas no empaquetadas con aplicaciones.
Coreografía vs Orquestación	Coreografía preferente para una mejor gobernabilidad central.	Orquestación preferente, cuando se usa HTTP.
virtualización de contenedores	Permitido mediante Docker, lxc, permite la escalabilidad y despliegue rápido	No tiene alguno.
Integración continua	Contiene integración continua, permite detectar fallos	No existe.
Entrega continua	Contiene entraga continua en pequeños ciclos	Existe parcialmente, de acuerdo al escenario.
Protocolo de comunicación	HTTP	HTTP, MQTT, CoAP

Fuente: Elaboración propia.

La tabla 1.2 expuesta por [8] muestra algunas similitudes que aborda los microservicios y los dispositivos IoT, asimismo que este tipo de de enfoque combinado tiene una capacidad de respuesta más ágil frente a arquitecturas monolíticas.

1.2.9. Un estudio de mapeo sobre arquitecturas de Microservicios de IoT y soluciones de Computación en la Nube

El mapeo de arquitecturas de microservicios de internet de las cosas con soluciones *cloud* muestra el incremento de este tipo de colaboración de tecnologías desde el año 2016 como afirma [9], de esta manera este trabajo de investigación tiene el objetivo claro de hacer un mapeo sistemático con el contexto de *microservices, IoT y Cloud Computing*, generando tres (03) preguntas de investigación:

- **RQ1**: ¿Cuántas publicaciones por año se encuentran en el área de investigación?
- **RQ2**: ¿Cuáles son los principales lugares para las publicaciones del área de investigación?
- **RQ3**: ¿Cuáles son los principales tipos de publicaciones en el área de investigación?

Esta investigación define una cadena de búsqueda con el contexto definido anteriormente y es sometido a filtros que el autor considera en base a los resultados existiendo un crecimiento investigaciones publicadas, como se demuestra en la figura 1.9.

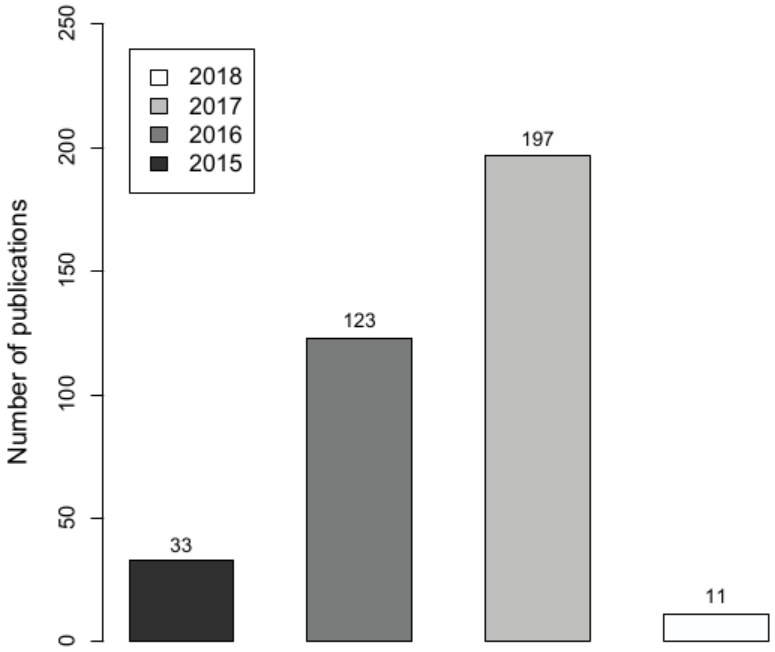


Figura 1.9. Distribución de publicaciones por año.

Fuente: [9].

La figura 1.9 muestra el crecimiento de investigaciones del tema propuesto, además responde a las preguntas de investigación:

- **RQ1**: Inicia con 23 publicaciones en el 2015, 123 en el 2016, 197 en el 2017 y 18 hasta en el 2018.
- **RQ2**: la Investigación indica que existen 364 publicaciones en diversos lugares, pero que existen más publicaciones en *Symposium on Service-Oriented System Engineering* con 15 publicaciones, *The International Conference on Cloud Computing* con 11 publicaciones y *International Symposium on Cluster, Cloud and Grid Computing* 10 publicaciones
- **RQ3**: Existe 364 publicaciones, 329 publicadas en conferencias y 15 publicaciones en revistas.

El autor concluye que este tipo de paradigma es relativamente nuevo, que recibe atención del mundo empresarial, ya que esta siendo adoptada por grandes empresas (Netflix, eBay, Amazon) ofreciendo flexibilidad para la adaptación a modelos de negocio, así como las ventajas que tiene en el desarrollo, implementación, despliegue independiente y viabilidad con IoT, además que existe un creciente número de investigaciones con respecto a este tema.

1.2.10. Un sistema de puerta de enlace inteligente IoT basado en microservicios

La investigación expuesta por [10] muestra un diseño basado en microservicios para el acceso a IoT interconectado con sensores mediante el uso de la nube, este diseño cuenta con una base de datos adicional para recolectar datos telemétricos de sensores. Este modelo se muestra en la figura 1.10 que cuenta con diversos módulos escalables que interactúan con los demás módulos.

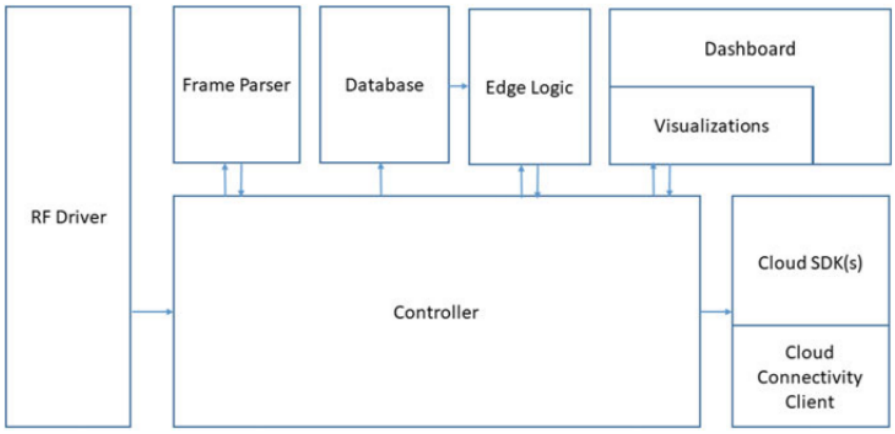


Figura 1.10. Arquitectura basada en microservicio con flujos de datos.

Fuente: [10].

La figura 1.10 muestra los diversos módulos hechos bajo una arquitectura basada en microservicios, cada uno de estos módulos con una funcionalidad diferente implementado por [10], estos módulos tienen las siguientes funciones específicas:

- ***Controller***: Módulo que permite la interacción y coordinación con otros módulos.
- ***RF Driver***: Módulo encargado de la transmisión de datos de los sensores, así como también la configuración de la red IoT todas estas tareas se llevan a cabo de manera asíncrona.
- ***Frame Parser***: Este módulo esta encargado del recibir los datos del módulo *Controller* para analizar las tramas de cada paquete, de acuerdo a la demanda del módulo *RF Driver*.
- ***Edge-Logic***: Módulo con dos importantes funciones que proporciona servicios hacía la niebla de los sensores, primero envía mediante un proceso realizar un análisis de los datos ya transmitidos y un segundo que obtiene las características principales de los datos.
- ***Database***: Almacena los datos analizados del módulo *Edge-Logic*.
- ***Cloud SDK y Cloud Connectivity Client*** : Los datos transferidos por el *Controller* que son datos de las telemetrías generadas por sensores son enviadas a la nube, trabaja en conjunto con el módulo cliente con conectividad a la nube.
- ***Dashboard y Visualiations***: Módulo que se conecta mediante un servidor HTTP, provee una interfaz de usuario, provee además el monitoreo de todos los dispositivos conectados, alertas, mensajes.

Esta investigación fue utilizada para el monitoreo de salud estructural, el cual controla la seguridad, la integridad y el rendimiento mediante la colocación de sensores dentro de una estructura para detectar la presencia de algún daño generado de forma natural o artificial.

1.2.11. Discusión de los antecedentes investigativos

Anteriormente se describieron varias investigaciones que aplican el uso de microservicios con dispositivos IoT, este último asimilado como una tecnología emergente que posee las capacidades de monitoreo, detección, control y servicios a usuarios o a diversos sistemas. Los microservicios demuestran ser una solución con capacidad de adaptación y sustitución frente a cambios que requerían una interconexión de objetos físicos [1], [2], [3], [7], [8].
Estos dispositivos usados en las investigaciones son implementados con propósitos específicos aplicados a diversas incidencias presentadas, demostrando su funcionalidad en la comunicación con los objetos físicos. No obstante, debe tomarse en cuenta que en algunas investigaciones existe una limitación en la comunicación utilizando M2M que reduce la flexibilidad de los dispositivos, otras investigaciones demuestran que existen un conjunto de microservicios que están intrínsecamente conectados, que puede interpretarse como un sistema monolito distribuido y finalmente ciertas investigaciones utilizan diversos microservicios pero una misma base de datos generando una inconsistencia cuando se requiera

realizar cambios significativos [4], [5], [6], [10]. Por lo tanto se considera oportuno el diseñar e implementar una arquitectura basada en microservicios para evaluar su funcionamiento en escenarios específicos usados por los dispositivos IoT, de tal forma demostrar que puede emplearse en distintos sectores de la sociedad.

1.3. Descripción del problema

Los constantes cambios de la tecnología producen adaptaciones de los diversos dominios conocidos de la sociedad, estos cambios se ven reflejados en nuevas tendencias tecnológicas que proporcionan métodos para transformar todo proceso operacional en beneficio económico, mejora de servicio, eficiencia operativa con capacidad de gestionar información de manera rápida y accesible.
En este contexto, el internet se ha convertido en una herramienta esencial para este avance tecnológico, el incremento de usuarios que acceden a internet mediante el uso de teléfonos inteligentes y planes móviles cada vez más asequibles, es notable; según [11], podemos detallar ese aumento:

- El número de usuarios conectados a internet en el 2020 es de 4.54 mil millones, incrementando 298 millones (7.00 %) de nuevos usuarios adicionales con respecto al año 2019.
- El número de usuarios que utilizan dispositivos móviles en el 2020 es de 5.19 mil millones, incrementando 124 millones (2.40 %) más que el año 2019.

El Perú no es ajeno a esta realidad, tal como se menciona en [12], el uso de internet en el hogar es de 28 % y en telefonía móvil 83.8 % en el 2018; en el sector empresarial en el 2015 es de 88.5 %, 94.3 % en telefonía móvil y 91.3 % en computadoras personales. Así mismo, la utilización de software licenciado es de 36,9 % mientras que el llamado "software libre" 17.60 %. En cuanto a la construcción de software para resolver necesidades específicas el 10.90 % [13].
Las organizaciones buscan constantemente el aumento de su productividad, alguno de los factores que permitirían este incremento son, la automatización de procedimientos y procesos de negocio, que por medio del uso de aplicaciones e interacción con objetos físicos, permitirían un mayor control y conocimiento del entorno. Sin embargo, la continua demanda de productos de software, requieren la adaptación de requerimientos de negocios cambiantes, buscando el aumento del valor de diferentes dominios y la transformación de procesos operacionales que generen beneficios comerciales de calidad a través de la eficiencia operativa y productiva [14].
Así mismo, el intercambio de información con diferentes dispositivos IoT en el que se transfiere datos generados de los objetos hacia los teléfonos inteligentes, a los sistemas de información, sistemas de recomendación, sistemas de predicción y otros dispositivos de la misma índole que puedan comunicarse mutuamente de forma independiente, con gran escalabilidad, objetivo comercial definido e incremento de la flexibilidad y agilidad en el desarrollo, es fundamental.
Todo software de aplicación o de sistemas, es el nodo central del intercambio de información, la mayoría de ellos tienen módulos que no pueden ejecutarse de

forma independiente, teniendo como consecuencia una deficiencia con la interoperabilidad con múltiples proveedores de soluciones emergentes, incremento de dependencias en base a la agregación de bibliotecas, limitación de escalabilidad y configuraciones requeridas [8].
Por otro lado, este tipo de arquitecturas definidas, tienen limitaciones en cuanto a la ejecución de diversos tipos de pruebas, tiempo de compilación de los diversos módulos y en el despliegue del producto de software; estos acontecimientos se producen periódicamente cuando se añaden nuevas funcionalidades, convirtiéndose en un punto crítico para la productividad en las diversas fases de desarrollo del software [15].
Una arquitectura basada en microservicios proporciona flexibilidad, escalabilidad y responsabilidad única que pueden aplicarse como:

- Una solución de inconvenientes al ser utilizado como *Middleware* entre dispositivos IoT y sistemas demasiado grandes o complejos.
- Una propuesta para el desarrollo de productos de software con el objetivo de integrar dispositivos IoT.

Adicionalmente, este tipo de arquitectura nos ofrece versatilidad y adaptación a cambios e innovación constantes de un dominio especifico, permitiendo que cada microservicio pueda ser ejecutado de forma granular, desarrollándose de forma autónoma con la especificación de un lenguaje de programación o de forma políglota según lo requiera las necesidades presentadas, generando despliegues independientes y datos descentralizados.
Es debido a esto, que los microservicios suplen las crecientes demandas generadas en el ciclo de desarrollo de software de manera más ágil; este tipo de arquitectura, basada en microservicios, satisface de manera óptima, las necesidades tecnológicas emergentes que requieran cambios constantes, actualizaciones y mantenimiento. De esta manera, se logra una ventaja significativa frente a una arquitectura monolítica tradicional, que no sustituye en cierta medida la inmersión del uso de nuevos dispositivos tecnológicos y objetos inteligentes que pueden proveer información necesaria.

Capítulo 2

Planteamiento del Proyecto

2.1. Fundamentos teóricos

2.1.1. Ingeniería de Software

La ingeniería de software esta destinada a apoyar al desarrollo de software de manera profesional, con el objetivo de mejorar la creación de sistemas de software que satisfagan las necesidades de los clientes usando técnicas de requisitos, especificación, diseño y evolución de un programa, de modo que la ingeniería de software es una disciplina que se ocupa de todos los aspectos de la producción de software con alta calidad, desde la concepción inicial hasta la operación y el mantenimiento [16], [17].
La definición de [18] sobre ingeniería de software es descrita como el diseño y desarrollo sistemático de productos de software así como la gestión del proceso de software, con el objetivo principal de producir software de calidad que cumplan con las especificaciones bajo un marco de precisión en tiempo y presupuesto. Mientras que para [19] es una aplicación sistemática, cuantificable y disciplinado para el desarrollo, operación y mantenimiento de software.
Para [20] la ingeniería de software se define como un proceso de creación de un producto de software que utiliza una metodología definida que contribuye a crear los artefactos de software y el proceso que ayuda a crear estos artefactos, por lo tanto, la metodología de ingeniería de software es el factor más importante que influye en un proyecto para construir un producto de software que incluye la ingeniería de requisitos, diseño de software, construcción de software, pruebas, etc.

2.1.1.1. Procesos de la Ingeniería de Software

El proceso de ingeniería de software esta conformada por actividades relacionadas entre si, generando un producto de software de calidad [16], [17], [21]. Estos procesos son considerados estándar para el desarrollo [22]. Existen diferentes métodos para aplicar en la ingeniería de software según el tipo de software y requisitos del cliente, pero generalmente siguen los mismos procesos como se muestra en la figura 2.1.

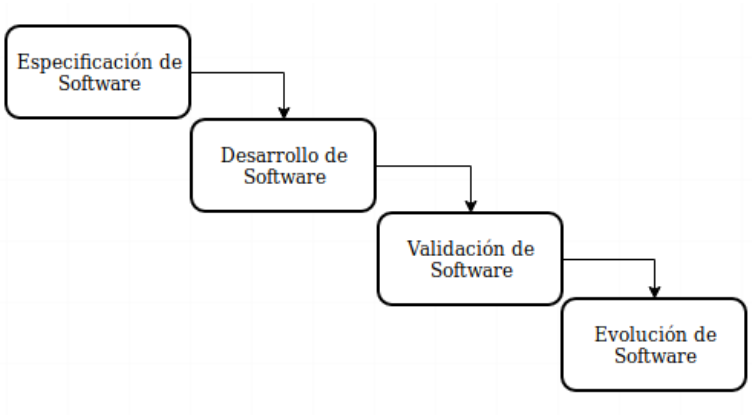


Figura 2.1. Procesos de la ingeniería de software.

Fuente: [16].

La figura 2.1 muestra los cuatro procesos de manera general seguidamente detallados por [16]:

- **Especificación del Software**: Proceso donde se define la funcionalidad del software, restricciones de funcionamiento y requisitos.
- **Desarrollo del Software**: Es el proceso donde se desarrolla el diseño e implementación del software cumpliendo con la especificación.
- **Validación del Software**: Es el proceso donde se comprueba la funcionalidad con enfoque a los requerimientos establecidos.
- **Evolución del Software**: Proceso donde el software evoluciona para satisfacer las necesidades cambiantes de los clientes, añadiendo nuevas capacidades y funcionalidades.

2.1.1.2. Modelos de Ingeniería de Software

La construcción de software tiene como dependencia la adopción de modelos según las necesidades de las partes interesadas con la finalidad de obtener resultados óptimos, estos modelos están categorizados en dos partes: el primero, un modelo tradicional (componentes, iterativos y cascada) y el segundo es el enfoque ágil basado en un modelo iterativo-incremental [16], [17], [20], [21].
Estos modelos y procesos de software evolucionan según lo requieran los modelos de negocios y el tiempo, mostrando que los ciclos de vida estrictamente secuenciales no son suficientes para adaptar las nuevas tendencias industriales y tecnológicas listados por [19]:

- Digitalización y sistemas cibernéticos.
- Soluciones *Cloud*.
- Soluciones orientadas a servicios.
- Fuerte vinculación entre desarrollo y operación (despliegue continuo, integración continua, Development Operations (DevOps)).

- La importancia de la seguridad en diferentes entornos con efecto de globalización.
- La demanda de productos listos para usarse con configuraciones y servicios.
- El incremento del uso de teléfonos móviles y aplicaciones.

2.1.1.3. Metodología de Desarrollo Ágil

La reducción de trabajo ligado a métodos tradicionales es uno de los motivos de que algunos proyectos fueron adoptando metodologías ágiles, reduciendo los ciclos de desarrollo de manera iterativa, minimizando el tamaño de equipos, participación constante de los clientes y crear un trabajo demostrable en cada iteración para llegar al producto final [21]. Estas metodologías ágiles dependen mucho de prototipos rápidos que cumplan con el objetivo de entregar software útil rápidamente, sin la necesidad de especificar a detalle el sistema y generando documentos al mínimo [16], [23]. Además, estas metodologías se definen por el manifiesto ágil que valora los siguientes puntos como lo definen [24], [25]:

- **Individuos e Interacción**: Enfoque en los procesos y herramientas.
- **Software Funcional**: Generar una medida de progreso más que documentación.
- **Colaboración con las partes interesadas**: Entender las necesidades del cliente que un contrato definido.
- **Respuesta al cambio**: Adaptarse a la mutabilidad necesaria que seguir el plan.

A. Scrum: El autor en [17] denota que es una metodología ágil desarrollada a inicio del año 1990, el cual tiene como prioridad seguir un conjunto de actividades, definiendo tareas y ejecutándolas para el éxito del producto de software, siendo posible la asignación de estas tareas en pequeños equipos para trabajar de manera autónoma. Esta metodología esta más centrada en el producto que en el proceso en comparación con las metodologías tradicionales [23]. Scrum también es una de las metodologías ágiles más amigables para ser usados en gestión de proyectos ya que tiene una visión más operativa del software, planificación flexible y distribución de trabajo en base a cada incremento llamado *Sprint* [21]. Esta metodología conforme a [16] puede integrarse rápidamente en cualquier empresa de manera práctica y que estos pueden escalar a sistemas grandes, ya que cuenta con etapas propias como se presenta en la figura 2.2.

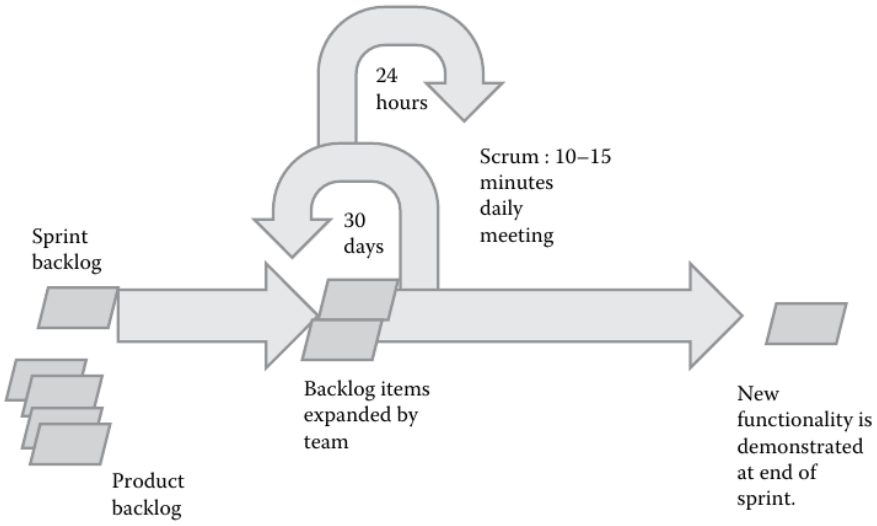


Figura 2.2. Etapas de Scrum.

Fuente: [23].

La figura 2.2 muestra las etapas de Scrum que tiene un orden definido para la ejecución, en cada etapa se generan documentos que el equipo debe cumplir según la asignación de roles. Estos elementos se conforman de la siguiente manera como definen [16], [24], [25]:

Roles

- ***Scrum Master***: Es el responsable de asegurar todo el proceso Scrum cumpla con los objetivos estipulados, solucionando problemas técnicos, externos e internos. Este rol no debe confundirse como un jefe de proyecto.
- ***Product Owner***: Tiene la responsabilidad de identificar, priorizar, revisar y desarrollar cada requisito del producto, satisfaciendo las necesidades principales requeridas por el dominio, este rol puede ser directamente un cliente o algún representante de las partes interesadas.
- ***Development team***: Es el equipo de desarrolladores con la única responsabilidad de la codificación del software y generación de documentos esenciales del proyecto.

Además de estos roles, es necesario un **Ingeniero de Calidad** como responsable de la gestión de la calidad de los equipos, el establecimiento de las mejores practicas y estándares necesarios para el proyecto [25]. Y un arquitecto de software como líder técnico para gestionar el diseño y código [26]. Ambos roles establecen una comunicación constante con el ***Scrum Master*** y ***Product Owner***.

Etapas

- ***Scrum***: Es la reunión diaria realizada por el equipo para determinar el avance del proyecto, normalmente estas reuniones tienen un tiempo entre $10-15$ minutos.
- ***Sprint planning***: Es la etapa inicial encargado de realizar las planificaciones para proceder con el *Sprint*.
- ***Sprint***: Es el tiempo de duración del trabajo limitado por una cantidad de días para evaluar los resultados, esta etapa se realiza de manera iterativa.
- ***Sprint preview***: Es la revisión y presentación mediante la demostración de un mínimo viable hacía los interesados.
- ***Sprint retrospective***: Es la etapa del análisis del Sprint que verifica errores e inspecciona problemas desarrollados para proceder a una mejora continua.

Documentos

- ***Product Backlog***: Es el documento que tiene una lista de elementos que debe "hacer" el equipo, este documento contiene funcionalidades, requisitos de software, historias de usuario y documentación sobre la definición de la arquitectura.
- ***Sprint Backlog***: Es un conjunto reducido de ítems extraído de la lista del *Product Backlog* que deben ser desarrolladas durante la ejecución del *Sprint*.

Además de estos elementos se considera: la **velocidad** como un factor importante para la comprensión de respuesta del equipo durante la ejecución del *Sprint* con el objetivo de proporcionar una base para la estimación de rendimiento y el **Producto entregable incremental** el cual es la finalización completa de un sprint sin requerir cambios añadiendo al producto final.

2.1.1.4. Ciclo de vida del Software

El ciclo de vida del software es una representación simplificada del proceso de software de acuerdo a las necesidades de desarrollo, requisitos del cliente, entorno y uso [16]. De esta manera un proyecto de software se gestiona de una forma ordenada con cada actividad interrelacionada de manera lógica y temporal [17]. Estas actividades como se muestra en la figura 2.3 pueden ejecutarse de manera simultanea para producir una salida compartida, ya sean interfaces de usuario o diferentes componentes desarrollados según la cantidad de equipos y tareas asignadas [21].

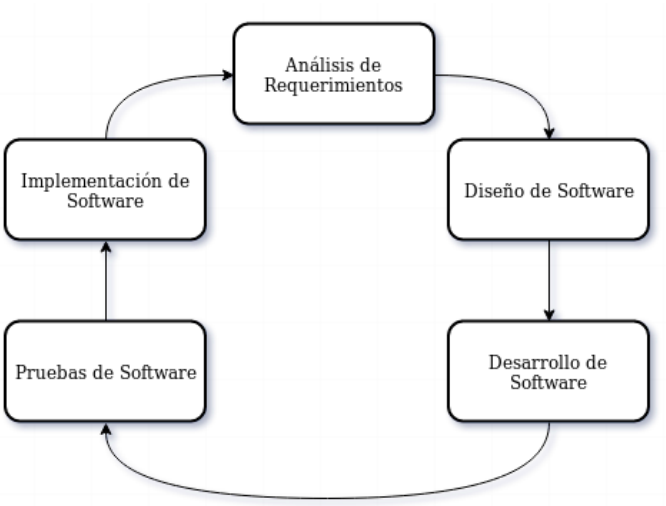


Figura 2.3. Actividades en el ciclo de vida de un proyecto de desarrollo de software.

Fuente: [18].

La imagen 2.3 muestra cada actividad desarrollada en la vida de un proyecto de software cada una con los siguientes objetivos definido por [18]:

- **Análisis de requisitos**: Actividad donde se especifican la producción, restricción, comportamiento y requisitos funcionales.
- **Diseño de software**: Es el conjunto de componentes, artefactos y aplicación de métodos que cumplan los objetivos del proyecto en base a los requisitos.
- **Desarrollo de software**: Es la representación del diseño en código fuente.
- **Pruebas de software**: Actividad de generación de pruebas al desarrollo de software, verificando que cumpla las expectativas del cliente.
- **Implantación de software**: Actividad final donde el cliente realiza la aceptación del producto de software y es desplegado.

2.1.2. Arquitectura de Software

Una arquitectura de software es un conjunto de estructuras necesarias como base para la construcción del software [27]. Esta estructura tiene elementos (componentes y conexiones) relacionados entre si que apoyan el adecuado comportamiento para un sistema de software [17], [18], [28], [29]. Además, contiene vistas que son una representación de varias estructuras diferenciados en tres tipos [22]:

- **Módulos**: Estructuras con el enfoque de organizar un conjunto de unidades de códigos construidos.
- **Componentes y Conectores**: Estructuras que definen los conjuntos en tiempo de ejecución (componente) e interacción (conector).
- **Asignación**: Estructuras relacionadas con enfoques externos al software.

Estas estructuras son puntos claves para la definición y construcción de la arquitectura de software ya que deben aplicarse los **atributos de calidad** con el propósito de optimizar la construcción de software, toda arquitectura no puede definirse si es apta o no, ya que puede tener diversos propósitos [30]. Uno de estos propósitos es que la arquitectura pueda soportar el ciclo de vida que requiere el software [29].

2.1.2.1. Atributos de calidad de una arquitectura de software

Una arquitectura de software debe estar compuesto por los atributos de calidad, un atributo de calidad es medible y comprobable [22]. El autor en [30] define que una arquitectura de software es aquella que puede lograr los atributos de calidad de software. Estos atributos definen la calidad del sistema de software en base a sus propiedades y requisitos no funcionales [31]. Se consideran los principales atributos de calidad afirmados por [22] y[32] :

- **Disponibilidad**: Se refiere a la prevención, detección y recuperación que debe tener el sistema sin alguna interrupción durante un tiempo específico, recuperándose sin afectar alguna estructura o componente.
- **Interoperabilidad**: Es el intercambio significativo de información con otros sistemas, de esta manera el sistema obtiene una retroalimentación.
- **Modificabilidad**: Se enfoca en el impacto que conlleva al realizar cambios, estos cambios pueden afectar las variables de tiempo y costo.
- **Rendimiento**: Este atributo tiene el objetivo de verificar el tiempo de respuesta del sistema, capacidad y cantidad de respuesta en base a eventos.
- **Seguridad**: Encargado de la protección del sistema y proveer el uso no autorizado de algún módulo o servicio, estos deben cumplir con las garantías de: no repudio, confidencialidad, integridad, disponibilidad, autenticación y autorización.
- **Comprobabilidad**: Se basa en que pueda comprobar posibles fallas en base a entornos de pruebas, el número de pruebas realizadas es el número del número probable de fallas de la arquitectura, además, la construcción de una arquitectura bien definida tiene como compensación un menor número de fallas.
- **Usabilidad**: Proporciona la facilidad de interacción del usuario con el sistema para realizar diversas tareas y que este proporcione un soporte necesario.

Durante la elaboración de la arquitectura puede tomarse diversos atributos de calidad que sean necesarios. El autor en [28] hace hincapié que los atributos de rendimiento, modificabilidad y seguridad deben considerarse seriamente en el análisis arquitectónico, estos atributos de calidad se muestran en la figura 2.4.

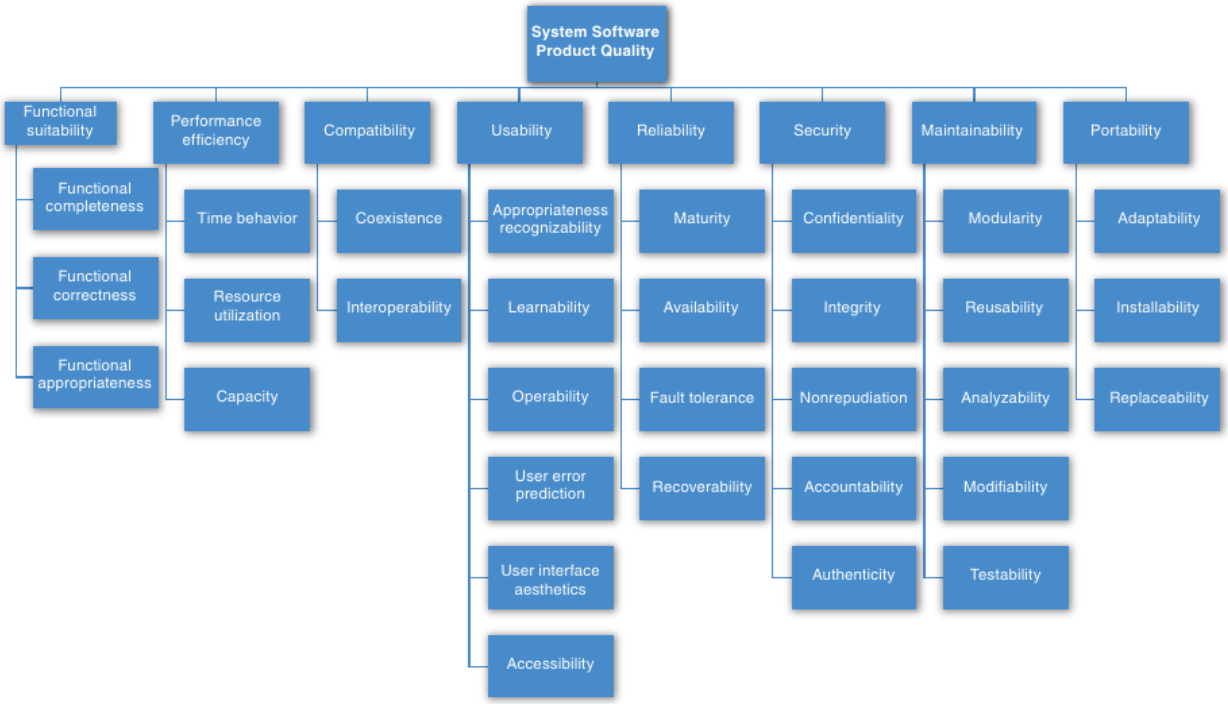


Figura 2.4. Organigrama de atributos de calidad de un producto software.

Fuente: [22].

La figura 2.4 muestra que estos atributos no solo forman parte de la arquitectura, si no de cada proceso que tiene el desarrollo de software así como el producto final.

2.1.2.2. Procesos de desarrollo de la Arquitectura de Software

Existen procesos de desarrollo de software que pueden tener enfoques estándar para la construcción del software [16]. Estos pueden ser adoptados para la elaboración de la arquitectura de software, ya que tiene como base los casos de uso del sistema [32]. Como también atributos de calidad, restricciones con los requisitos funcionales y no funcionales. Que deben ser visualizados desde un alto nivel [28].
El proceso de arquitectura de software puede variar según la necesidad y tamaño del proyecto, el autor en [20] menciona que debe considerarse los siguientes puntos:

- Considerar si la arquitectura debe ser dividida de forma modular.
- Considerar si existen actividades paralelas.
- Estimar el desarrollo del producto final de software.

Por lo tanto existe consideraciones para desarrollar una arquitectura, estos procesos de análisis para la construcción de una arquitectura se muestran en la figura 2.5.

Figura 2.5. Proceso de análisis para la construcción de una arquitectura de software.

Fuente: [30].

La figura 2.5 muestra que el proceso de construcción de la arquitectura, estos procesos son descritos por [30] de manera general:

- **Análisis del dominio**: Esta actividad tiene como objetivo definir las soluciones en base a los problemas existentes del dominio, estas soluciones son posibles mediante el análisis de requisitos arquitectónicos, contexto de las partes interesadas, objetivos del negocio, requisitos funcionales, no funcionales generado los requisitos arquitectónicos significativos [22].
- **Diseño, descripción y decisión arquitectónica**: Esta actividad esta enfocada a la toma decisión de selección de uno o varios diseños arquitectónicos, una vez seleccionado se genera la documentación, plantillas y notaciones pertinentes.
- **Evaluación arquitectónica**: Esta fase busca asegurar que la arquitectura escogida sea correcta en base a los requisitos arquitectónicos significativos.
- **Implementación de la arquitectura**: En esta etapa se hace la construcción de la arquitectura de software, el equipo de desarrolladores debe adoptar y alinearse a las decisiones de diseño de la arquitectura en alto nivel.
- **Mantenimiento de la arquitectura**: Esta fase esta encargada de realizar modificaciones en base a la evolución de la arquitectura, esta evolución se origina en base a posibles nuevos requisitos, demandas tecnológicas, volviéndose a diseñar algunos componentes sin generar algún daño en la arquitectura.

Todas estas actividades como muestra la figura 2.5, no siguen un proceso de manera secuencial, sino, de forma evolutiva e iterativa, estas actividades deben considerarse según el criterio necesario para la solución del problema y ejecutase cuantas veces sea necesario. Además este proceso de desarrollo puede adherirse a cualquier modelo de desarrollo [22].

2.1.3. Microservicios

Los autores en [33] y [34] establecen que los microservicios son un conjunto de servicios independientes y discretos no acoplados con propósitos funcionales, poseen interfaces estándar de comunicación (mensajería, RESTful) con un enfoque de tecnología agnóstica que funcionan en base a los requerimientos. Cada

microservicio debe mantener la lógica de negocio de forma reducida, de esta manera se ajusta a cumplir todos los requisitos comerciales específicos [35]. Trabajando de manera conjunta pero ejecutándose en diversos procesos y despliegues independientes [36] como muestra la figura 2.6.
La definición de un microservicio de [15] se expresa como un proceso cohesivo e independiente que interactúa a través de mensajes con otras entidades. Con la capacidad potencial de reutilización dentro de la solución [37]. Además puede implementarse de manera iterativa, proporcionando una mejor manera de diseñar y desarrollar software [38].

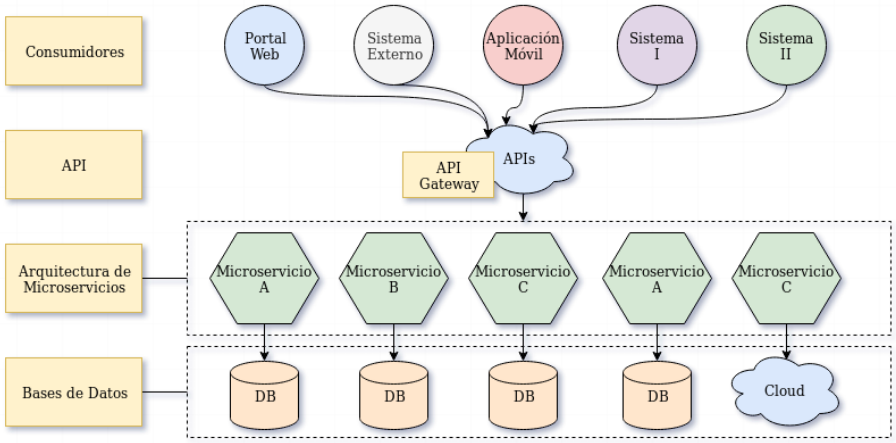


Figura 2.6. Arquitectura de microservicios de un sistema de software.

Fuente: [34].

La figura 2.6 muestra las capas construidas de un sistema de software compuesta por diversas capas, una de estas capas tiene una arquitectura de microservicios donde cada una se conecta a una capa de diversas base de datos locales o en la nube, las funcionalidades de estos servicios son expuestas por una capa API que es utilizada por diversos clientes o consumidores (Sistemas web, móviles, sistemas externos, etc) [34].
Todo microservicio es autónomo y aislado, siendo unidades autocontenidas de dependencias con flexibilidad de acoplamiento, desarrollando todo el proceso de software de forma independiente [15] y [39]. Puede iniciarse, detenerse o reemplazarse en cualquier momento sin un acoplamiento estrecho a otros servicios [40].

2.1.3.1. Arquitectura monolítica

Para el autor en [33] un arquitectura monolítica esta comprometido con la pila técnica en el cual las aplicaciones de este tipo tienen la dificultad en adoptar o cambiar una tecnología, generando una curva de aprendizaje de las estructuras, funciones del sistemas y un esfuerzo de valor comercial. Este tipo de arquitectura tiene un conjunto de componentes y funcionalidades construidas en una sola unidad que generan inconvenientes descritos seguidamente por [34]:

- Consume grandes cantidades de tiempo en las actividades de: mantenimiento, actualización y agregación de nuevas funcionalidades.

- Tiene dificultades para adaptarse a metodologías ágiles, debido a que cada módulo no cuenta con un propio ciclo de vida.
- A medida que exista un crecimiento en la aplicación monolítica conlleva a ralentizar el inicio del sistema.
- No posee modularidad de recursos en base a cada módulo creado que cumpla con especificaciones de rendimiento.
- No puede adoptar nuevas tecnologías debido a que tiende a trabajar con marcos constituidos de forma homogénea o del mismo lenguaje.

Cada punto descrito anteriormente demuestra que este tipo de arquitecturas genera diversos problemas en el proceso de construcción y despliegue de software, pese a la adaptación de diseños de estructuras modulares siguen teniendo dependencias en módulos del sistema [36], volviéndose inflexible y resistente al cambio, generando un impacto significativo para la evolución del sistema, aunque se trate de mitigar abstrayendo con nuevas capas [37].

2.1.3.2. Diseño basado en el dominio

Domain-Driven Design (DDD) tiene el objetivo de modelar la lógica empresarial compleja, donde el software desarrollado esta relacionado a los intereses del usuario y el dominio esta relacionado con el negocio, la clave detrás de este diseño es la descomposición del dominio en subdominios [34] como se muestra en la figura 2.7. Por consiguiente, una arquitectura basada en microservicios puede adoptar el uso de DDD para estructurar internamente cada microservicios [36].

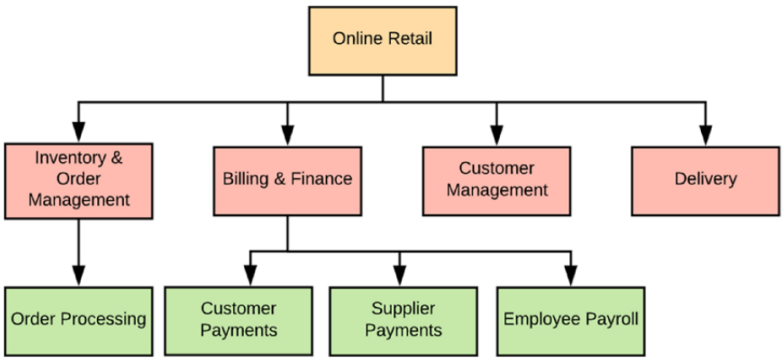


Figura 2.7. Ejemplo de descomposición de dominio en subdominios.

Fuente: [34].

La 2.7 figura muestra un ejemplo de un dominio empresarial con diversos departamentos donde son tratados como subdominios relacionados, de esta manera se encuentra el nivel de granularidad adecuado para la construcción de una arquitectura de microservicios. De esta forma se cumple la ley de Conway, el cual define que una organización que defina la estructura de comunicación entre áreas, producirá un software cuya estructura es una copia de la comunicación de la organización [36].

2.1.3.3. Integración continua

La integración continua o Continuous Integration (CI) es una practica de software que realiza integraciones automáticas de compilación y ejecución [36]. Siendo esencial para los microservicios desde facilitar el despliegue de la aplicación, evitar la miopía tecnológica y tener una visión general del contexto para desarrolladores, reescribir y refactorizar rápidamente servicios sin generar problemas [33].

2.1.3.4. Despliegue continuo

Es un proceso el cual despliega cada artefacto producido, actualizándose con cada iteración realizada de un conjunto de desarrolladores [36]. Adecuándose al proceso de desarrollo de software empresarial, así como la adaptación de requisitos y procesos que necesita el proyecto [38]. Además, este proceso facilita la corrección de posibles errores, el seguimiento de cambios y la verificación de calidad del código generado de la aplicación desplegada [34].

2.1.3.5. Contenedores

Un contenedor de aplicaciones esta basado en la virtualización, tiene el objetivo de crear espacios aislados con propios procesos separados del sistema operativo, gestionando archivos, procesos y redes que comparten recursos de la computadora sin afectar el sistema operativo en general [34]. Estos procesos son creados sin generar sobrecarga, ya que existen archivos de configuración de recursos para limitar el consumo de estos procesos [36].

2.1.3.6. Descubrimiento de servicios

Es el proceso en el cual se registran los servicios para que otros de la misma índole pueda solicitar su ubicación en la red [34] y [36]. Este proceso tiene cuenta dos procesos para su funcionamiento como lo define [38]:

- **Registro del servicio:** registra el servicio generando metadatos en nivel de servicios, host, puertos y otros de elementos de red.
- **Descubrimiento del servicio:** establece la comunicación entre dependencias en tiempo de ejecución, teniendo balanceo de carga inteligente y monitoreo.

2.1.3.7. Balanceo de carga

Es un proceso necesario para balancear, optimizar y atender las peticiones de tráfico que reciben los servidores sin que estos tengan una sobrecarga [38]. existiendo dos implementaciones: la primera por lado cliente o Client Side (CS) configurando el tráfico mediante el uso de protocolos HTTP/TCP, esta implementación debe realizarse dentro de la aplicación y la segunda por el lado del servidor o Server Side (SS) que integran tecnologías de balanceo de carga que no están integradas dentro de la aplicación final [34] y [36].

2.1.3.8. Patrones de comunicación

DDD contiene un conjunto de patrones de comunicación entre contextos que son aplicados al diseño de microservicios,[36] define los siguientes:

- **Capa anticorrupción**: Proporciona una capa de traducción en caso de obtener servicios de sistemas externos, de esta manera se mantiene el diseño del microservicio dividido de los datos obtenidos externamente.
- **Share Kernel**: Propone que es necesario compartir modelos de dominio cuando existan objetos en común ya sean mediante el uso de bibliotecas compartidas o archivos directamente compartidos.
- **Cliente/Proveedor**: Este patrón proporciona una interfaz como proveedor hacía un cliente, esta interfaz se adapta a cambios requeridos para el cliente.
- **Lenguaje publicado**: Cada microservicio tiene un contexto limitado que siguen lineamientos de un idioma de comunicación (XML, REpresentational State Transfer (REST)) y estos microservicios pueden estar construidos en diferentes lenguajes de programación.
- **Patrón de asociación**: Este patrón puede complementarse con el patrón *Share kernel* ya que su objetivo es construir un núcleo compartido para establecer asociación, cumpliendo el objetivo en conjunto de diversos microservicios.
- **Open Host Service**: Este patrón permite que pueda comunicarse con otros microservicios que implementen la capa anticorrupción, de esta manera se previene que un servicio de tercero no altere el diseño de cada microservicio.
- **Caminos separados**: Proporciona que debe existir segregación entre servicios y evitar integraciones innecesarias, se debe tener en cuenta: costo y beneficio a nivel de desarrollo.

2.1.4. Internet de las cosas

La existencia de diversos objetos tangibles en la sociedad es frecuentemente empleada en diversos campos existentes, estos objetos no tienen la capacidad de comunicación que puedan proporcionar información de forma remota. Por esta razón, IoT nace con el objetivo de proporcionar comunicación, control y detección del mundo físico mediante una red de forma remota, integrando el mundo físico al hardware y software [41]. Es un intermediario entre internet, las cosas y la generación de datos, por tanto, IoT es una red que conecta diversos objetos en cualquier momento y en cualquier lugar mediante el uso de internet con los objetivos de control y monitoreo [42]. IoT abarca una gran cantidad de disciplinas desde la producción de energía, fabricación, agricultura, atención médica, comercio minorista, transporte, logística y otros campos [14].
IoT no solo tiene un enfoque de control y monitoreo en objetos tangibles, sino que tiene una visión donde millones de objetos se comuniquen entre sí a través de Internet Protocol (IP) [43]. De esta manera genera grandes cantidades de datos

(*Big Data*) que proporcionan información, estadísticas predicativa y descriptiva para ser transformada en conocimiento con objetivos de productividad y eficiencia [14], [44].

2.1.4.1. Arquitectura IoT

Los autores en [42] y [45] definen a una arquitectura IoT como una colección de objetos físicos, sensores y actuadores con capacidad de comunicación bajo estándares y protocolos IoT aceptados que funcionan en base a determinadas capas como en la figura 2.8.
Las capas definidas en IoT son partes funcionales de todo sistema IoT, cumpliendo funciones específicas capaces de identificar y localizar objetos de manera inteligente en tiempo real, mediante el uso de componentes diversos que lo lleva a la recopilación de datos y envio a través de la nube o red existente para su procesamiento y almacenamiento.

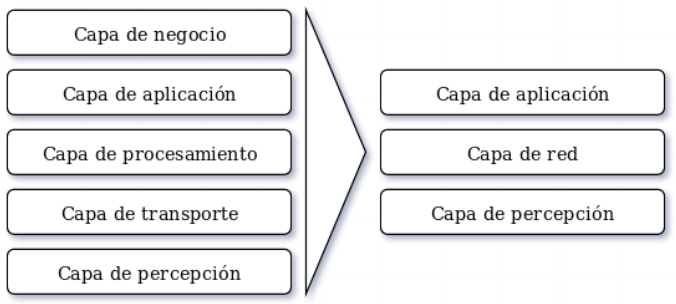


Figura 2.8. Capas de una arquitectura convencional de IoT.

Fuente: [42].

La figura 2.8 muestra la arquitectura IoT convencional basada en (05) cinco capas. Esta arquitectura permite ser escalable y flexible para adaptarse a diversos componentes y tecnologías, estas capas son descritas seguidamente como afirman [42], [44] y [45]:

- **Capa de negocio**: Capa que realiza la administración general de todas las actividades y servicios de IoT.
- **Capa de aplicación**:La capa proporciona servicios de aplicación específicos a usuarios.
- **Capa de procesamiento**: Es la capa intermedia o capa *middleware* tiene la capacidad de almacenamiento, envio y procesamiento de datos, puede utilizar diversas tecnologías como bases de datos, procesamiento de *Big Data* y otros módulos.
- **Capa de transporte**: Capa con el objetivo de transportar los datos de la capa de percepción hacía la capa de procesamiento y viceversa a través de tecnologías como: bluetooh, 3G, 4G, Local Area Network (LAN), Near Field Communication (NFC) y Radio Frequency Identification (RFID).

- **Capa de percepción**: Esta capa física anexa componentes que determina la detección de información, ambiente, parámetros físicos y otros objetos inteligentes.

La arquitectura también puede definirse en (03) tres capas, abstrayendo la capa de negocio dentro de la capa de aplicación así como la capa de trasporte y procesamiento dentro de la capa de red, el cual define la idea general de todo IoT.

2.1.4.2. Protocolos de Comunicación

Todo dispositivo IoT tiene la posibilidad de comunicación con otros dispositivos de su misma índole, sistemas propios o sistemas de terceros. Existen diversos protocolos de comunicación con diseños específicos descritos seguidamente por [41], [42], [43], [44] y [46] como los más importantes:

A. MQTT : Message Queuing Telemetry Transport (MQTT) es un protocolo estándar de mensajería ligera basado en Transmission Control Protocol (TCP), que permite la comunicación entre dispositivos IoT, computadoras portátiles, teléfonos móviles y otros dispositivos inteligentes conectados. La comunicación es posible por medio del envío de mensajes a través de agentes MQTT conectados, esta comunicación es bidireccional entre puntos finales (*endpoints*), obteniendo una ventaja en comparación con el protocolo HTTP, puesto que requiere menor consumo informático, menor consumo de energía y menor uso en ancho de banda, esta ventaja significativa es debido a que el modelo utilizado es de tipo publicador-suscriptor como se detalla en la figura 2.9, donde los dispositivos suscriptores consumen los datos del publicador, implementando los siguientes métodos:

- Conexión
- Desconexión
- Publicación
- Suscripción
- Cancelación
- Remover suscripción

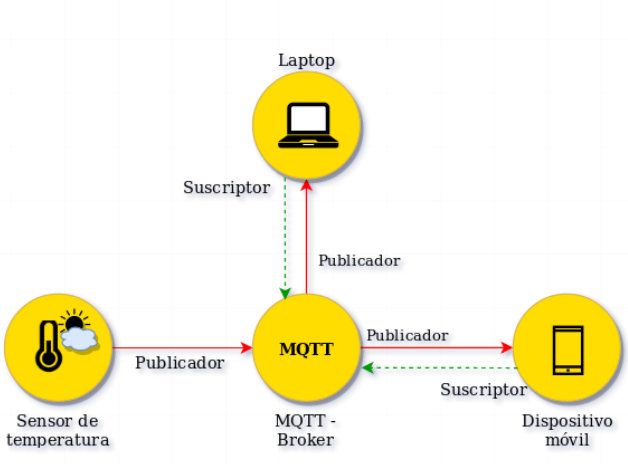


Figura 2.9. Modelo de operación del protocolo MQTT.

Fuente: [46].

La figura 2.9 muestra el funcionamiento del protocolo MQTT usado por un sensor de temperatura funcionando como un publicador, de esta manera se envían los datos del sensor hacía otros dispositivos (laptop y dispositivo móvil), estos dos dispositivos implementan el método de suscripción para tener una comunicación bidireccional de los datos.

B. CoAP: Constrained Application Protocol (CoAP) es un protocolo especializado de transferencia web para el uso en nodos y redes IoT con el objetivo de facilitar el intercambio de mensajes de forma flexible entre puntos finales (*endpoints*), la figura 2.10 muestra el modelo de operación del protocolo, que admite cuatro (04) tipos de mensajes:

- Confirmable (CON).
- No confirmable (NON).
- Reconocimiento (ACK).
- Reiniciar (RST).

Este modelo de mensajería también incorpora distintos tipos de transportes (TCP, Transport Layer Security (TLS), WebSocket y Short Message Service (SMS))[1]. Además, CoAP comparte características similares con el protocolo HTTP y cubriendo sus carencias tales como:

- Descubrimiento de recursos.
- soporte de multidifusión IP.
- Implementación en diferentes lenguajes de programación.

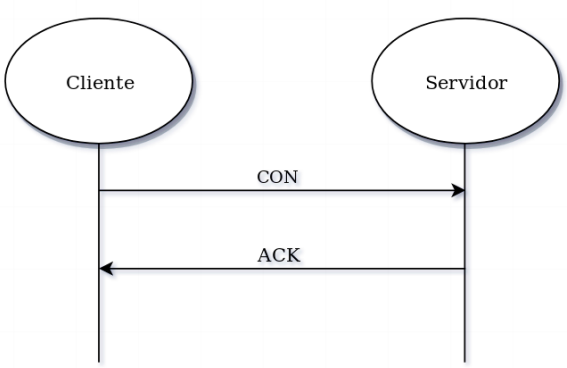


Figura 2.10. Modelo de operación del protocolo CoAP.

Fuente: Elaboración propia.

En la figura 2.10 muestra el funcionamiento del protocolo CoAP donde el cliente hace una petición del del mensaje(CON) y el servidor envía una respuesta de reconocimiento (ACK).

C. XMPP: Extensible Messaging and Presence Protocol (XMPP) es un protocolo de comunicación basado en XML que funciona bajo TCP, con características adicionales como: seguridad, autenticación de usuarios, monitorización,

[1]https://www.openmobilealliance.org

contactos y bloqueos. Este protocolo además cuenta con el modelo publicador/suscriptor para la distribución múltiple de información hacía otros dispositivos, la figura 2.11 describe el modelo de operación de XMPP.

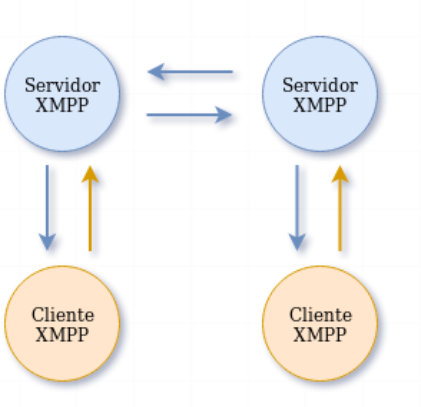


Figura 2.11. Modelo de operación XMPP.

Fuente: Elaboración propia.

La figura 2.11 muestra el funcionamiento del protocolo XMPP, los clientes se suscriben intercambiando información con la conexión a diversos servidores, de esta manera los mensajes recibidos y enviados se realizan a tiempo real.

D. SOAP: Simple Object Access Protocol (SOAP) es un protocolo basado en mensajería XML ampliamente utilizado, permite el intercambio de información mediante el uso de una Uniform Resource Identifier (URI). Además, este protocolo tiene la capacidad de detectar errores y la seguridad del intercambio de datos. Cada mensaje tiene que tener un tamaño adecuado para utilizarlo con dispositivos IoT debido a que SOAP funciona bajo la capa de aplicación, la figura 2.12 muestra el modelo de operación.

Figura 2.12. Modelo de operación SOAP.

Fuente: Elaboración propia.

La figura 2.12 muestra la comunicación entre un cliente y servidor, el cliente hace el envio de información con formato XML, el servidor obtiene los datos y manda alguna respuesta hacía el cliente.

E. REST: REST es un estilo arquitectónico ligero para el uso de servicios web, al igual que SOAP es accedido mediante una URI pero con una notación y formato JSON bajo el funcionamiento de la capa de aplicación, que frente a XML lo hace más liviano, REST tiene una adecuación liviana para el intercambio de información con clientes IoT como se muestra en la figura 2.13.

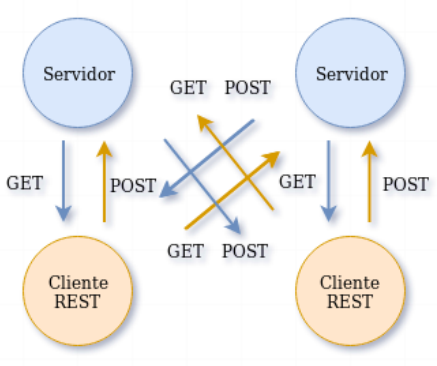


Figura 2.13. Modelo de operación REST.

Fuente: Elaboración propia.

La figura 2.13 muestra que cada cliente hace una petición hacía los servidores el cual intercambian información controlados por códigos de estado HTTP.

2.1.4.3. Patrones de diseño IoT

La conexión de objetos y dispositivos permite que IoT explote y utilice de manera adecuada diferentes redes, ofreciendo servicios de innovación en diversos escenarios según su granularidad y nivel de abstracción. Por este motivo, todo dispositivo IoT adopta un patrón de diseño acorde al escenario y a la interacción requerida, ya sea por protocolos compatibles, métodos y capacidades de comunicación o potencia computacional, requiriendo que su diseño sea robusto y abstrayendo la solución de forma reutilizable para la solución de problemas, similar a los patrones de diseño de software, estos patrones IoT se dividen generalmente en los siguientes como lo definen [43], [44] y [47]:

A. Cliente en Tiempo real: *Realtime Clients* es un patrón IoT con la capacidad de obtener o detectar datos que posteriormente son expuestos a diversos clientes en tiempo real por medio de un servidor, MQTT, CoAP y XMPP proporcionan los métodos necesarios para trazar la recepción de datos e información de donde proviene, de esta manera se mantiene una comunicación bidireccional.

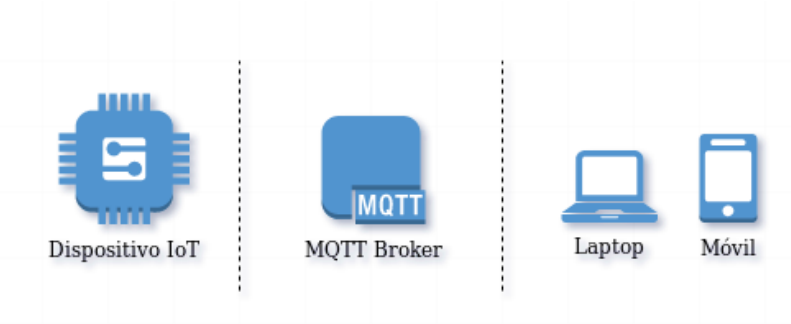


Figura 2.14. Modelo de patrón de diseño a tiempo real.

Fuente: Elaboración propia.

La figura 2.14 muestra la comunicación entre el dispositivo IoT y el cliente (laptop y teléfono móvil) por medio del protocolo MQTT, el cual posee un *Broker* que funciona como un nodo central y tiene la responsabilidad de enviar la información a los clientes, similar a la función de un servidor.

B. Control Remoto: *Remote Control* permite operar dispositivos IoT de forma remota interactuando con objetos físicos del entorno que tengan una misma red, este patrón hace el envio de comandos a dispositivos con una o diversas acciones, en mayor parte se da el uso para automatización de hogar.

Figura 2.15. Modelo de patrón de diseño control remoto.

Fuente: Elaboración propia.

La figura 2.15 muestra un ejemplo del patrón control remoto, el teléfono móvil envia un comando mediante un protocolo de comunicación (XMPP) hacía el dispositivo IoT que controla el objeto físico (cerradura) según el tipo de comando.

C. Ubicación Consistente: mejor conocido como *Location-aware* el cual es una tecnología ya existente de reconocimiento de ubicación con capacidades de detección de ubicación, manipulación y control de eventos e información, generalmente estos cálculos de ubicación geográfica son posibles por Global Positioning System (GPS) y triangulación de red, permitiendo la transmisión la ubicación del dispositivo.

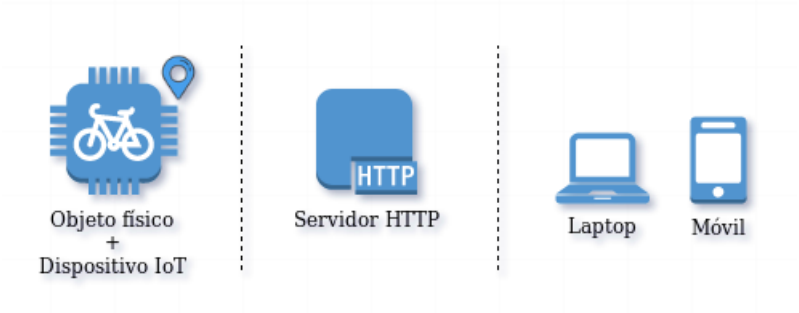


Figura 2.16. Modelo de patrón de diseño Ubicación consistente.

Fuente: Elaboración propia.

La figura 2.16 muestra un objeto físico (bicicleta) que incorpora un dispositivo IoT con tecnología de ubicación (GPS en este caso), el cual envia las coordenadas de ubicación hacía un servidor HTTP, donde un cliente hace peticiones al servidor desde otros dispositivos electrónicos.

D. Maquina a Humano: o Machine to Human (M2H) es un patrón donde el dispositivo IoT que interactúa con una persona humana a través de dispositivos portátiles, haciendo envio de datos previamente recopilados o calculados donde puedan dar acciones sugerentes hacía las personas o algún proceso que requiera intervención humana.

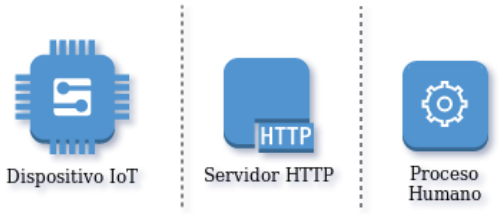


Figura 2.17. Modelo de patrón de diseño M2H.

Fuente: Elaboración propia.

La 2.17 muestra como un dispositivo IoT envia datos de un objeto físico hacía un servidor HTTP, este servidor recopila y procesa estos datos con el uso de algoritmos para generar conocimiento ya sea para recomendación o predicción mediante técnicas de *Machine Learning*, esta información es enviada a diversos métodos de uso para consideración que considere relevante una persona.

E. Maquina a Maquina: M2M es un patrón que tiene como principio la comunicación entre dispositivos IoT, con el objetivo de reducir la intervención humana, ya que puede de forma autónoma responder o tomar acciones que requieran otras máquinas. Dentro de este patrón incluye M2H como valor agregado para facilitar información hacía el usuario con fines de información.

Figura 2.18. Modelo de patrón de diseño M2M y M2H.

Fuente: Elaboración propia.

La figura 2.18 muestra una comunicación M2M (recuadro plomo) entre el dispositivo IoT A y dispositivo IoT B mediante el uso del protocolo de mensajería MQTT. Las acciones tomadas por el dispositivo IoT B sobre A son enviadas a un servidor HTTP donde puede mostrarse esta información de manera gráfica, como interfaz de usuario o como servicio web.

2.1.4.4. Seguridad en IoT

Todo escenario IoT puede abstraerse de sus limitaciones, heterogeneidad, conectividad y seguridad. Este último es considerado crucial en su funcionamiento como producto final, debido a que estos dispositivos son sometidos y expuestos en diversas áreas de la sociedad, desde ambientes sin intervención humana hasta ambientes agresivos con mucha interactividad donde no existe supervisión constante [47]. Estos dispositivos IoT proporcionan los siguientes servicios al igual que la seguridad de las redes informáticas y los sistemas de información como refiere [43]:

- **Integridad**: Garantizar que no exista modificación de datos por terceros.
- **Autenticación**: Verificar la identidad de datos, usuarios y dispositivos.
- **No repudio**: Garantizar que el mensaje no pueda ser negado a un remitente.
- **Disponibilidad**: Garantizar la disponibilidad a usuarios verificados.
- **Privacidad**: Garantizar que los datos, comportamientos o acciones de usuarios de un sistema no sean rastreables o identificables hacía terceros no autorizados.
- **Confidencialidad**: Garantizar que la información sea ininteligible para terceros no autorizados.

Estos objetos IoT deben poseer características mínimas de seguridad, los autores en [41], [42], [43], [44] y [47] señalan determinadas consideraciones para la seguridad de diversos dispositivos:

A. **Prevención de amenazas y ataques**: La criptografía es una herramienta que provee servicios de seguridad básica, existen diferentes tipos de criptografía, una de ellas es la criptografía ligera o *Lightweight Cryptography* que esta

acorde para el uso en dispositivos IoT, con un menor consumo energético y bajo requerimiento computacional, no obstante, esto no debe confundirse la palabra "ligera" con "Debilidad" o que la seguridad se vea comprometida con algún riesgo.

B. Seguridad en la arquitectura: Todo dispositivo IoT esta basado en la arquitectura compuesta por cinco (05) o tres (03) capas descrito en el punto 2.1.4.1, en este dominio arquitectónico, la capa de red y la de aplicación deben contener mecanismos de seguridad contra ataques de comunicación, denegación de servicio o clonación de identidad de productos IoT, mientras que la capa de percepción debe contener una comunicación segura con el objeto físico. Sin embargo, existe un desafío en los dispositivos IoT debido a que puede sufrir amenazas físicas.

C. Tolerancia a fallos : La tolerancia a fallos es esencial para la confiabilidad del servicio. Existen tres tipos que colaboran con la tolerancia a fallas en IoT:

- **Aseguramiento de objetos en base al software**: Se debe tomar mecanismos y protocolos para desarrollar mejor el software, ya puede existir muchos dispositivos.
- **Capacidad de reconocimiento de servicios**: Se tiene que tomar en cuenta que todo dispositivo debe estar dentro de un sistema donde pueda enviar estados y servicios, de esta manera se tiene información de situaciones anómalas.
- **Retroalimentación y protección de ataques**: En todo protocolo implementado debe poder recuperarse y hacer retroalimentación por medio de otros mecanismos o entidades IoT, de esta manera puede generar un mapeo de las áreas inseguras de ataques o denegación de servicios.

2.1.4.5. Dispositivos IoT

Todo dispositivo IoT es un objeto inteligente emergente informático impulsado por la electrónica que requiere energía necesaria para su funcionamiento así como sus componentes anexados, [43] y [44] describen el diseño de alto nivel de un IoT así como diversos dispositivos de hardware.

A. Diseño en alto nivel de IoT: permite conocer las distintas partes que componen un IoT, provisto por cuatro (04) componentes.

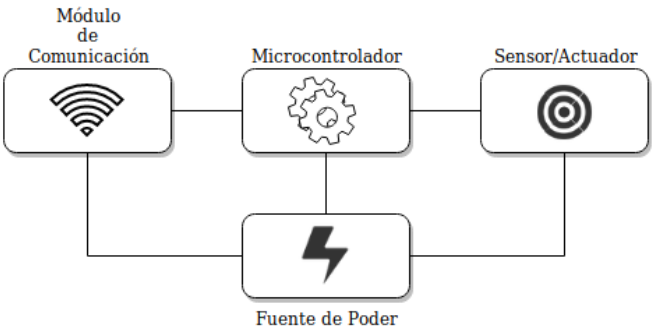


Figura 2.19. Componentes de un IoT.

Fuente: [43].

La figura 2.19 muestra los cuatro (04) componentes con funciones específicas descritas a continuación:

- **Módulo de comunicación**: Permite la asignación de funcionalidades y capacidades de comunicación al objeto inteligente, esto atribuye al objeto inteligente las capacidades de comunicación por medio de uso de cables o soldadura.
- **Microcontrolador**: Es el núcleo principal del objeto inteligente que permite adaptar el comportamiento requerido mediante el uso de un microprocesador que ejecuta software.
- **Sensores o actuadores**: Son dispositivos con capacidades de detectar y percibir diferentes variables de instrumentación físicas.
- **Fuente de alimentación**: Es el pilar necesario requerido por el objeto inteligente, estos pueden provenir de baterías, piezoelectricidad, panel solar o energía eléctrica.

B. **Arduino**: Es una empresa dedicada a la fabricación de microcontroladores así como la construcción de dispositivos digitales que pueden detectar e interactuar con el mundo físico. Los diseños de placa adoptan diversos microprocesadores y controladores equipados con un conjunto de pines de entrada, salida digital y analógica, así como también placas de expansión interconectadas a otros circuitos. El lenguaje de programación utilizado es C y C++ con diversas librerías para su uso.

C. **Raspberry Pi**: Nació con el objetivo inicial para soporte a la robótica y extendió su uso a diversos propósitos, una de las características es que puede ejecutar diversos sistemas operativos conocidos (Raspbian, linux, Windows y Android Things). Esta combinación de hardware, software y sistemas operativos de alto perfil funcione para el dominio de IoT.

D. **Intel Galileo**: Es una placa certificada por Arduino basada en la arquitectura x86 de Intel Quark SoC X1000, combina la tecnología Intel con soporte de módulos, bibliotecas y sotware de arduino, el sistema operativo usado esta en base Linux.

E. **Near field communication**: O NFC, es una tecnología inalámbrica de alta frecuencia con un radio de acción mínimo, con funcionalidades de intercambio de información y comunicación hasta 15cm, generalmente este se encuentra en diversos etiquetas, tarjetas y teléfonos móviles.

F. **Radio Frequency Identification**: O RFID al igual que NFC es una tecnología inalámbrica con la diferencia que existe diferentes rangos de frecuencia de comunicación para la lectura. Estos dispositivos diminutos existen en diferentes presentaciones sin la necesidad de utilizar energía según el tipo de etiqueta.

2.2. Objetivos

2.2.1. Objetivo General

Diseñar e implementar una arquitectura de software basada en microservicios para el uso en dispositivos IoT.

2.2.2. Objetivo Específico

Para generar el resultado del objeto general se proponen los siguientes objetivos específicos:

- Identificar tecnologías existentes para diseñar e implementar una arquitectura de software basada en microservicios.
- Definir una arquitectura de software basada en microservicios para proveer comunicación e información a sistemas de terceros o propios.
- Implementar y desarrollar dispositivos IoT usando componentes electrónicos y otros elementos auxiliares de comunicación.
- Comprobar la comunicación de dispositivos IoT mediante el uso de microservicios.

2.3. Justificación del estudio

La evolución de la tecnología requiere cambios constantes que puedan adaptarse a las demandas de los diferentes dominios en la sociedad (industria, comercio, educación y hogar). Uno de estos cambios es la existencia de nuevos dispositivos no invasivos que tienen interconexión digital con diferentes sistemas llamados IoT, permitiendo así el fácil acceso a la gestión de información de los objetos físicos en diferentes entornos [48]. Estos dispositivos IoT son utilizados en diferentes ámbitos junto a las tecnologías de computación en la *Cloud*, *Big Data*, análisis de datos y otros tipos de sistemas independientes, generando un gran impacto en la economía y en la sociedad, conocida como la Industria 4.0 o cuarta revolución industrial [5].

En adición, el uso de estos dispositivos a lo largo de los años se ha incrementado en diversos dominios, generando de esta manera una estimación de más de 20 billones de dispositivos IoT conectados para el año 2020 [1], [3], [6], [8].
La existencia de diversos tipos de sistemas en su mayoría con una arquitectura monolítica limita la capacidad del sistema para adaptarse, escalar y evolucionar a cambios que el software requiera [8]. Este tipo de arquitectura utilizada actualmente para la integración con dispositivos IoT, no es recomendable debido a que estos sistemas presentan deficiencias como lo señala [15]:

- **Tamaño del sistema:** La complejidad de este tipo de sistemas se incrementa con cada integración de nuevos módulos, incrementando más líneas de código, tiempo para la comprensión del sistema y la realización de cambios de manera significativa, puede comprometer en parte la funcionalidad y disponibilidad del sistema por los diversos módulos acoplado; que están en funcionamiento.

- **Desarrollo dependiente**: Todos los componentes del sistema trabajan juntos, por lo que alguna modificación o reescritura de algún módulo puede ser perjudicial para otros módulos dependientes, el cual genera un tiempo de inactividad para determinar las nuevas funcionalidades del componente actualizado.

- **Uso de recursos y configuraciones**: La necesidad de generar un conjunto de dependencia en este tipo de sistemas produce inconsistencias y conflictos al añadir librerías, no obstante el desarrollador debe estar estrictamente comprometido y familiarizado a un entorno de configuración único para la codificación y pruebas con respecto a los módulos del sistema, obteniendo altos costos de recursos humanos y computacionales.

- **Compilación y despliegue**: La compilación y despliegue de todo un sistema para integrar nuevos módulos con funcionalidades para cada respectivo dispositivo físico IoT, genera una incidencia en cuanto al tiempo de compilación, debido a que este proceso se vuelve repetitivo por cada nueva integración de diversos módulos o dispositivos que requieran interactuar con el sistema, del mismo modo perjudica el uso de otros módulos durante el despliegue del sistema.

- **Tecnologías emergentes**: Este tipo de sistemas tiene complicaciones para adoptar nuevas tecnologías dado que los cambios afectarían la aplicación completa, el cual origina mucha demanda en tiempo y costo.

Por lo expuesto, el uso de una arquitectura de microservicios en comparación a una arquitectura monolítica es la capacidad de poder descomponer en pequeñas aplicaciones con responsabilidades individuales que se pueden implementar, escalar y probar de forma independiente, actuando como sub-unidades independientes que se interconectan y comunican con otros sistemas, el cual lo hacen candidato para el uso con dispositivos IoT, por lo que el uso de esta arquitectura basada en microservicios es liviana y fácil de actualizar en diferentes escenarios en los que no se pueden anticipar las funcionalidades [2],[6], [8].

2.4. Viabilidad

Para implementar la arquitectura de software basada en microservicios para el uso en dispositivos IoT es necesario la construcción de software que supla las necesidades requeridas, también es requerido la implementación del hardware para diseñar y construir los dispositivos IoT, esto requiere diversos componentes que estén integrados de manera física adecuado a factores ambientales que requiera cada escenario.

2.4.1. Viabilidad económica

Todos los recursos económicos y humanos son solventados por el tesista que son detallados en las tablas 2.1, 2.2, 2.3 y 2.4.

Tabla 2.1. Tabla de costos de hardware empleado en la investigación.

Descripción	Unidades	Precio	Sub Total
Arduino Uno	03	S/ 35.00	S/ 105.00
Módulo ESP8266	03	S/ 20.00	S/ 60.00
Módulo RFID-RC522	01	S/ 20.00	S/ 20.00
Relé	02	S/ 20.00	S/ 40.00
Higometro de humedad FC-28	01	S/ 20.00	S/ 20.00
Bomba de agua	01	S/ 20.00	S/ 20.00
Fuente de alimentación	03	S/ 20.00	S/ 60.00
Componentes electrónicos	01	S/ 100.00	S/ 100.00
		Total	S/ 425.00

Fuente: Elaboración propia.

La tabla 2.1 el hardware necesario en mayor parte son componentes electrónicos requeridos para la construir dispositivos IoT, siendo componentes claves el microcontrolador, sensores y actuadores.

Tabla 2.2. Tabla de recursos humanos del proyecto

Rol	**Responsable**	**Cantidad**
Encargado del Proyecto	Tesista	01
Scrum Master		01
Product Owner		01
Desarrollador Back-End		01
DevOps		01

Fuente: Elaboración propia.

La tabla 2.2 muestra los roles basados en Scrum, todos estos roles son asumidos por el tesista de manera que no existe gasto económico para el recurso humano.

Tabla 2.3. Costo de herramientas tecnológicas.

Descripción		Precio
Amazon AWS EC2		S/ 150.00
	Total	S/ 150.00

Fuente: Elaboración propia.

La tabla 2.3 muestra el costo de alojamiento de la nube, este servicio es utilizado para desplegar los microservicios y exponerlos para el uso en dispositivos IoT.

Tabla 2.4. Costo de suministros y servicios.

Descripción	Precio
Consumo energético	S/ 50.00
Consumo hídrico	S/. 5.00
Servcicio de Internet	S/ 45.00
Otro suministros	S/ 35.00
Total	S/ 135.00

La tabla 2.4 muestra el costo de los suministros y servicios requeridos para elaborar el producto de software, también estos recursos son usados para los dispositivos IoT que se describen seguidamente:

- Los dispositivos IoT requieren una fuente de alimentación externa por medio de un adaptador.
- En el caso de un dispositivo IoT aplicado al escenario de riego requiere un suministro de agua para el riego de planta, así como también otros materiales que permitan facilitar el transporte de este liquido.
- Los dispositivos IoT requieren conexión constante a un punto de acceso de internet, para comunicarse con los servicios web proporcionados un servidor alojado en la nube.

2.4.2. Viabilidad técnica

Para la parte de implementación de los microservicios es necesario un conjunto de herramientas que genera una plataforma que facilite el desarrollo de software, también es necesario un conjunto de componentes de hardware para construir el dispositivo IoT, la tabla 2.5 muestra la pila de tecnologías usadas que cubre las partes de *Backend* y *DevOps*; mientras que la tabla 2.6 muestra los componentes electrónicos. Todas las herramientas y tecnologías seleccionadas son instaladas en 02 computadoras, El primer equipo es utilizado para el desarrollo de software y el segundo equipo es manejado como servidor de integración y entrega continua.

Tabla 2.5. Tabla de tecnologías y herramientas de software.

Herramientas de software	
JDK / OpenJDK	Kit de desarrollo para la construcción de código en java.
Jakarta EE	Conjunto de estándares de tecnologías de *Server Side* dedicadas al desarrollo de software bajo el lenguaje de programación Java del lado del servidor.
Jetty	Servidor web con soporte HTTP basado en java.
Maven	Gestión de proyectos basados en lenguaje de programación java y administrador de dependencias.
MySQL	Sistema de gestión de base de datos relacional.
JRebel	Herramienta que permite realizar cambios en código sin necesidad de desplegar una aplicación continuamente.
Sloeber IDE	Entorno de desarrollo para la construcción de código fuente C++ y arduino.
Docker	Software que permite la gestionar contenedores mediante la virtualización de aplicaciones.
Portainer	Administrador y gestionador de contenedores docker en forma visual.
Fritzing	Programa de creación de esquemas electrónicos para la construcción de prototipos y productos finales.
Netbeans IDE 12	Entorno de desarrollo para la construcción de código fuente BackEnd en java.
Workbench Mysql	Herramienta visual integral bajo la funcionalidad de MySQL.
Swagger	*Framework* que permite consumir, visualizar y documentar servicios web.
GitLab	Servicio de control de versionamiento y gestor de repositorios.
Docker	Software que permite la virtualización de aplicaciones.
Jenkins	Software que permite la integración continua y despliegue automático.
SonarQube	Software que permite el análisis de código estático para mejora de calidad de productos de software.
Ubuntu	sistema operativo de código abierto para computadoras e IoT, es una distribución de Linux basada en la arquitectura de Debian.

Fuente: Elaboración propia.

Tabla 2.6. Tabla de componentes de hardware.

Hardware	
Arduino Uno	Prototipo electrónico de código abierto (*open- source*) basada en hardware y software flexibles y fáciles de usar.
Módulo ESP8266	Dispositivo complementario que permite la conexión Wi-Fi.
Módulo RFID-RC522	Dispositivo complementario que permite lectura y escritura de etiquietas NFC.
LED	Fuente de luz constituida por semiconductores.
Resistencias	dispositivo eléctrico que tiene la particularidad de oponerse al flujo de la corriente.
Sensor de humedad	Sensor que detecta y hace lectura de la humedad.
Bomba de agua	Actuador que permite succión de agua por medio de energía de presión y energía centrífuga.
Relé	Dispositivo electromagnético utilizado como interruptor que controla un circuito eléctrico.
Notebook	Computadora portátil para el desarrollo e implementación de la investigación.
Fuente de alimentación	Fuente de alimentación de voltaje para dispositivos.
Shield Protoboard	Placa de inserción para circuitos electrónicos.
Tarjetas NFC / RFID	Dispositivo que almacena datos limitados adherido a superficies plásticas .

Fuente: Elaboración propia.

2.4.3. Viabilidad operativa

En nuestro medio se cuenta con los recursos humanos disponibles con los conocimientos suficientes para construir este tipo de arquitecturas. Así mismo, La arquitectura basada en microservicios, dispone de capacidades de comunicación web para ser usados en dispositivos IoT y en diversos sistemas; los que permitirían mayor accesibilidad y en su implementación.

2.5. Limitaciones

La arquitectura basada en microservicios y los dispositivos IoT están conectados con los objetos físicos en escenarios específicos, interactuando y generando información, que cual denota las siguientes limitaciones:

- Los dispositivos IoT están construidos bajo el microcontrolador Arduino.
- El componente Wi-Fi usado en el dispositivo IoT es un controlador ESP8266-01.

- Los dispositivos IoT únicamente se comunican mediante el protocolo de comunicación HTTP.
- Los dispositivos IoT requieren conexión a internet.
- Los dispositivos IoT funcionan específicamente en los escenarios de: asistencia a eventos participantes, irrigación de planta y control de cierre remoto.
- El dispositivo IoT que funciona en el escenario de control de cierre remoto se desarrolla solo como prototipo.
- Todos microservicios proveen servicios web disponibles por medio del protocolo HTTP.
- Los microservicios son construidos bajo las especificaciones necesarias en base a los escenarios.

Capítulo 3

Metodología de desarrollo

3.1. Definición del Backlog del producto

Para el desarrollo de este proyecto, se emplea la metodología ágil Scrum, que permite cumplir con los objetivos específicos así como obtener resultados óptimos en el diseño e implementación, proporcionando un componente clave para adaptarse a las historias de usuario creadas en base a los siguientes escenarios:

Control de apertura remota: El escenario es un prototipo enfocado a realizar aperturas de una cerradura de forma remota, el prototipo verifica constantemente una señal enviada por un usuario para realizar la acción correspondiente, algunos de los requisitos se muestran en la tabla 3.1.

Tabla 3.1. Elementos principales del Backlog de apertura remota.

ID	Como ...	Necesito ...	Para ...	Sprint
8	Usuario	Abrir la puerta remotamente	Permitir el acceso de personas	1
9	Usuario	Verificar el estado de la puerta	Comprobar si la puerta esta abierta o cerrada	1
10	Usuario	Listar los últimos quince estados de la puerta	Controlar el registro	1

Fuente: Elaboración propia.

La tabla 3.1 muestra los elementos del backlog como una primera iteración del sprint correspondientes al escenario, las historias de usuario describen que el usuario no tiene interacción directa con la puerta.

Registro de asistencia de participantes a eventos: Este escenario esta situado dentro del dominio de educación universitaria, específicamente en el registro de actividades académicas. El escenario considera como objetivo principal registrar la asistencia de los alumnos por medio del uso de etiquetas o tarjetas NFC, estos eventos son gestionados por un coordinador para disponer de un mejor control de los créditos extracurriculares, la figura 3.2 muestra los elementos del backlog.

Tabla 3.2. Elementos principales del Backlog de los usuarios.

ID	Como ...	Necesito ...	Para ...	Sprint
1	Coordinador	Gestionar eventos académicos	controlar los créditos extracurriculares de los alumnos	2
2	Coordinador	Visualizar los asistentes por cada evento	Controlar la cantidad de asistentes por evento	2
3	Coordinador	Conocer el total de créditos de un alumno	Asegurar que el alumno cumpla con los créditos mínimos necesarios	2
4	Estudiante	Registrar asistencia al evento	Obtener el crédito del evento	2
5	Estudiante	Listar nuevos eventos académicos	Saber a que evento debo de asistir según mi interés	2
6	Estudiante	Listar los eventos académicos asistidos anteriormente	Saber cuantos créditos extracurriculares otorga cada evento	2
7	Estudiante	Conocer el total de créditos extracurriculares acumulados	Saber si cumplo con el mínimo de créditos requeridos	2

Fuente: Elaboración propia.

La tabla 3.2 muestra los principales elementos del backlog del producto, enfocados en las necesidades de los usuarios finales, el ámbito principal entre ambos (coordinador y estudiante) es la creación de eventos, asistencia y control de los créditos extracurriculares.

Irrigación de planta: Este escenario se encuentra situado dentro de los dominios de hogar y agricultura, con el objetivo de principal de suministrar agua para humedecer la tierra, la figura 3.3 muestra los principales elementos del backlog.

Tabla 3.3. Elementos principales del Backlog de irrigación de planta.

ID	Como ...	Necesito ...	Para ...	Sprint
11	Usuario	Conocer la humedad del suelo	Asegurar la hidratación del cultivo de la planta	3
12	Usuario	Humedecer el suelo	Beneficiar el crecimiento de la planta	3
13	Usuario	Listar los últimos riegos realizados	Controlar el registro	3

Fuente: Elaboración propia.

La figura 3.3 muestra los elementos principales del backlog de un determinado sprint con respecto al escenario, similares al prototipo de puerta, debido a que

solo tiene funciones específicas para que el dispositivo IoT pueda realizar así como también comprobar que la acción sea necesario realizar.

Los elementos principales mostrados anteriormente no solo son las únicas historias de usuario, ya que existen historias de usuario por parte del interesado del proyecto como se ve en la tabla 3.4, el cual no se incorpora directamente al grupo de usuarios finales, debido a que su objetivo es influir en el proyecto por medio del conocimiento en los dominios.

Tabla 3.4. Elementos principales del Backlog del interesado.

ID	Como ...	Necesito ...	Para ...	Sprint
14	Interesado	Disponer etiquetas NFC a los alumnos	Registrar las asistencias a los eventos académicos programados	4
15	Interesado	Disponer etiquetas RFID a los alumnos	Registrar las asistencias a los eventos académicos programados como dispositivo auxiliar en caso de problemas con las etiquetas NFC	4
16	Interesado	Obtener el Identificador único de cada etiqueta NFC	Relacionar el identificador único con la información de alumno	4
17	Interesado	Evitar que la etiqueta NFC tenga información	Prevenir la escritura, modificación, eliminación de los datos.	4
18	Interesado	Leer etiquetas NFC por medio de un microcontrolador Arduino y otros componentes	Controlar la asistencia de los alumnos a los eventos	4
19	Interesado	Leer la humedad del suelo por medio de un microcontrolador Arduino y otros componentes	Conocer la humedad de la tierra así como enviar una señal para humedecerla	4

Fuente: Elaboración propia.

La tabla 3.4 muestra los elementos del product backlog del interesado el cual define el hardware que debe de usarse especificando solo dispositivos Arduino y otros componentes eléctricos para crear los dispositivos IoT.

Además de la especificación de los dispositivos también existe una lista de elementos de software especificado por el parte del interesado, la tabla 3.5 muestra las especificaciones de software para ser utilizados por medio de servicios.

Tabla 3.5. Elementos principales del Backlog del interesado.

ID	Como ...	Necesito ...	Para ...	Sprint
17	Interesado	Obtener información del estado de una puerta por medio de un microcontrolador Arduino y otros componentes	Confirmar la apertura de una puerta al usuario final	4
18	Interesado	Generar servicios REST de todos los sistemas	Poder consumir los servicios en sistemas propios o de terceros	4
19	Interesado	Generar documentación de los servicios REST de todos los sistemas	Poder conocer la ruta de los servicios	4
20	Interesado	Comprobar disponibilidad de los servicios	prevenir errores con la conexión de otros sistemas y consumo de servicios	4
20	Interesado	Seguridad en los servicios de registro de asistencia a participantes a eventos	prevenir y controlar el acceso a los servicios y recursos	4
21	Interesado	Visualizar la documentación generada de los servicios REST de todos los sistemas por medio de una interfaz	ver interactivamente los servicios del sistema	4
22	Interesado	Consumir los servicios REST de todos los sistemas por medio de la interfaz gráfica de documentación	Poder ejecutar pruebas de cada servicio	4

Fuente: Elaboración propia.

La tabla 3.5 es parte de la lista de backlog del producto, los elementos de la lista muestra los lineamientos de software que deben de tener los microservicios por parte del interesado del proyecto.

3.2. Diseño

3.2.1. Diseño Hardware

El dispositivo IoT es colocado en cada entorno con el rol principal de la emisión, recepción, recolección de datos, control y monitoreo del objeto físico siendo posible por su componentes principales como demuestra en la figura 3.1.

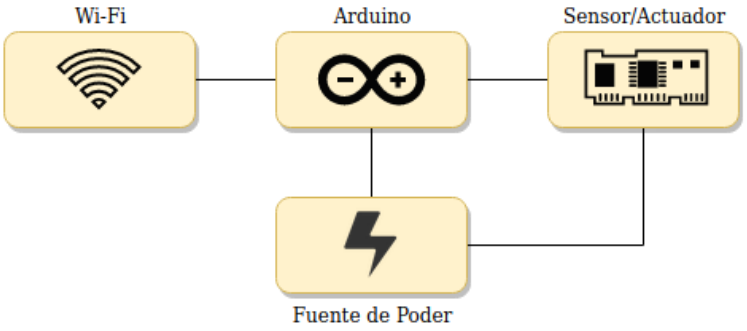


Figura 3.1. Diagrama general del diseño del dispositivo IoT

Fuente: Elaboración propia.

La figura 3.1 muestra la composición del dispositivo IoT, el componente principal es el microcontrolador Arduino conectado con un módulo Wi-Fi, permitiendo la comunicación con servicios web, que determina las acciones previstas por el usuario para que el sensor o actuador interactué con el objeto físico, suplidos por una fuente de poder o fuente de alimentación eléctrica.

A. Diseño de dispositivo de registro de asistencia de participantes a eventos con etiquetas NFC/RFID

Este dispositivo IoT tiene como objetivo realizar lecturas de etiquetas no invasivas NFC/RFID, poseen un identificador único el cual es extraído por medio de un lector, de forma que los usuarios portadores de estas etiquetas pueden ser rápidamente identificados, este dispositivo se muestra en la figura 3.2.

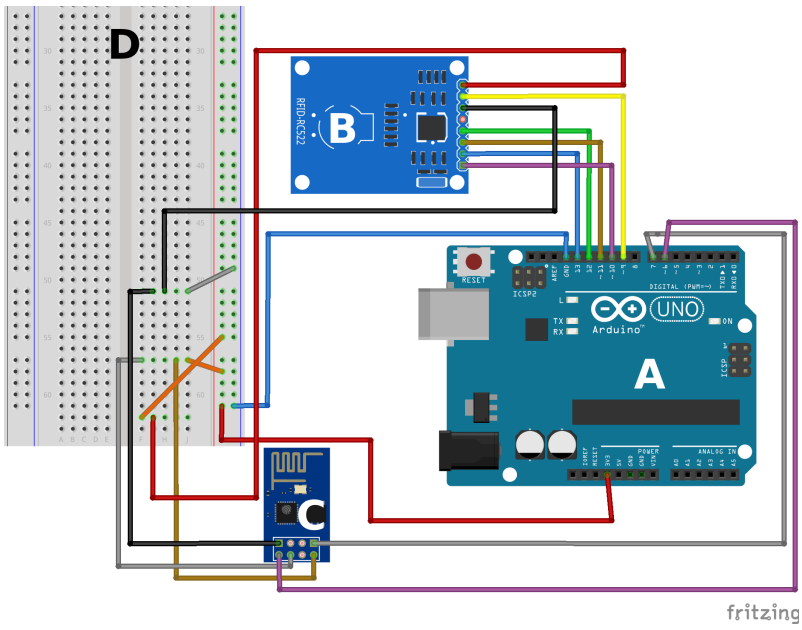


Figura 3.2. Diagrama del prototipo IoT de lectura NFC/RFID.

Fuente: Elaboración propia.

La figura 3.2 muestra la conexión e interacción que existe entre componentes con el microcontrolador Arduino UNO (A), que provee suministro energético al protoboard (D) que alimenta al lector NFC/RFID (B) y al módulo Wi-Fi (C); estos dos últimos componentes tienen conexión adicional con el componente A para recibir y enviar información.

B. Diseño del prototipo IoT de control de apertura remota

Este dispositivo IoT tiene el propósito de simular la apertura de una puerta de forma remota, el dispositivo recibe señales para verificar constantemente el estado de la puerta y determinar que acción debe realizar el dispositivo, el ensamblaje de este prototipo se demuestra en la figura 3.3.

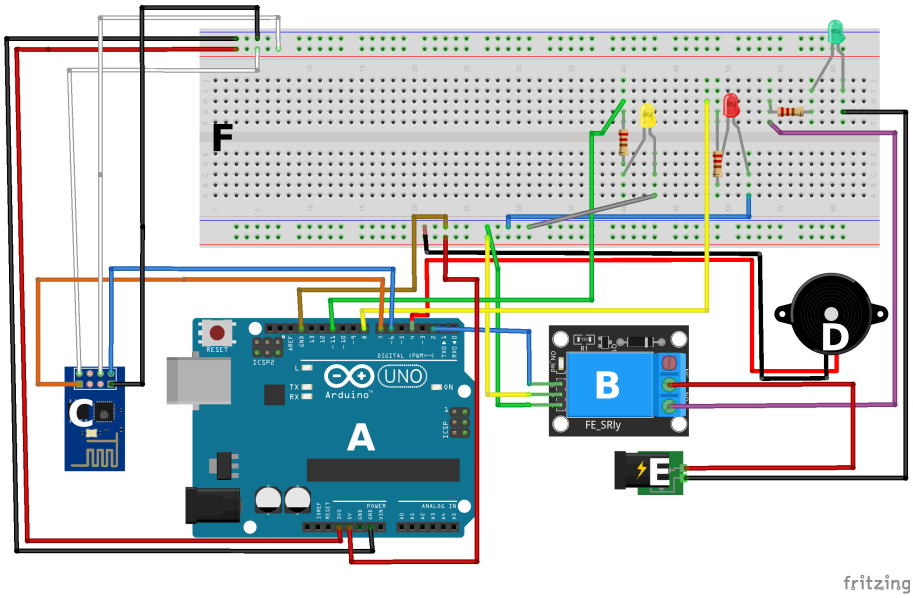


Figura 3.3. Diagrama del prototipo IoT de control de apertura de puerta.

Fuente: Elaboración propia.

La figura 3.3 muestra la interconexión que tiene los diversos componentes con el microcontrolador (A) que distribuye energía y funciones de entrada/salida al protoboard (F), el protoboard posee LEDs (amarillo, rojo) para visualizar el estado de la puerta, provee también sustentación eléctrica al módulo Wi-Fi (C), al buzzer de sonido (D) para notificación auditiva. El protoboard además interconecta el componente A con el relé (B) el cual suministra energía eléctrica externa (E) para encender el LED (verde) que simula la apertura de la puerta.

C. Diseño del prototipo IoT de irrigación

Este dispositivo IoT esta enfocado al control de riego remoto, siendo posible por medio de un higrómetro que permite la lectura de la humedad del suelo y percibir si el suelo requiere humedecer por medio de una bomba de agua sumergible, este diseño se demuestra en la figura 3.4.

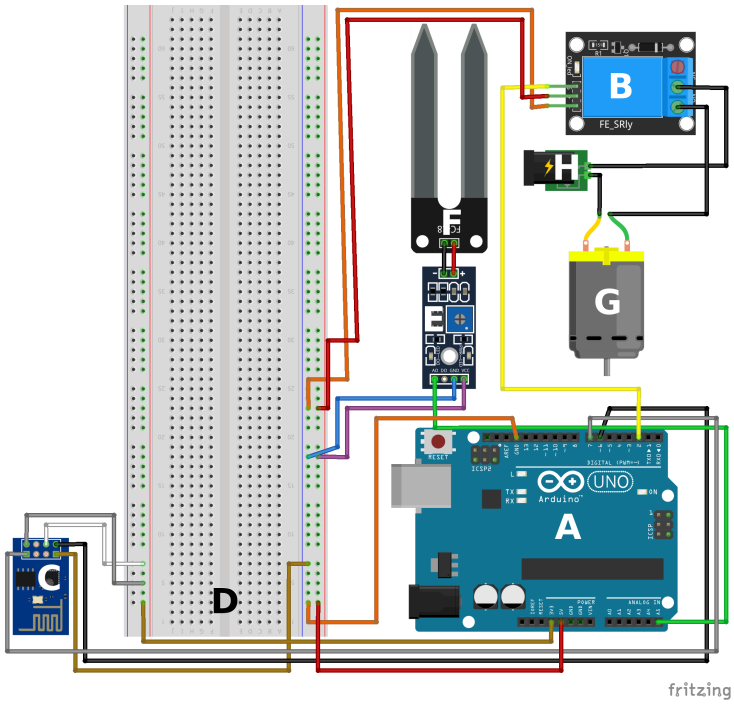


Figura 3.4. Diagrama del prototipo IoT de irrigación.

Fuente: Elaboración propia.

La figura 3.4 muestra los componentes del dispositivo IoT, el microcontrolador Arduino (A) provee energía y funciones de entrada/salida al protoboard (D) que distribuye al componente Wi-Fi (C) y al sensor de humedad (E y F), además el protoboard interconecta el componente A con el relé (B) que envía una señal al componente (G) para suministrar agua por medio de suministro eléctrico (H).

Todos los prototipos IoT descritos anteriormente utilizan el módulo Wi-Fi que se conectan a un servidor web para realizar peticiones HTTP, de esta forma el microcontrolador Arduino puede ejecutar los componentes acoplados en base a la acción requerida, este flujo de ejecución se representa en la figura 3.5.

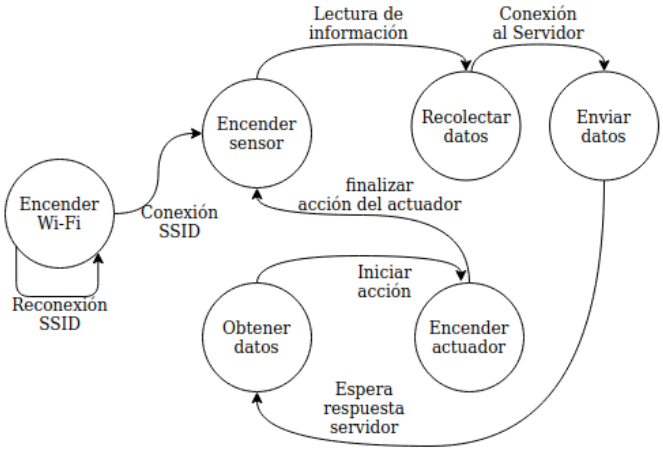


Figura 3.5. Diagrama de maquina de estados de Arduino.

Fuente: Elaboración propia.

La figura 3.5 muestra el diagrama de maquina de estados donde los componentes del dispositivo pasan por diversos estados, inicia por conectarse a una red inalámbrica que permite obtener datos de los servicios proveídos por el servidor, de esta manera los sensores y actuadores pueden ejecutar acciones correspondientes al escenario.

3.2.2. Diseño de software

3.2.2.1. Diseño general del dispositivo IoT

Los dispositivos IoT utilizan un conjunto de instrucciones de código que se encuentran organizados en carpetas, la figura 3.6 muestra la organización de carpetas o paquetes utilizados por el dispositivo IoT, de tal forma que permite la interacción de los componentes electrónicos así como el uso de los servicios.

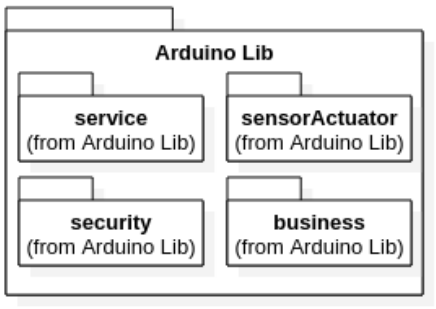


Figura 3.6. Diagrama de paquetes del software del dispositivo IoT.

Fuente: Elaboración propia.

La figura 3.6 muestra el diagrama de paquetes que es utilizado por el dispositivo IoT con respecto al software y hardware, estos paquetes se describen seguidamente:

- **Service**: El paquete proporciona controladores para el componente Wi-Fi asi como comandos para que permiten realizar invocaciones TPC/IP, de esta forma se permite las conexiones con los servicios web.
- **SensorActuator**: El paquete proporciona configuraciones con respecto a los componentes electrónicos acoplados al dispositivo IoT, se utiliza en base al componente actuador o sensor requerido según en cada escenario.
- **Security**: Este paquete proporciona seguridad para el dispositivo IoT, provee un identificador al dispositivo.
- **Business**: Contiene el funcionamiento del dispositivo con respecto al escenario, este paquete utiliza los paquetes Service, SensorActuator y Security para su funcionamiento.

Todos los paquetes son utilizados y denominados dentro de un paquete general llamado Arduino Lib, esta biblioteca funciona como el componente principal de

software para el dispositivo IoT. Se debe mencionar también que el código fuente es grabado en el firmware del dispositivo, siendo su extracción o manipulación del código fuente difícil de modificar.

3.2.2.2. Diseño general de los microservicios

El diseño general esta compuesto por microservicios que se ejecutan como subsistemas autónomos, la comunicación que tienen es por medio de servicios web que proveen y reciben información de dispositivos IoT, de usuarios que interactúan con el dispositivo según el escenario y de usuarios externos que pueden acceder a los servicios correspondientes, la figura 3.7 muestra las comunicaciones correspondientes.

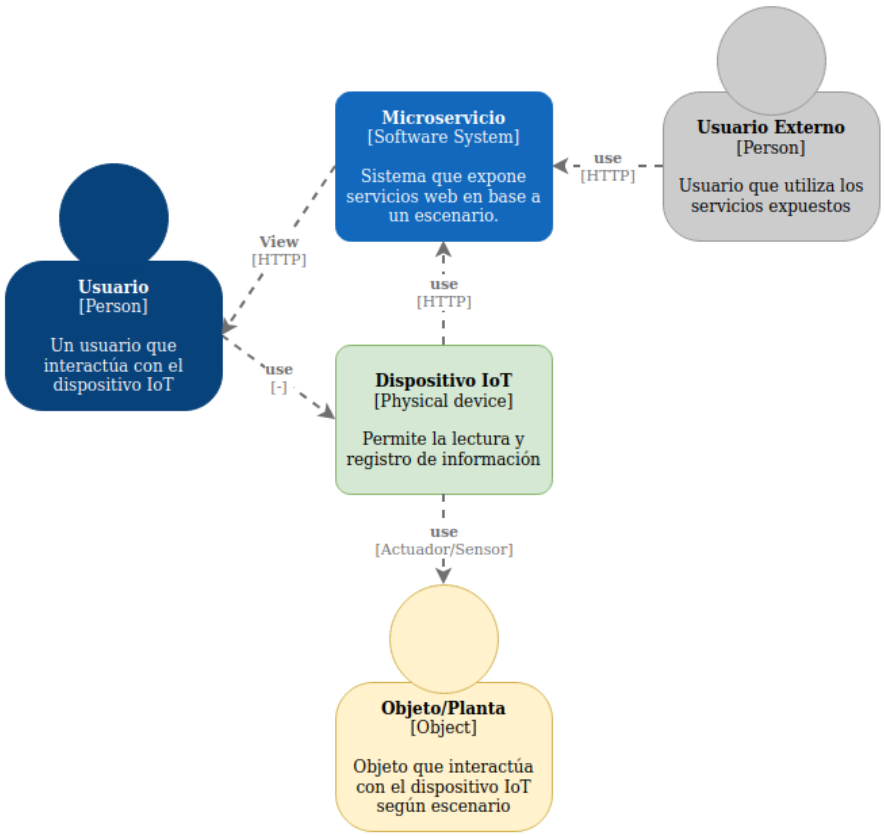


Figura 3.7. Diagrama C4 de contexto.

Fuente: Elaboración propia.

La figura 3.7 muestra la comunicación que tiene el microservicio con el dispositivo IoT y con los usuarios, todas las comunicaciones son posibles por medio del protocolo HTTP, en el caso de la comunicación del objeto físico se utiliza los sensores y actuadores correspondientes.
El diseño general de los microservicios posee tres componentes principales: base de datos, microservicio o conjunto de microservicios según el escenario y una interfaz Swagger UI que documenta los servicios correspondientes, los usuarios y dispositivos se comunican con el sistema por HTTP, la figura 3.8 muestra el diseño respectivo.

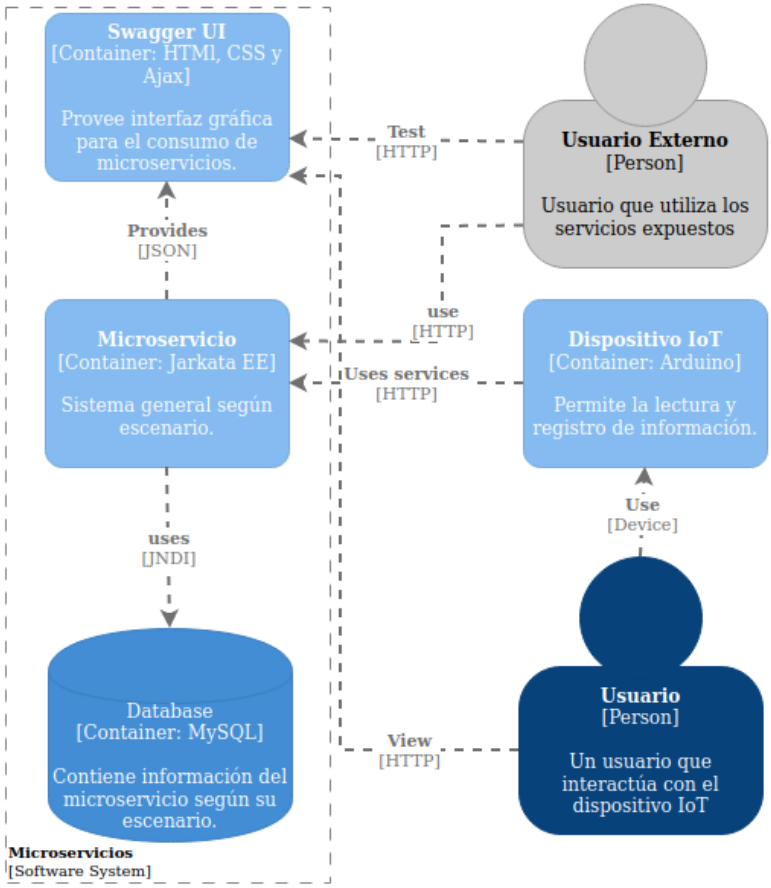


Figura 3.8. Diseño C4 de contenedor del microservicio.

Fuente: Elaboración propia.

La figura 3.8 muestra la composición del microservicio en componentes, el componente principal es el microservicio que provee comunicación con el dispositivo IoT, usuario, el usuario externo y una interfaz gráfica amigable llamada `Swagger UI` que utiliza un archivo JSON que provee el microservicio para probar y documentar los servicios expuestos.

Los microservicios del diseño propuesto se crean de forma granular en base al número de contextos, se toma como referencia el diseño dirigido por el dominio aplicado a cada escenario, de esta manera cada escenario genera un dominio que esta compuesto por uno o varios contextos que finalmente proporciona la información de cuantos microservicios son requeridos.

a) **Registro de asistencia de participantes a eventos:** Este escenario tiene como dominio principal el registro de asistencia de los alumnos por medio del uso de etiquetas NFC a eventos académicos administrados por un coordinador, denotando dos contextos en el dominio como muestra la figura 3.9 y la figura 3.10 como el diseño propuesto para este escenario. De esta forma es posible gestionar y controlar los créditos extracurriculares generados por las asistencias de los alumnos.

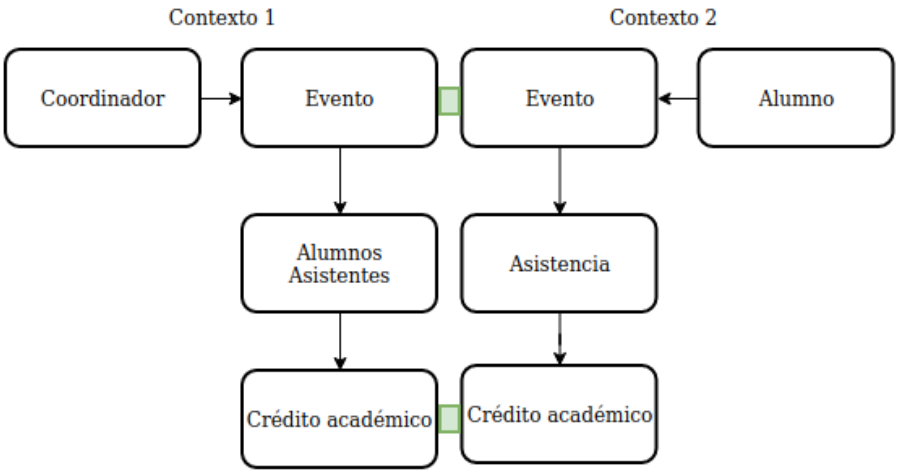


Figura 3.9. Contextos de asistencia a eventos.

Fuente: Elaboración propia.

La figura 3.9 muestra dos contextos dentro del dominio, el primer contexto esta determinado por las funciones específicas necesarias para el coordinador, mientras que el segundo contexto esta centrado en funciones del alumno, estos contextos comparten las entidades de evento y crédito académico.

Una vez determinada la cantidad de contextos en base al dominio se implementan los microservicios con respecto a los contextos, en adición se añade un microservicio que contienen el patrón *Gateway* para definir un único punto de entrada para todos los clientes, la figura 3.10 muestra el diseño propuesto para el dominio correspondiente.

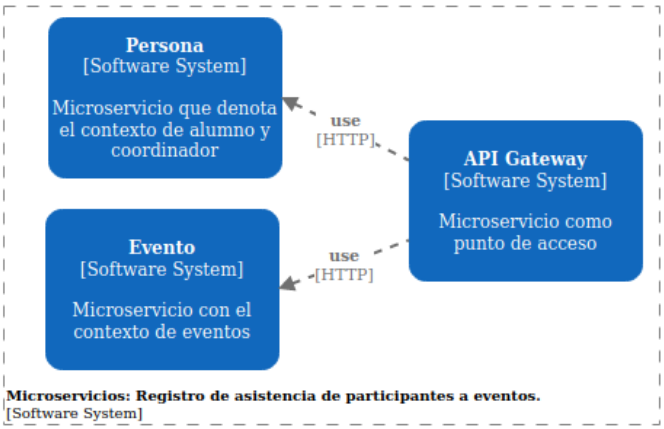


Figura 3.10. Diagrama C4 de contexto de escenario de eventos.

Fuente: Elaboración propia.

La figura 3.10 muestra la cantidad de microservicios utilizados para este escenario, el microservicio `Persona` usa una propia base de datos y esta compuesto de las entidades de coordinador y alumno, el microservicio `Evento` también con su propia base de datos y posee las entidades de

eventos, créditos académicos y asistencias de alumnos. Mientras que el microservicio `API Gateway` funciona como una puerta de enlace principal para controlar el acceso a los servicios que generen los microservicios de Persona y Evento, toda comunicación es por medio de servicios REST.

b) Control de apertura remota: Este escenario esta enfocado hacía el control de apertura de puerta por medio de servicios proveídos de un microservicio como se representa en la figura 3.11.

Figura 3.11. Diagrama C4 de contexto de apertura remota.

Fuente: Elaboración propia.

c) Irrigación de planta: El escenario esta orientado a la supervisión y riego o manual del riego de plantas, por medio del uso de sensores y actuadores que suministren información por medio de servicios, esta se representa por medio de la figura 3.12.

Figura 3.12. Diagrama C4 de contexto del irrigación de planta.

Fuente: Elaboración propia.

La figura 3.11 y 3.12 muestran el diseño del microservicio necesario para los escenarios de: control de apertura remota e irrigación de planta, ambos escenarios no requieren dividirse en contextos debido a que cada escenario solo requiere un microservicio por su mínima complejidad.

3.3. Implementación

3.3.1. Fases de desarrollo e implementación

El siguiente flujo de trabajo propuesto facilita la implementación de software en base al diseño realizado, ya que se utiliza herramientas, bibliotecas y marcos de

trabajo existentes que se alinean a las buenas prácticas empleando un enfoque ágil con respecto al proyecto, este flujo se representa por medio de la figura 3.13.

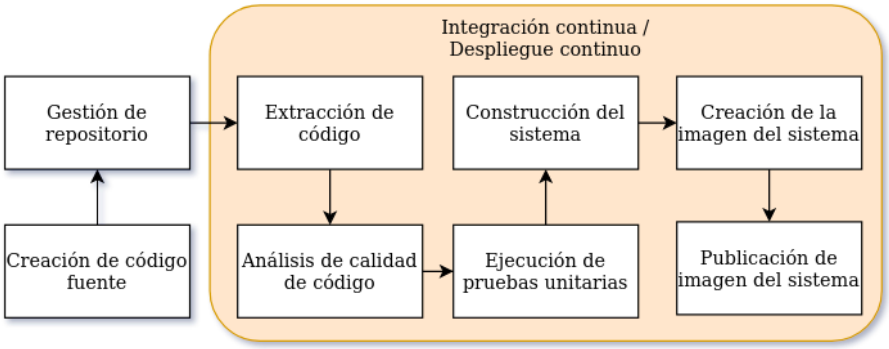


Figura 3.13. Fases de desarrollo e implementación.

Fuente: Elaboración propia.

La figura 3.13 muestra las actividades realizadas para el desarrollo del proyecto, cada fase utiliza herramientas necesarias especificadas previamente en la tabla 2.6.

a) **Creación de código fuente**: Es generado a través del IDE Netbeans 12.0 utilizado por el desarrollador, adicionado con el componente JRebel el cual permite aplicar cambios en el código de aplicaciones Jakarta EE sin realizar despliegues continuos de forma manual.

b) **Gestión de repositorio**: Esta encargado del almacenamiento de código generado por el desarrollador, por medio de la herramienta Git que permite la gestión de versiones y ser enviado a GitLab.

Existe un servidor de automatización Jenkins que permite la integración continua y entrega continua, permite agilizar la entrega del producto de software por medio de integraciones y constantes entregas a un servidor público.

c) **Extracción de código**: El servidor de automatización Jenkins, a través del uso de credenciales accede a los repositorios para descargar el código fuente almacenado.

d) **Análisis de la calidad del código (QA)**: Jenkins utiliza un complemento para realizar análisis de código estático bajo métricas necesarias por medio de SonarQube el cual se ejecuta en un servidor externo; SonarQube permite detectar complejidad ciclomática, código duplicado, estándares y convenciones de código.

e) **Ejecución de pruebas unitarias**: Estas pruebas son diseñadas en la fase de creación de código fuente por el desarrollador, las pruebas son ejecutadas por el servidor de automatización las cuales deben ser correctas para pasar a la siguiente fase.

f) Construcción del sistema: Esta fase se encarga de construir el sistema con todas las dependencias correspondientes para su funcionamiento, se omite la ejecución de pruebas.

g) Creación de la imagen del sistema: El servidor descarga y crea los archivos necesarios para crear un contenedor del sistema por medio del uso de docker.

h) Publicación de imagen del producto: Es la fase final donde la imagen docker del sistema creada localmente es publicada en el repositorio público, siendo descargado o utilizado por medio de un cliente docker.

Dentro de las fases de desarrollo descrito anteriormente una de las partes cruciales es la integración y despliegue continuo, esta compuesto por los servidores de contenedores virtualizados en Docker, se representa por medio de la figura 3.14.

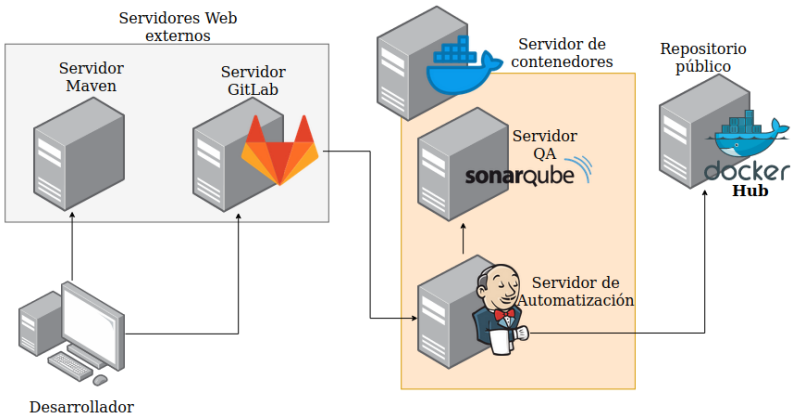


Figura 3.14. Vista de física del uso de servidores.

Fuente: Elaboración propia.

La figura 3.14 muestra la cantidad de servidores usados durante las fases de implementación, el desarrollador utiliza un servidor externo para descargar las bibliotecas desde el servidor Maven, luego es enviado al repositorio externo GitLab; realizado el `commit` en el repositorio, el servidor de automatización realiza las fases anteriormente descritas utilizando el servidor QA, ambos servidores estan en contenedores de un servidor Docker. Finalmente el servidor de automatización publica las imágenes de software en el repositorio de Docker Hub.

Todos los microservicios construidos por el servidor de automatización son colocados en contenedores para ser ejecutados luego de la liberación del producto, siendo posible por medio de la descarga los contenedores en Docker Hub, estos contenedores se comunican con el cliente por medio de peticiones HTTP, como se observa en la figura 3.15.

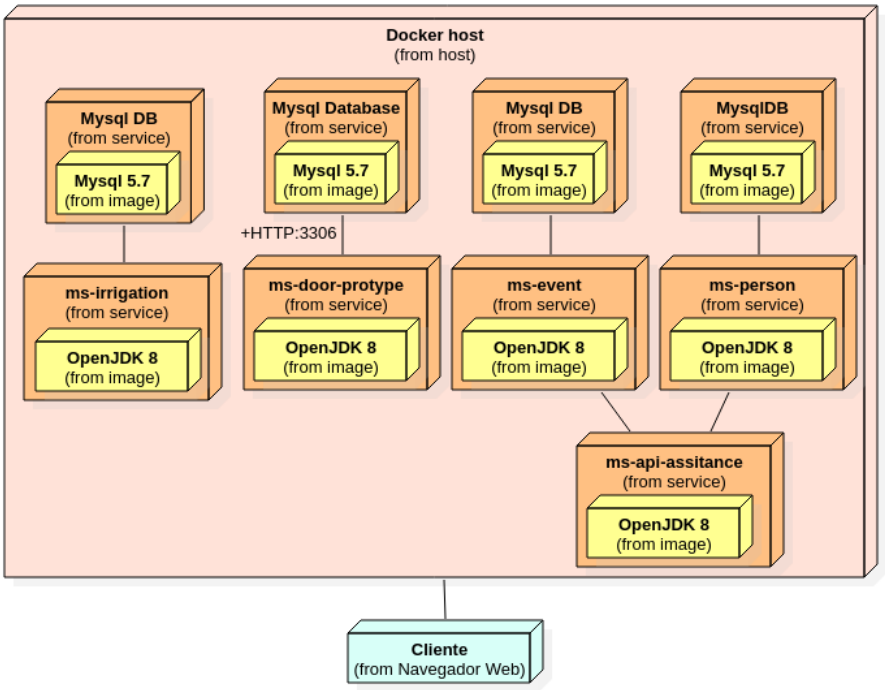


Figura 3.15. Diagrama de despliegue de contenedores del sistema general.

Fuente: Elaboración propia.

La figura 3.15 muestra el diagrama de despliegue del sistema usando docker, que posee los contenedores (services) que utilizan la imagen principal para crear cada microservicio (image), estos contenedores son accedidos desde un cliente (Navegador Web) web, herramientas de peticiones HTTP o directamente para ser utilizados por sistemas de terceros.

Cada microservicio utiliza un propio servidor embebido el cual no requiere realizar instalación, configuración o administración externas tales como: desplegar la aplicación dentro del servidor, configuraciones de base de datos, configuraciones de puerto, etc. Permitiendo realizar despliegues individuales así como la conexión a su respectiva base de datos en caso de que lo requiera, la figura 3.16 muestra el diagrama de despliegue de un microservicio utilizando un contenedor Docker.

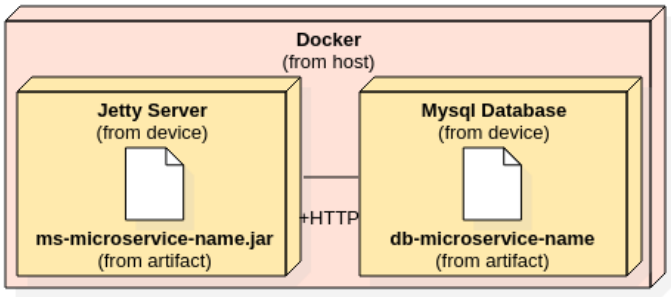


Figura 3.16. Diagrama de despliegue de un microservicio.

Fuente: Elaboración propia.

La figura 3.16 muestra el diagrama de despliegue de un microservicio en Docker,

el microservicio requiere un servidor Jetty embebido para desplegar y una base de datos MySQL para su ejecución, Docker crea los contenedores que son ejecutados en su propio entorno, ambos contenedores se comunican por medio de peticiones HTTP.
Los componentes necesarios para los microservicios están cohesionados con las tecnologías correspondientes, de esta manera se puede reemplazar cada componente (microservicio) rápidamente cuando exista una nueva versión o se requiera realizar cambios.

3.3.2. Arquitectura general de microservicios

La construcción de cada microservicio esta definido según las especificaciones de las historias de usuario, compartiendo características similares en estructura e implementaciones tecnológicas necesarias. En base al diseño general propuesto de los microservicios, la figura 3.17 sigue la misma estructura utilizando las tecnologías correspondientes.

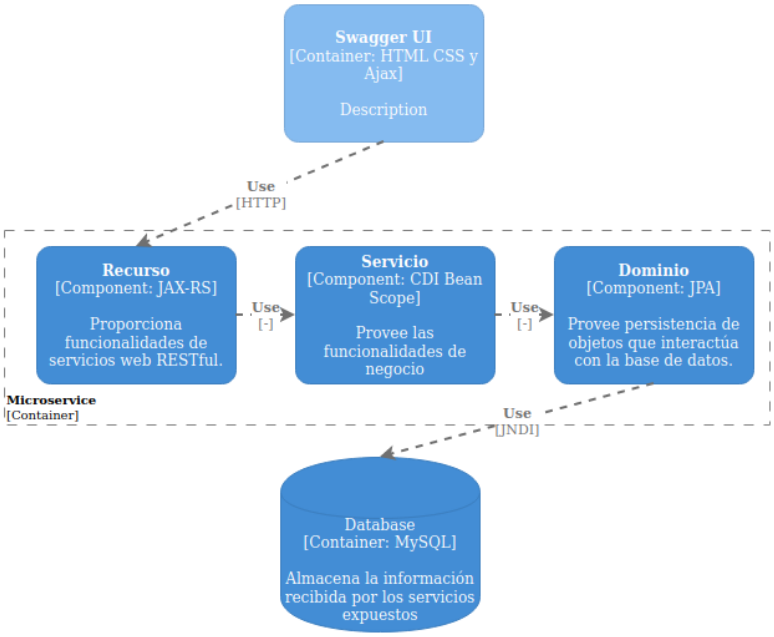


Figura 3.17. Diagrama C4 de componentes.

Fuente: Elaboración propia.

La figura 3.17 muestra la composición técnica general de cada microservicio con respecto al diseño general, esta composición es descrita seguidamente:

- **Swagger-UI**: Tiene como objetivo generar documentación de cada servicio RESTful creado en JAX-RS, por medio de anotaciones CDI.
- **Recurso/JAX-RS**: Esta encargado de proveer soporte de creación de servicios web RESTful.
- **Servicio/ CDI o Bean Scope**: Permite la inversión de control de dependencia, es decir crea el recurso cuando sea necesario y no desde la ejecución inicial del microservicio.

- **Dominio/Java Persistence API (JPA)**: Esta encargado de persistir objetos a la base de datos en vez de realizar consultas directamente.
- **Base de datos MySQL**: Es el motor de base de datos principal que existe por cada microservicio, el cual se dedica a almacenar los datos de cada microservicio.

Cada escenario aplica la misma estructura de los microservicios a nivel granular en base al diseño propuesto, todo microservicio se ejecuta de forma autónoma e individual el cual se aplica en los dos casos siguientes:

- **Microservicios múltiples**: Se crea un microservicio utilizado como *Gateway* e implementa el patrón Anti-Corruption Layer (ACL) que provee de autorización y autenticación usando JSON Web Token (JWT), de esta forma se unifica todos los microservicios en una salida como se demuestra en la figura 3.18 (a), tolerando fallos en caso de caída de algún microservicio.
- **Microservicios individuales**: Los servicios creados se comunican directamente sin necesidad de alguna interfaz, se demuestra mediante la figura 3.18 (b).

(a) Microservicios múltiples. (b) Microservicio individual.

Figura 3.18. Arquitecturas de microservicios en base a los escenarios.

Fuente: Elaboración propia.

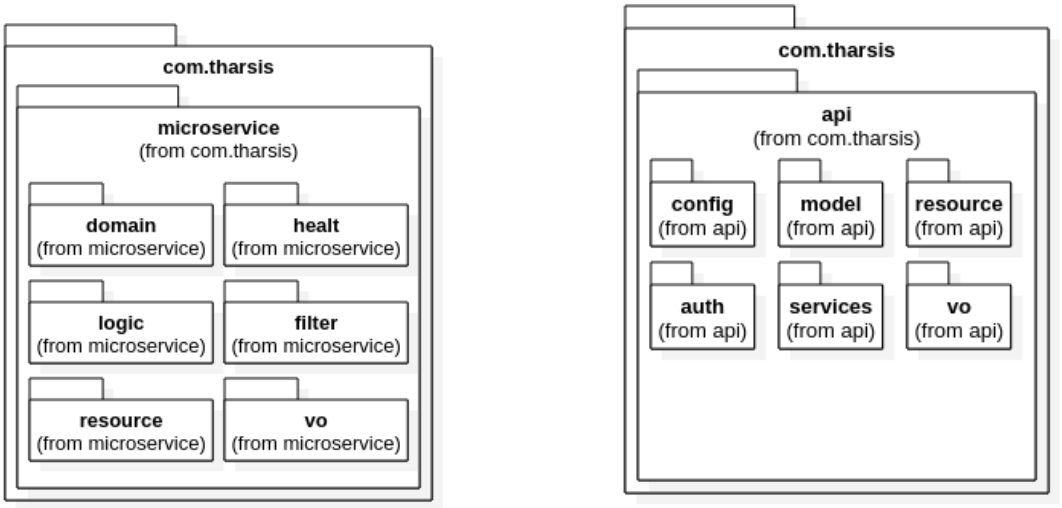
La figura 3.18 muestra la implementación usada en base al escenario propuesto, la figura (a) es aplicada para el escenario de registro de asistencia por NFC, mientras tanto la figura (b) es utilizada en los escenarios de irrigación y control de apertura remota.

3.3.3. Implementación

3.3.3.1. Implementación de código de microservicios

La implementación de los microservicios se codifica de manera individual en base a los dominios diseñados y desplegados en su propio servidor como fue descrito con anterioridad, cada microservicio sigue una estructura explicado previamente en conjunto con la figura 3.18.

El proyecto estructura todos los microservicios en paquetes con nombres específicos para organizar y categorizar la lógica de cada componente que forma parte de un microservicio respectivo como se ve en la figura 3.19.



(a) Diagrama de paquetes de un microservicio general.

(b) Diagrama de paquetes de un microservicio como gateway.

Figura 3.19. Diagrama de paquetes de los microservicios.

Fuente: Elaboración propia.

La figura 3.19 muestra el diagrama de paquetes que estructura los archivos base de cada microservicio, la figura 3.19(a) muestra la composición de los microservicios creados individualmente, mientras que la figura 3.19(b) muestra la estructura si un microservicio es utilizado como puerta de enlace, todos los paquetes se detallan seguidamente:

- **Paquete domain**: Contiene las clases de persistencia y consultas que se realiza a la base de datos.
- **Paquete logic**: Contiene la lógica del contexto del dominio, utiliza clases del paquete domain.
- **Paquete resource**: Contiene las clases que proporcionan creación de servicios web RESTful por medio de inyecciones de dependencias, utiliza el paquete logic.
- **Paquete healt**: Proporciona el estado de salud del microservicio, siendo posible por medio del consumo de un propio servicio RESTful, de tal forma que permite saber si esta activo o no el microservicio.
- **Paquete vo**: Son clases llamadas objetos valor, proporcionan accesibilidad y transferencia de datos entre paquetes.
- **Paquete config**: Contiene clases de configuración de rutas para acceder a los microservicios con el tipo de diseño general.

- **Paquete auth**: Paquete que proporciona seguridad mediante el uso de tokens JWT y anotaciones con roles que es usado del paquete logic y paquete resource respectivamente.

El núcleo de cada microservicio es factible por medio de las anotaciones que permiten la inyección de dependencia como demuestra la figura 3.20, el código fuente permite realizar las configuraciones necesarias rápidamente que se ajustan a la implementación de los microservicios con respecto al diseño propuesto.

```
1 @ApplicationPath("api")
2 @SwaggerDefinition(info =
3     @Info(
4         title = "API Assistance",
5         version = "v1.0.0"))
6 @CrossOrigin
7 public class Resource extends Application{
8
9 }
```

Figura 3.20. Código principal para habilitar servicios web.

Fuente: Elaboración propia.

La figura 3.20 muestra la configuración inicial que tiene el microservicio `ms-api-assitance`, `ms-door-prototype` y `ms-irrigation`, la clase `Resource` hereda de `Application` que permite `JAX-RS` (permite habilitar las funciones RESTful), y las anotaciones que permiten las configuraciones siguientes:

- **@ApplicationPath**: Especifica la ruta raíz del microservicio que permite acceder a cada servicio creado.
- **@SwaggerDefinition**: Define que el microservicio utilizará Swagger para visualizar y documentar los servicios expuesto, este elemento utiliza la anotación `@Info` para describir el servicio.
- **@Info**: Especifica los metadatos que describen la información del microservicio, es utilizado en Swagger como documentación del servicio, este elemento se visualiza para los clientes que accedan a la interfaz gráfica de Swagger o el archivo JSON generado.
- **@CrossOrigin**: Es una medida de seguridad que controla el acceso a consumir los recursos de un servidor por medio de un cliente, verificando que se acceda desde un origen permitido.

Para el caso del microservicio `ms-api-assitance` requiere especificar los servicios de enrutamiento para funcionar como un cliente de los microservicios `ms-person` y `ms-event`, la anotación `@RegisterRestClient` (línea de código 2) y `@Dependent` (línea de código 3) que muestra la figura 3.21 permite habilitar funcionalidades como cliente.

```java
@Path("register")
@RegisterRestClient
@Dependent
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
public interface EventRegisterService {

    @POST
    @Path("add")
    public void saveRegister(EventRegisterVO registerVO);

    @GET
    @Path("list-event-by-student/{idStudent}")
    public List<Object> getAllEventRegisterByStudent(
       @PathParam("idStudent") Integer idStudent);

    @GET
    @Path("total-credits/{idStudent}")
    public BigDecimal getSumCreditsByStudent(@PathParam("
       idStudent") Integer idStudent);

    @GET
    @Path("list-student-by-event/{idEvent}")
    public List<Number> getStudentByEvent(@PathParam("
       idEvent") Integer idEvent);

    // more code...
}
```

Figura 3.21. Interfaz cliente de ms-api-assitance

Fuente: Elaboración propia.

La figura 3.21 muestra una interfaz cliente `EventRegisterService` que el microservicio `API-Gateway` utiliza, este servicio funciona como cliente consumidor con respecto al microservicio `ms-event`, las anotaciones `@RegisterRestClient` y `@Dependent` permiten registrar y utilizar los métodos implementados por el microservicio `ms-event`. Los servicios que requieran utilizar estos métodos deben añadir la anotación `@RestClient`.

Además del consumo y exposición de servicios web, `API-Gateway` define la autorización y autenticación para acceder al consumo de los servicios, siendo posible por medio de un inicio de sesión que valida la credencial y rol del usuario, las credenciales son verificadas el microservicio que devuelve una respuesta generando un token de acceso `JWT` para acceder al consumo de los servicios según el rol del usuario, la figura 3.22 muestra el código donde un método se encuentra seguro.

```
@Path("event-register")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
@RequestScoped
@Api(tags = "Event register")
public class EventRegisterResource {

    @RestClient
    private EventRegisterService eventRegisterService;

    @RestClient
    private StudentService studentService;


    @PostConstruct
    private void init() {
        studentService = PathService
                .clientBuilder(Route.PERSON)
                .build(StudentService.class);

        eventRegisterService = PathService
                .clientBuilder(Route.EVENT)
                .build(EventRegisterService.class);
    }

    @GET
    @Path("student/list-assitance-by-studentId/{id}")
    @Secured({Role.ORGANIZER, Role.STUDENT})
    public Response listEventByStudent(@PathParam("id")
       Integer id) {
        if (eventRegisterService.
           getAllEventRegisterByStudent(id).isEmpty()) {
            return Response.status(Response.Status.
               NO_CONTENT).build();
        } else {
            return Response.ok(eventRegisterService.
               getAllEventRegisterByStudent(id)).build();
        }
    }
 // More code...
}
```

Figura 3.22. Exposición de servicios

Fuente: Elaboración propia.

La figura 3.22 muestra la clase `EventRegisterResource` las anotaciones `@PostConstruct` y `@RegisterClient` para inicializar, acceder y consumir respectivamente los servi-

cios expuestos de `ms-event` y `ms-person`, el método `listEventByStudent` utiliza una anotación `@Secured` que provee seguridad para proteger al método para ser solo utilizado por el usuario y rol correspondiente.

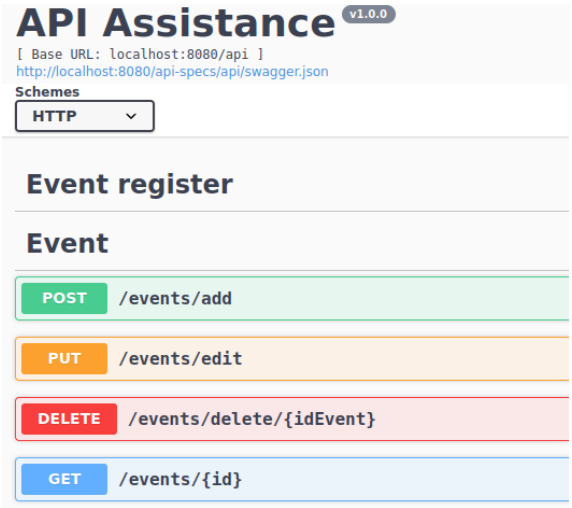


Figura 3.23. Interfaz gráfica de Swagger documentado los servicios.

Fuente: Elaboración propia.

La figura 3.23 muestra la interfaz gráfica generada por `Swagger`, las figuras 3.20, 3.21 y 3.22 mostrado anteriormente son implementaciones realizadas en `ms-api-assitance` que denota código fuente similar en los microservicios de `ms-person`, `ms-event`, `ms-irrigation` y `ms-door-prototype` que utilizan `Swagger` para visualizar los servicios expuestos, las anotaciones `@SwaggerDefinition` e `@Info` mostrado en la imagen 3.20 en conjunto con una configuración inicial de un archivo de extensión `.yaml` que genera un archivo JSON y visualiza las rutas de servicio habilitadas a través de la anotación `@Api` demostrado en la figura 3.22 línea 5.

Tabla 3.6. Técnologías usadas en la implementación.

Tecnología	Descripción
Jakarta EE	Base de código para desarrollar software
Jenkins, GitLab, SonarQube	Integración y entrega continua
Swagger	Generación de documentación RESTful
Docker	Creación de contenedores
Arquillian	Contenedor de pruebas
REST assured	Validación y pruebas de servicios en java

Fuente: Elaboración propia.

La tabla 3.6 muestra la mayor parte de las tecnologías utilizadas para la implementación de los microservicios y las herramientas para la integración continua y entrega continua.

```
version: '3'
services:
  docker-mysql:
    restart: always
    container_name: docker-mysql
    image: mysql:5.7
    environment:
      MYSQL_DATABASE: ms-irrigation
      MYSQL_USER: root
      MYSQL_ROOT_PASSWORD: admin
      MYSQL_PASSWORD: admin
      MYSQL_ROOT_HOST: '%'
    volumes:
      - ./db:/docker-entrypoint-initdb.d
    networks:
            ms-iot-net:
                ipv4_address: 172.28.1.1
    ports:
      - "6033:3306"
    healthcheck:
            test: ["CMD", "mysqladmin" ,"ping", "-h", "
               localhost"]
            timeout: 20s
            retries: 10
  ms-irrigation-app:
    image: christianlzap/ms-irrigation:1.0
    restart: on-failure
    networks:
            ms-iot-net:
                ipv4_address: 172.28.1.2
    ports:
      - 8086:8086
    environment:
      WAIT_HOSTS: mysql:6033
    depends_on:
      - docker-mysql
networks:
    ms-iot-net:
        ipam:
            driver: default
            config:
                - subnet: 172.28.0.0/16
```

Figura 3.24. Código para contener un microservicio

Fuente: Elaboración propia.

La figura 3.24 muestra el código de configuración usadaba para contener el

microservicio y asignar una base de datos correspondiente usando Docker por medio del archivo de configuración `Dockerfile` y `Docker Compose`.

3.4. Pruebas

3.4.1. Pruebas de microservicios

Cada microservicio con excepción de `ms-api-assitance` tiene pruebas unitarias construidas en base a los elementos del Backlog respectivo, el resultado de las pruebas es probar la funcionalidad de los servicios expuestos de cada dominio como se muestra en la figuras 3.27 y 3.28, las pruebas son construidas por medio de las siguientes herramientas:

- **Arquillian**: Es un contenedor de pruebas que permite ejecutar funcionalidades de Jakarta EE.
- **REST assured**: Es una herramienta que permite hacer pruebas HTTP, estas pruebas son realizadas a los servicios o *endpoints*.

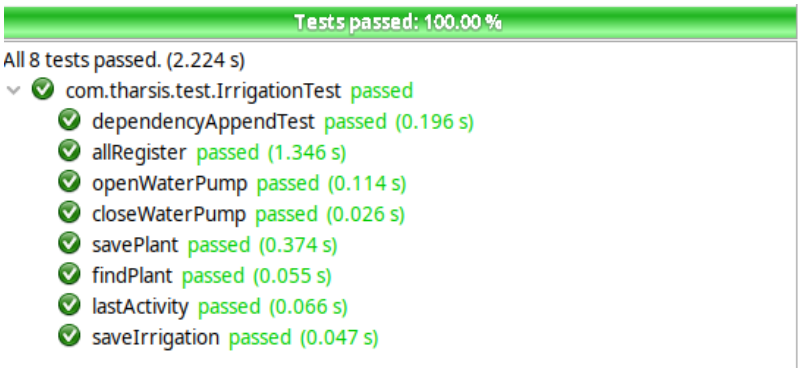


Figura 3.25. Ejecución de pruebas del microservicio ms-irrigation.

Fuente: Elaboración propia.

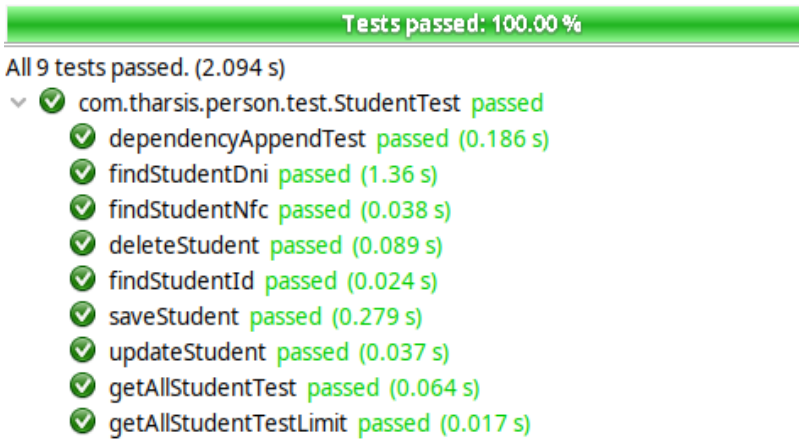


Figura 3.26. Ejecución de pruebas de alumno del microservicio ms-person.

Fuente: Elaboración propia.

La figura 3.25 y 3.26 muestra las ejecuciones exitosas de las pruebas realizadas en los microservicios, las pruebas son ejecutadas primero por el desarrollador y posteriormente es ejecutado por el servidor de integración y entrega continua.

```
@RunWith(Arquillian.class)
public class PersonTest {
    @Deployment
    public static JavaArchive createDeploy() {
        return ShrinkWrap.create(JavaArchive.class)
                .addClass(Resource.class)
                .addClass(Person.class)
                .addClass(PersonLogic.class)
                .addClass(LoginVO.class)
                .addClass(Role.class)
                .addClass(Token.class)
                .addClass(PersonResource.class)
                .addAsManifestResource(EmptyAsset.INSTANCE,
                    "beans.xml")
                .addAsResource("config.yml", "config.yml")
                .addAsManifestResource(new FileAsset(new
                   File("src/test/resources/META-INF/
                   persistence.xml")),
                        "persistence.xml");
    }
    // More code...
}
```

Figura 3.27. Interfaz cliente de ms-event

Fuente: Elaboración propia.

La figura 3.27 muestra la configuración inicial en Arquillian, se realiza la configuración incluyendo las clases que interactúan con los servicios y verificar las funcionalidades JAX-RS, posteriormente se ejecutan las pruebas necesarias como se muestran en la figura 3.28.

```
    // More code...
    @Test
    @RunAsClient
    public void allRegister() {
        when().get("v1/log-plant-service/all-registers")
                .then().statusCode(200);
    }
    // More code...
```

Figura 3.28. Código de prueba de ms-irrigation

Fuente: Elaboración propia.

Capítulo 4

Resultados y Discusión

4.1. Resultados

La arquitectura de software basada en microservicios es evaluada y verificada por el método Architecture Trade-off Analysis Method (ATAM) que permite la verificación, validación y evaluación de la arquitectura empleada en base a los atributos de calidad descritos en el segundo capítulo, se usa una matriz o árbol de atributos de utilidad para determinar que atributos de calidad cumplen con la arquitectura propuesta.

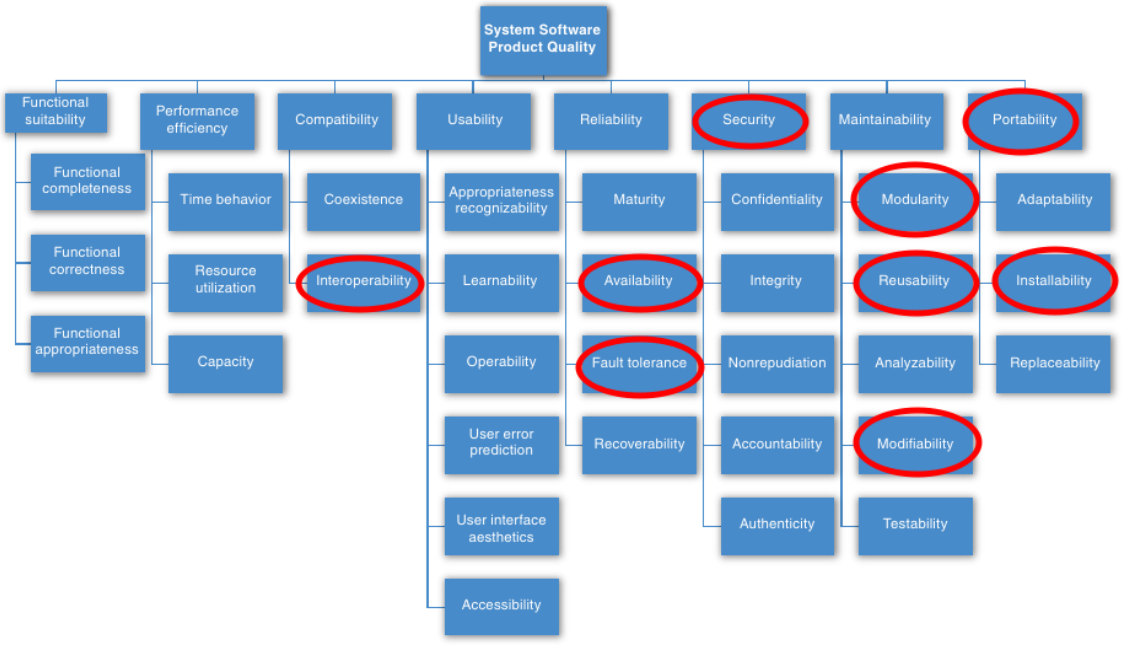


Figura 4.1. Atributos de calidad seleccionados para evaluar al arquitectura.

Fuente: [22].

La figura 4.1 muestra los atributos de calidad de un producto de software, son señalados con un círculo rojo los atributos de calidad considerados en la implementación y uso de la arquitectura basada en microservicios, que con la ayuda de herramientas o tecnologías utilizadas pueden evaluarse por su utilidad.

Para cumplir los atributos de calidad definidos en la arquitectura de software propuesta, existen componentes arquitectónicos que por medio de la tecnología proporcionan cumplir un atributo de calidad en base a un escenario, las tecnologías utilizadas como componentes se describen en la tabla 4.1

Tabla 4.1. Matriz de atributos de utilidad ATAM.

Atributo	Componente arquitectónico	Tecnología empleada	E1[1]	E2[2]	E3[3]
Modificabilidad	Microservicios	Jakarta EE	Sí	Sí	Sí
Instalación	Contenerización de microservicios	Docker, Docker-Hub	Sí	Sí	Sí
Reusabilidad	Microservicios	Docker	Sí	Sí	Sí
Portabilidad	Contenerización de microservicios	Jakarta EE, Docker	Sí	Sí	Sí
Tolerancia a fallos	Microservicios (Circuit Breaker)	Jakarta EE	Sí	Sí	Sí
Disponibilidad	Microservicios, contenerización de aplicaciones	Jakarta EE, Docker	Sí	Sí	Sí
Seguridad	Microservicios (Autorización y autenticación)	Jakarta EE	Sí	No	No

Fuente: Elaboración propia.

La tabla 4.1 muestra los atributos de calidad considerados para la validación de la arquitectura, en gran parte los componentes arquitectónicos cumplen con cada atributo de calidad por medio de la tecnología usada y probada en cada escenario, se describe seguidamente el componente en relación al atributo de calidad.

Debido a que cada microservicio esta diseñado como un componente individual y es desplegado en un servidor embebido, puede realizarse modificaciones sin afectar las funcionalidades de otros componentes, cumpliendo con el atributo de modificabilidad. Además Jakarta EE provee características de inyección de dependencia o inversión de control.

El atributo de instalación se comprueba por medio de los microservicios que son construidos como ejecutables simples, luego son empaquetados en un contenedor con Docker y publicados en Docker-Hub, de tal forma que es fácil extraer, instalar y ejecutar en un servidor con dependencias mínimas.
Con respecto al atributo de reusabilidad, los microservicios pueden volver a

[1]Escenario de asistencia de participantes a eventos
[2]Escenario de irrigación de planta
[3]Escenario de control de apertura remota

ser utilizados como componentes de integración en otros sistemas, en caso de requerir cambios en las funcionalidades pueden conservar la arquitectura que utiliza el microservicio, adaptándose a las necesidades que requieran los sistemas que gestionen el microservicio.

Los microservicios son virtualizados en contenedores Docker y publicados en Docker-Hub, permitiendo la facilidad de ejecutarse en cualquier plataforma que tenga soporte con la tecnología Docker, minimizando y encapsulando todas las dependencias requeridas por los microservicios, por lo tanto, cumple con el atributo de portabilidad.

En el caso del atributo de calidad de tolerancia a fallos, la arquitectura esta enfocada a verificar que las funcionalidades más críticas puedan responder y seguir funcionando en caso de que exista un error, esto es posible por medio de anotaciones que permite Jakarta EE.

El atributo de disponibilidad se cumple por medio de dos factores, primero cada microservicio posee un chequeo de salud que indica el estado del microservicio y segundo el microservicio a ser virtualizado Docker verifica repetidamente que disponibilidad se de un 99.99 %.

Finalmente, con el atributo de seguridad, la arquitectura de software hace énfasis en la autorización y autenticación, donde las anotaciones de seguridad de la tecnología Jakarta EE protegen los recursos expuestos por los microservicios, siendo utilizados por los usuarios permitidos del sistema. En caso de los escenarios E2 y E3 no cumplen con este atributo, debido a la simplicidad de las especificaciones del interesado y las consideraciones convenientes durante el desarrollo para estos escenarios.

Los microservicios implementados con la arquitectura de software utilizados en cada escenario, demuestran que cada atributo en relación con la tecnología empleada cumple con cada atributo de calidad propuesto por la arquitectura de software.

Para evaluar el funcionamiento y la fiabilidad de los dispositivos IoT, se realiza un conjunto de pruebas de cada dispositivo en relación con cada microservicio concerniente a su escenario, todas las pruebas realizadas son posibles por medio de la recolección de información que son almacenados en bases de datos individuales accedidos por los microservicios, comprobando la intercomunicación de datos entre el dispositivo IoT y el microservicio. Las pruebas realizadas son las de verificación de humedad, pruebas de apertura remota y registro de asistencia mediante etiquetas NFC representados por las figuras 4.2, 4.3 y 4.4 respectivamente.

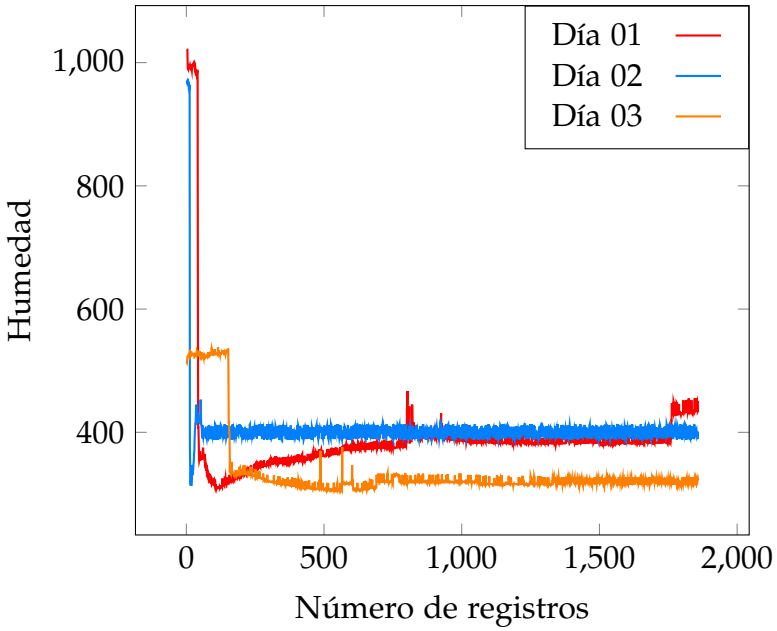


Figura 4.2. Niveles de humedad registrado por el dispositivo IoT

La figura 4.2 muestra el registro de humedad realizada por el dispositivo IoT en el escenario de riego, se registraron 1858 datos que denotan la humedad de la tierra, el mayor valor registrado es de 1021 el cual significa que el estado de la tierra es árida y como menor valor 305 que indica humedad en la tierra. El registro se realizó por tres días donde el día 03 toma valores debajo de los días 01 y 02 debido a factores climáticos.

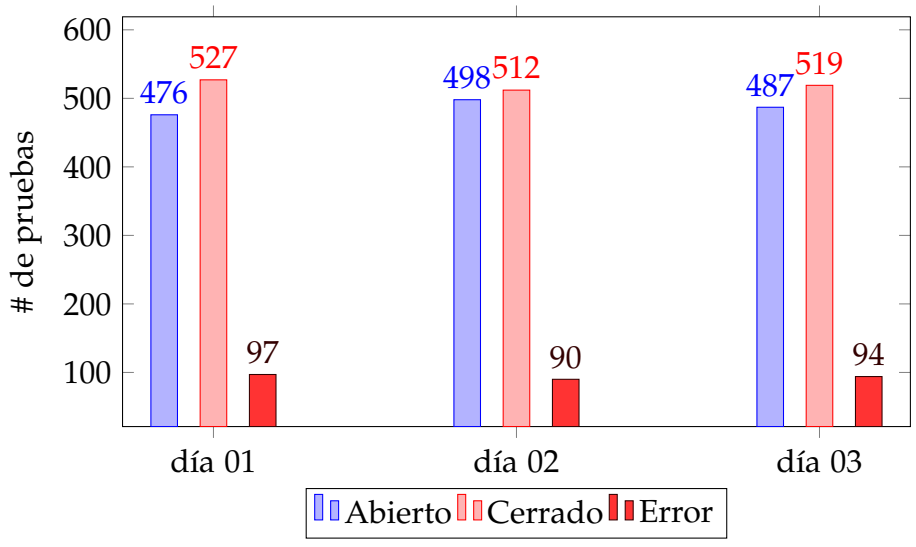


Figura 4.3. Resultado de ejecuciones del dispositivo IoT

La figura 4.3 muestra el número de pruebas realizado en el dispositivo prototipo, cada día se realizó 550 pruebas de apertura y cierre respectivamente; el día 01 denota 97 registros erróneos del total de pruebas que representa un 17.63 % de margen de error, el día 02 denota 90 registros fallidos que señala un 16.40 % y finalmente el día 03 con 94 registros erróneos que señala el 17.09 % de error. Los

errores generados son debido a que el dispositivo IoT no interactúa con el objeto físico a causa del componente Wi-Fi que no hace una lectura de los servicios, no obstante esto no significa que el dispositivo deja de funcionar o requiere un reinicio.

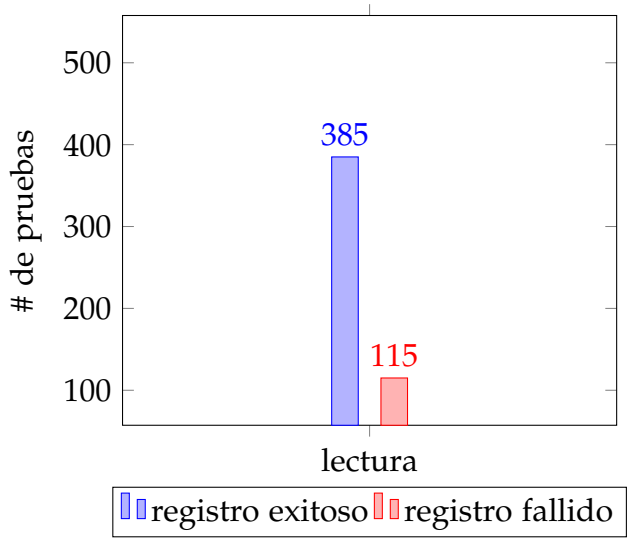


Figura 4.4. Resultado de las ejecuciones de lectura NFC/RFID

La figura 4.4 muestra la ejecución de 500 lecturas de etiquetas NFC/RFID donde 115 de ellas no fueron registrados y que representa el 23 % de margen de error. Al igual que el escenario de control de apertura remota, el error se detecta al enviar los datos capturados por medio del componente Wi-Fi.

Conclusiones

- La arquitectura de software basada en microservicios diseñada e implementada permite trabajar adecuadamente con los dispositivos IoT utilizando tecnologías como Jakarta EE, Docker, integración y entrega continua que tienen una funcionalidad significativa para los dispositivos IoT debido a su modificabilidad, disponibilidad y portabilidad.
- Las tecnologías identificadas permiten diseñar e implementar la arquitectura basada en microservicios, generando el número de microservicios necesario por cada escenario en base al diseño dirigido por el dominio y siendo implementado por la tecnología Jakarta EE, Mysql, Docker para el desarrollo y Docker, GitLab, Jenkins y SonarQube para la integración y entrega continua.
- La arquitectura basada en microservicios proporciona servicios web RESTful, únicamente por medio del protocolo HTTP para compartir información entre sistemas de terceros y microservicios propios. No obstante no se considera otro tipo de comunicación para interactuar con otros sistemas.
- La implementación y desarrollo de los dispositivos IoT permitieron identificar que los componentes Arduino UNO, Módulo Wi-fi ESP8266 y otros componentes electrónicos permitieron que el dispositivo pueda comunicarse con los servicios web, sin embargo se debe de escribir la petición, la cabecera y el cuerpo HTTP para enviar y recibir datos.
- La comprobación de la comunicación del dispositivo IoT mediante el uso de microservicios es realizada por un plan de pruebas hechas en tres días, el primer escenario de riego muestra la variación de humedad de la tierra en base al tiempo, el segundo y tercer escenario tiene un 82.96 % y 77 % respectivamente de comunicación fiable con los microservicios.

Recomendaciones

- Utilizar otros componentes de hardware que permita la comunicación en el dispositivo IoT y facilite la comunicación en protocolos HTTP, el cual puede obtener un mejor rendimiento y estabilidad de comunicación.
- Considerar otros tipos de comunicación como MQTT, CoAP o XMPP que tiene especificaciones simples para comunicarse con microservicios, de tal forma que se puede obtener los datos con un mínimo margen de error.
- Incluir dentro de la arquitectura de microservicios un protocolo de mensajería Advanced Message Queuing Protocol (AMPQ) para comunicarse con dispositivos IoT.

Trabajos Futuros

- Profundizar la investigación de dispositivos IoT y la comunicación con microservicios en diversos protocolos de comunicación para establecer una forma de interconexión en smart cities.
- Investigar la integración de nuevos microservicios que pueda procesar grandes cantidades de información para analizar los datos generados por los dispositivos IoT que sirva como plataforma de Big Data.
- Considerar la investigación de plataformas construidas bajo microservicios que adicionen Deep Learning y Machine Learning para automatizar el control de los dispositivos IoT.

Bibliografía

[1] M. A. Jarwar, S. Ali, M. G. Kibria, S. Kumar, and I. Chong, "Exploiting interoperable microservices in web objects enabled internet of things," in *Ubiquitous and Future Networks (ICUFN), 2017 Ninth International Conference on*, pp. 49–54, IEEE, 2017.

[2] A. Power and G. Kotonya, "A microservices architecture for reactive and proactive fault tolerance in iot systems," in *2018 IEEE 19th International Symposium on.ª World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*, pp. 588–599, IEEE, 2018.

[3] K. Khanda, D. Salikhov, K. Gusmanov, M. Mazzara, and N. Mavridis, "Microservice-based iot for smart buildings," in *Advanced Information Networking and Applications Workshops (WAINA), 2017 31st International Conference on*, pp. 302–308, IEEE, 2017.

[4] P. Krivic, P. Skocir, M. Kusek, and G. Jezic, "Microservices as agents in iot systems," in *KES International Symposium on Agent and Multi-Agent Systems: Technologies and Applications*, pp. 22–31, Springer, 2017.

[5] K. Thramboulidis, D. C. Vachtsevanou, and A. Solanos, "Cyber-physical microservices: An iot-based framework for manufacturing systems," in *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*, pp. 232–239, IEEE, 2018.

[6] T. Vresk and I. Čavrak, "Architecture of an interoperable iot platform based on microservices," in *Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2016 39th International Convention on*, pp. 1196–1201, IEEE, 2016.

[7] A. Krylovskiy, M. Jahn, and E. Patti, "Designing a smart city internet of things platform with microservice architecture," in *Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on*, pp. 25–30, IEEE, 2015.

[8] B. Butzin, F. Golatowski, and D. Timmermann, "Microservices approach for the internet of things," in *Emerging Technologies and Factory Automation (ETFA), 2016 IEEE 21st International Conference on*, pp. 1–6, IEEE, 2016.

[9] G. Campeanu, "A mapping study on microservice architectures of internet of things and cloud computing solutions," in *2018 7th Mediterranean Conference on Embedded Computing (MECO)*, pp. 1–4, IEEE, 2018.

[10] P. Sridhara, N. Kamath, and S. Srinivas, "A microservices-based smart iot gateway system," in *Smart Intelligent Computing and Applications*, pp. 619–630, Springer, 2019.

[11] S. Kemp, "Digital 2020: 3.8 billion people use social media." [En línea.] Disponible: `https://wearesocial.com/blog/2020/01/digital-2020-3-8-billion-people-use-social-media`, 2020. [Accedido: 11/11/2020].

[12] I. N. de Estadística e Informática INEI, "Perú: Características de las viviendas particulares y los hogares. acceso a servicios básicos." [En línea.] Disponible: `http://www.inei.gob.pe/media/MenuRecursivo/publicaciones_digitales/Est/Lib1538/index.html`, 2018. [accedido: 11/11/2020].

[13] I. N. de Estadística e Informática INEI, "Perú: Tecnologías de información y comunicación en las empresas, 2015." [En línea.] Disponible: `https://www.inei.gob.pe/media/MenuRecursivo/publicaciones_digitales/Est/Lib1482/libro.pdf`, 2015. [accedido: 10/11/2020].

[14] A. Gilchrist, *Industry 4.0*. Apress, 2016.

[15] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: yesterday, today, and tomorrow," in *Present and Ulterior Software Engineering*, pp. 195–216, Springer, 2017.

[16] I. Sommerville, *Software Engineering*. Pearson, 10th ed., 2016.

[17] M. E. B. Eric J. Braude, *Software Engineering: Modern Approaches*. Waveland Press, second ed., feb 2016.

[18] R. Narang, *Software Engineering—Principles and Practices*. McGraw Hill Education, 2015.

[19] R. Kneuper, *Software Processes and Life Cycle Models*. Springer International Publishing, first ed., 2018.

[20] A. Ahmed, *Software Engineering in the Agile World*. USA: CreateSpace Independent Publishing Platform, 2018.

[21] P. Bourque, R. E. Fairley, *et al.*, *Guide to the software engineering body of knowledge (SWEBOK (R)): Version 3.0*. IEEE Computer Society Press, 2014.

[22] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*. Addison-Wesley Professional, 3rd ed., 2012.

[23] P. A. Laplante, *Requirements Engineering for Software and Systems*. Applied Software Engineering Series, Auerbach Publications, third ed., 2017.

[24] A. A. Project Management Institute, *Guía Práctica de Ágil*. Project Management Institute, Inc, 2017.

[25] D. McKenna, *The Art of Scrum: How Scrum Masters Bind Dev Teams and Unleash Agility*. Apress, 2016.

[26] I. Bibik, *How to Kill the Scrum Monster*. Apress, first ed., 2018.

[27] H. Cervantes and R. Kazman, *Designing Software Architectures: A Practical Approach*. Addison-Wesley Professional, 1st ed., 2016.

[28] W. Hasselbring, *Software Architecture: Past, Present, Future*, pp. 169–184. Springer International Publishing, 2018.

[29] R. C. Martin, *Clean architecture: a craftsman's guide to software structure and design*. Prentice Hall Press, 2017.

[30] M. A. Babar, A. W. Brown, and I. Mistrík, *Agile Software Architecture: Aligning Agile Processes and Software Architectures*. Newnes, 2013.

[31] J. Ingeno, *Software Architect's Handbook*. Packt Publishing, first ed., 2018.

[32] R. S. Sangwan, *Software and Systems Architecture in Action*. Applied Software Engineering Series, Auerbach Publications, 2018.

[33] D. C. Ashmore, *Microservices for Java Architects: Addendum for the Java EE Architect's Handbook*. DVT Press, 2016.

[34] P. S. Kasun Indrasiri, *Microservices for the Enterprise*. Apress, 2018.

[35] J. P. Alex Soto Bueno, Andy Gumbrecht, *Testing Java Microservice*. Manning Publications, 2018.

[36] E. Wolff, *Microservices: flexible software architecture*. Addison-Wesley Professional, 2016.

[37] T. Erl, *Service-Oriented Architecture: Analysis and Design for Services and Microservices*. Prentice Hall, second ed., 2016.

[38] R. Rodger, *The Tao of Microservices*. Manning Publications, 2017.

[39] B. Familiar, *Microservices, IoT and Azure*. Apress, first ed., 2015.

[40] M. H. Katherine Stanley, Erin Schnabel, *Microservices Best Practices for Java*. IBM Redbooks, first ed., 2016.

[41] D. Hanes, G. Salgueiro, P. Grossetete, R. Barton, and J. Henry, *IoT Fundamentals: Networking Technologies, Protocols, and Use Cases for the Internet of Things*. Cisco Press, 1st ed., 2017.

[42] J. Jamali, B. Bahrami, A. Heidari, P. Allahverdizadeh, and F. Norouzi, *Towards the Internet of Things*. Springer International Publishing, first ed., 2020.

[43] S. Cirani, G. Ferrari, M. Picone, and L. Veltri, *Internet of Things: Architectures, Protocols and Standards*. John Wiley & Sons, 2018.

[44] P. Raj and A. C. Raman, *The Internet of things: Enabling technologies, platforms, and use cases*. Auerbach Publications, 2017.

[45] L. Khalid, *Software Architecture for Business*. Springer International Publishing, first ed., 2020.

[46] P. Xiao, *Practical Java Programming for IoT, AI, and Blockchain*. John Wiley & Sons Inc, first ed., 2019.

[47] A. Chaudhuri, *Internet of Things, for Things, and by Things*. Internal Audit and IT Audit, Auerbach Publications, first ed., 2018.

[48] A. Zimmermann, R. Schmidt, K. Sandkuhl, D. Jugel, J. Bogner, and M. Möhring, "Decision-controlled digitization architecture for internet of things and microservices," in *International Conference on Intelligent Decision Technologies*, pp. 82–92, Springer, 2017.

Anexos

Anexo A

Dispositivos IoT

A.1. Etiquetas NFC

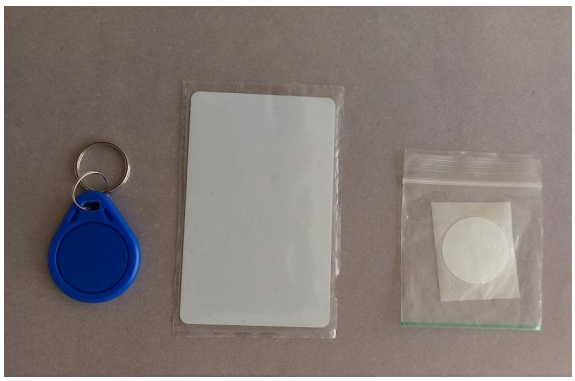
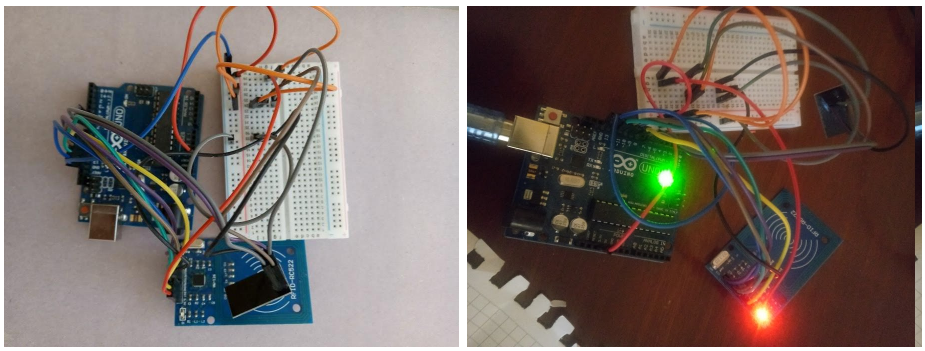


Figura A.1. Etiquetas NFC/RFID

Fuente: Elaboración propia.

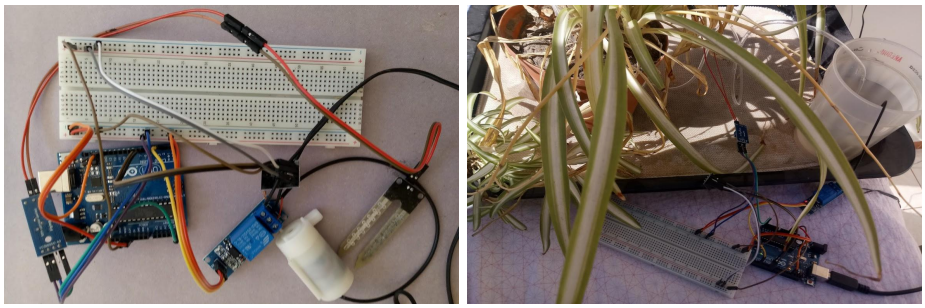
A.2. Dispositivos IoT



(a) Dispositivo Ensamblado (b) Dispositivo Operativo

Figura A.2. Dispositivo IoT lector de tarjetas NFC/RIFD.

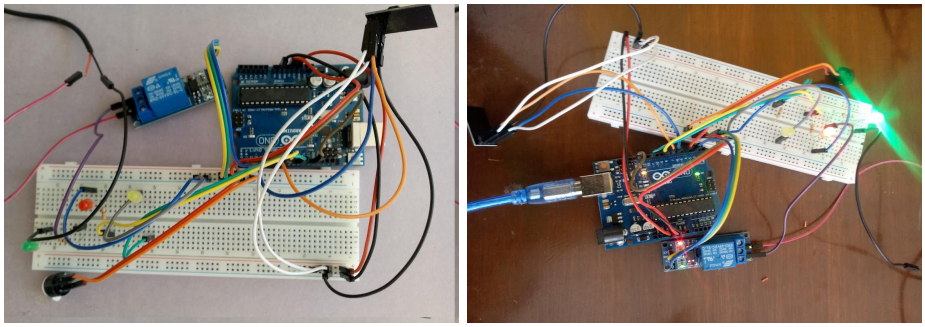
Fuente: Elaboración propia.



(a) Dispositivo Ensamblado (b) Dispositivo Operativo

Figura A.3. Dispositivo IoT de irrigación.

Fuente: Elaboración propia.



(a) Dispositivo Ensamblado (b) Dispositivo Operativo

Figura A.4. Dispositivo prototipo IoT de apertura y cierre de puerta.

Fuente: Elaboración propia.

Anexo B

Software

B.1. Integración y entrega continua

Todo +

S	W	Nombre ↓	Último Éxito	Último Fallo	Última Duración
		ms-api-assitance	9 Hor 2 Min - #1	N/D	2 Min 13 Seg
		ms-door-prototype	41 Min - #3	N/D	3 Min 12 Seg
		ms-gen-uuid	15 Hor - #1	N/D	4 Min 25 Seg
		ms-irrigation	15 Hor - #1	N/D	3 Min 45 Seg
		nfc-assistance-microservices	41 Min - #1	N/D	6 Min 51 Seg

Figura B.1. Microservicios en el servidor Jenkins

Fuente: Elaboración propia.

Figura B.2. Proceso de integración y entrega continua

Fuente: Elaboración propia.

	Prepare environment	Code analysis	Unit Test	Build	Build docker image	Publish image
Average stage times: (Average full run time: ~6min 51s)	6s	1min 26s	1min 57s	29s	25s	2min 22s
#1 Sep 05 11:38 No Changes	6s	1min 26s	1min 57s	29s	25s	2min 22s

Figura B.3. Tiempo de procesos de Integración y entrega continua

Fuente: Elaboración propia.

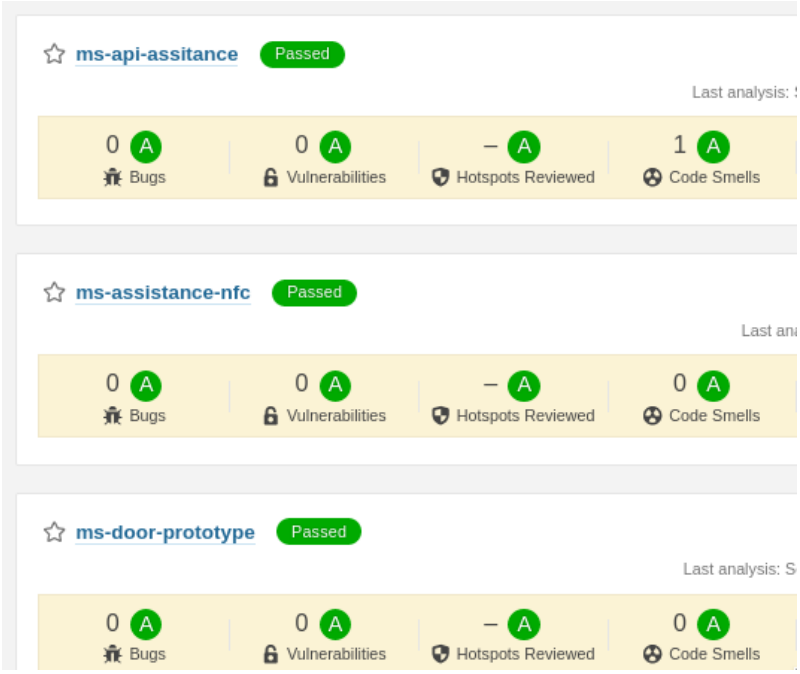


Figura B.4. Servidor de análisis de calidad

Fuente: Elaboración propia.

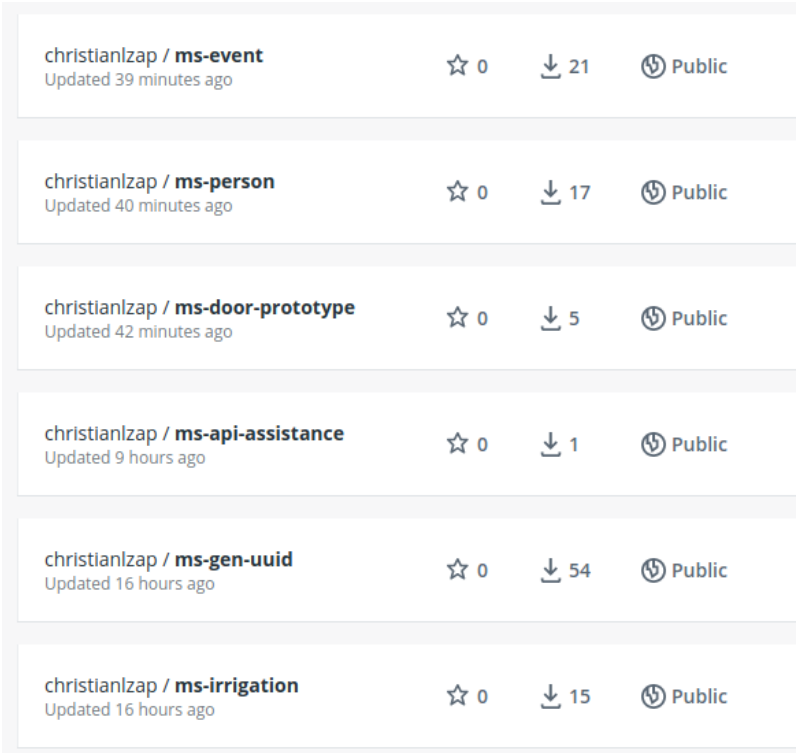


Figura B.5. Publicación de imágenes de los microservicios

Fuente: Elaboración propia.

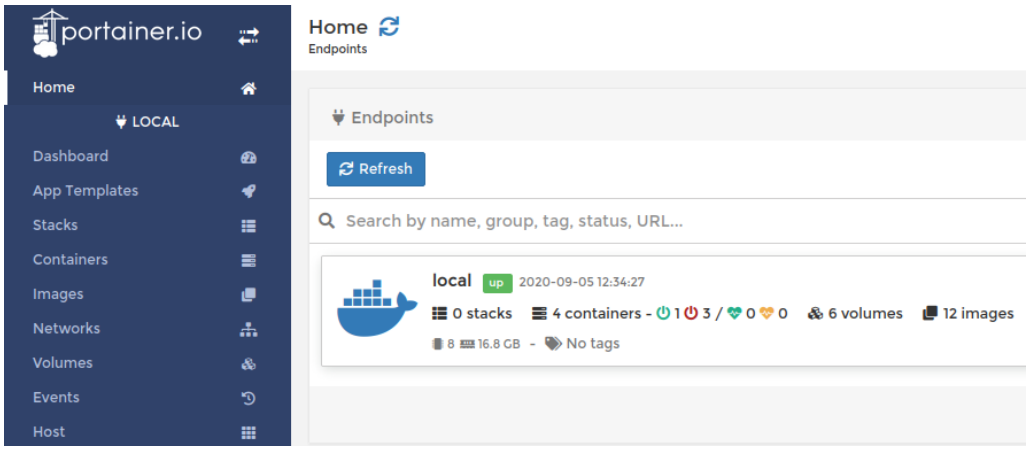


Figura B.6. Herramienta de gestión de contenedores

Fuente: Elaboración propia.

```
1  FROM openjdk:8-jre-alpine
2  RUN mkdir /app
3  WORKDIR /app
4  ADD ./target/ms-irrigation-1.0.jar /app
5  EXPOSE 8085/TCP
6  CMD ["java", "-jar", "ms-irrigation-1.0.jar"]
```

Figura B.7. Archivo de configuración Dockerfile

Fuente: Elaboración propia.

Anexo C

Casos de prueba y backlog de proyecto

C.1. Casos de prueba

Los casos de prueba realizados son los siguientes:

- Funcionamiento del escenario del prototipo de riego
- Funcionamiento del escenario de prototipo de puerta
- Funcionamiento del escenario de registro de asistencia a alumnos

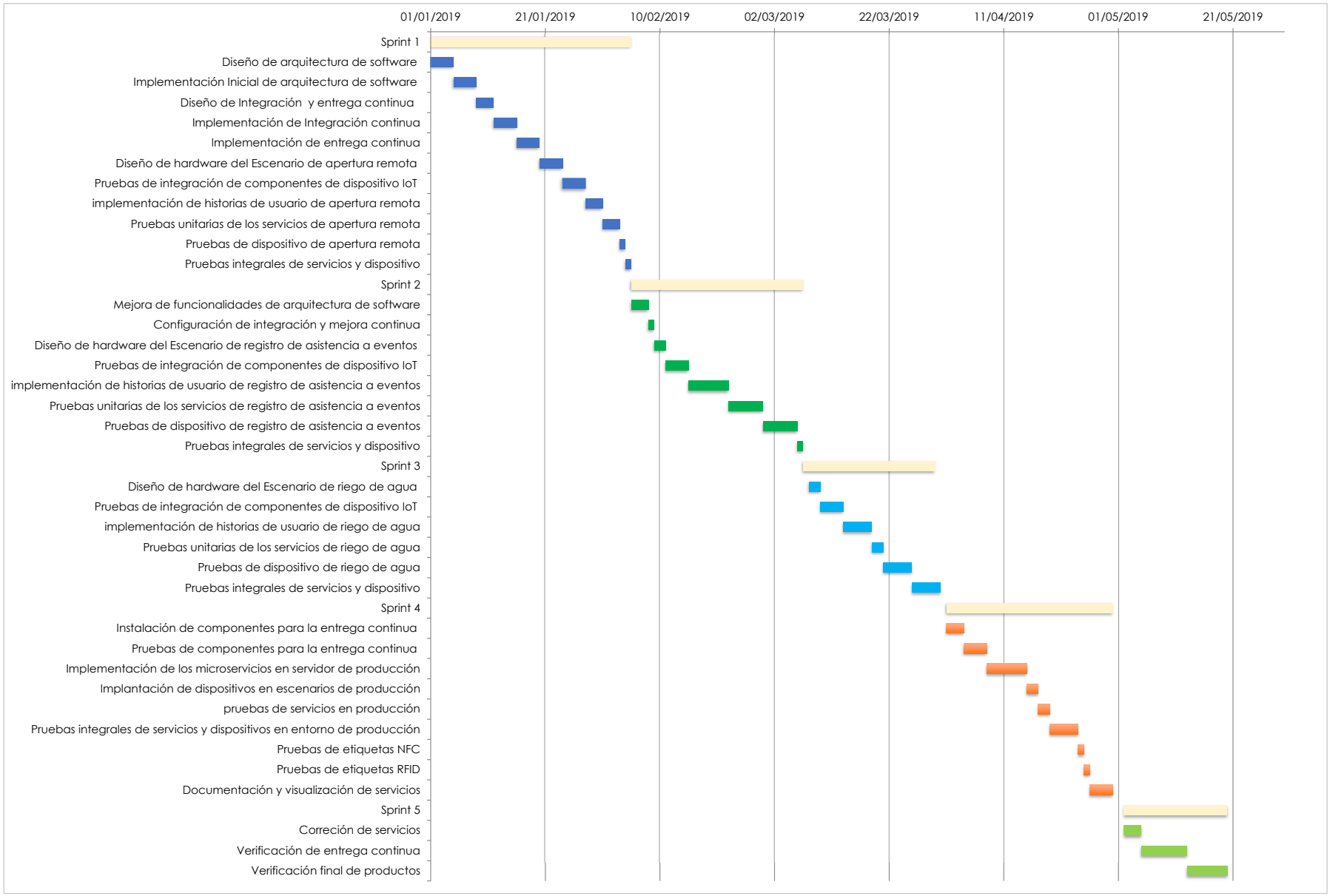
Estos casos de prueba pueden ser visualizados mediante el siguiente:
`https://drive.google.com/drive/folders/11RVFHtXekeqG_Vs42M9n5acjzaL-_YXc`.

C.2. Backlog de proyecto

Se demuestra el Backlog de implementación de proyecto, primero se muestra una tabla donde especifica todas las tareas realizadas y el tiempo que determina para cada actividad, también se demuestra el desarrollo de historias de usuario con un diagrama que demuestra el número actividades realizadas en base a los días, de esta forma se determina el esfuerzo correspondiente por cada historia de usuario.

NOMBRE DEL PROYECTO	ENCARGADO DEL PROYECTO	FECHA DE INICIO	FECHA FIN	TOTAL DIAS
AMS-IOT	Christian loza peralta	1/01/2019	7-Ene	135

EN RIESGO	NOMBRE DE LA TAREA	TIPO DE CARACTERISTICA	RESPONSABILIDAD	INICIO	FIN	DURACION (DIAS)	ESTADO	COMENTARIO
☐	**Sprint 1**			**01/01/2019**	**05/02/2019**	**35**	**Completo**	
☐	Diseño de arquitectura de software	Arquitectura	Christian Loza P.	01/01/2019	05/01/2019	4	Completo	
☐	Implementación Inicial de arquitectura de software	Aquitectura	Christian Loza P.	05/01/2019	09/01/2019	4	Completo	
☐	Diseño de Integración y entrega continua	DevOps	Christian Loza P.	09/01/2019	12/01/2019	3	Completo	
☐	Implementación de Integración continua	DevOps	Christian Loza P.	12/01/2019	16/01/2019	4	Completo	
☐	Implementación de entrega continua	DevOps	Christian Loza P.	16/01/2019	20/01/2019	4	Completo	
☐	Diseño de hardware del Escenario de apertura remota	Hardware	Christian Loza P.	20/01/2019	24/01/2019	4	Completo	
☐	Pruebas de integración de componentes de dispositivo IoT	Pruebas	Christian Loza P.	24/01/2019	28/01/2019	4	Completo	
☐	implementación de historias de usuario de apertura remota	Desarrollo	Christian Loza P.	28/01/2019	31/01/2019	3	Completo	
☐	Pruebas unitarias de los servicios de apertura remota	Pruebas	Christian Loza P.	31/01/2019	03/02/2019	3	Completo	
☐	Pruebas de dispositivo de apertura remota	Pruebas	Christian Loza P.	03/02/2019	04/02/2019	1	Completo	
☐	Pruebas integrales de servicios y dispositivo	Pruebas	Christian Loza P.	04/02/2019	05/02/2019	1	Completo	
☐	**Sprint 2**			**05/02/2019**	**07/03/2019**	**30**	**Completo**	
☐	Mejora de funcionalidades de arquitectura de software	Arquitectura	Christian Loza P.	05/02/2019	08/02/2019	3	Completo	
☐	Configuración de integración y mejora continua	DevOps	Christian Loza P.	08/02/2019	09/02/2019	1	Completo	
☐	Diseño de hardware del Escenario de registro de asistencia a eventos	Hardware	Christian Loza P.	09/02/2019	11/02/2019	2	Completo	
☐	Pruebas de integración de componentes de dispositivo IoT	Pruebas	Christian Loza P.	11/02/2019	15/02/2019	4	Completo	
☐	implementación de historias de usuario de registro de asistencia a eventos	Desarrollo	Christian Loza P.	15/02/2019	22/02/2019	7	Completo	
☐	Pruebas unitarias de los servicios de registro de asistencia a eventos	Pruebas	Christian Loza P.	22/02/2019	28/02/2019	6	Completo	
☐	Pruebas de dispositivo de registro de asistencia a eventos	Pruebas	Christian Loza P.	28/02/2019	06/03/2019	6	Completo	
☐	Pruebas integrales de servicios y dispositivo	Pruebas	Christian Loza P.	06/03/2019	07/03/2019	1	Completo	
☐	**Sprint 3**			**07/03/2019**	**31/03/2019**	**23**	**Completo**	
☐	Diseño de hardware del Escenario de riego de agua	Hardware	Christian Loza P.	08/03/2019	10/03/2019	2	Completo	
☐	Pruebas de integración de componentes de dispositivo IoT	Pruebas	Christian Loza P.	10/03/2019	14/03/2019	4	Completo	
☐	implementación de historias de usuario de riego de agua	Desarrollo	Christian Loza P.	14/03/2019	19/03/2019	5	Completo	
☐	Pruebas unitarias de los servicios de riego de agua	Pruebas	Christian Loza P.	19/03/2019	21/03/2019	2	Completo	
☐	Pruebas de dispositivo de riego de agua	Pruebas	Christian Loza P.	21/03/2019	26/03/2019	5	Completo	
☐	Pruebas integrales de servicios y dispositivo	Pruebas	Christian Loza P.	26/03/2019	31/03/2019	5	Completo	
☐	**Sprint 4**			**01/04/2019**	**30/04/2019**	**29**		
☐	Instalación de componentes para la entrega continua	Software	Christian Loza P.	01/04/2019	04/04/2019	3	Completo	
☐	Pruebas de componentes para la entrega continua	Software	Christian Loza P.	04/04/2019	08/04/2019	4	Completo	
☐	Implementación de los microservicios en servidor de producción	Software	Christian Loza P.	08/04/2019	15/04/2019	7	Completo	
☐	Implantación de dispositivos en escenarios de producción	Software	Christian Loza P.	15/04/2019	17/04/2019	2	Completo	
☐	pruebas de servicios en producción	Software	Christian Loza P.	17/04/2019	19/04/2019	2	Completo	
☐	Pruebas integrales de servicios y dispositivos e	Hardware	Christian Loza P.	19/04/2019	24/04/2019	5	Completo	
☐	Pruebas de etiquetas NFC	Hardware	Christian Loza P.	24/04/2019	25/04/2019	1	Completo	
☐	Pruebas de etiquetas RFID	Hardware	Christian Loza P.	25/04/2019	26/04/2019	1	Completo	
☐	Documentación y visualización de servicios	Documentación	Christian Loza P.	26/04/2019	30/04/2019	4	Completo	
☐	**Sprint 5**			**02/05/2019**	**20/05/2019**	**18**		
☐	Correción de servicios	Software	Christian Loza P.	02/05/2019	05/05/2019	3	Completo	
☐	Verificación de entrega continua	DevOps	Christian Loza P.	05/05/2019	13/05/2019	8	Completo	
☐	Verificación final de productos	Software	Christian Loza P.	13/05/2019	20/05/2019	7	Completo	


01/01/2019
21/01/2019
10/02/2019
02/03/2019
22/03/2019
11/04/2019
01/05/2019
21/05/2019
Sprint 1
Diseño de arquitectura de software
Implementación Inicial de arquitectura de software
Diseño de Integración y entrega continua
Implementación de Integración continua
Implementación de entrega continua
Diseño de hardware del Escenario de apertura remota
Pruebas de integración de componentes de dispositivo IoT
implementación de historias de usuario de apertura remota
Pruebas unitarias de los servicios de apertura remota
Pruebas de dispositivo de apertura remota
Pruebas integrales de servicios y dispositivo
Sprint 2
Mejora de funcionalidades de arquitectura de software
Configuración de integración y mejora continua
Diseño de hardware del Escenario de registro de asistencia a eventos
Pruebas de integración de componentes de dispositivo IoT
implementación de historias de usuario de registro de asistencia a eventos
Pruebas unitarias de los servicios de registro de asistencia a eventos
Pruebas de dispositivo de registro de asistencia a eventos
Pruebas integrales de servicios y dispositivo
Sprint 3
Diseño de hardware del Escenario de riego de agua
Pruebas de integración de componentes de dispositivo IoT
implementación de historias de usuario de riego de agua
Pruebas unitarias de los servicios de riego de agua
Pruebas de dispositivo de riego de agua
Pruebas integrales de servicios y dispositivo
Sprint 4
Instalación de componentes para la entrega continua
Pruebas de componentes para la entrega continua
Implementación de los microservicios en servidor de producción
Implantación de dispositivos en escenarios de producción
pruebas de servicios en producción
Pruebas integrales de servicios y dispositivos en entorno de producción
Pruebas de etiquetas NFC
Pruebas de etiquetas RFID
Documentación y visualización de servicios
Sprint 5
Correción de servicios
Verificación de entrega continua
Verificación final de productos

TAREA BACKLOG	Responsable	Estado	TIEMPO ESTIMADO	DIA 1	DIA 2	DIA 3	DIA 4	DIA 5	SPRINT REVIEW
HU - Prototipo Puerta remota									
Gestionar prototipos puerta	Christian Loza Peralta	Completo	1	1	0	0	0	0	0
Verificación de estado de puerta	Christian Loza Peralta	Completo	1	0	1	0	0	0	0
Lista de registro de la estados de puerta	Christian Loza Peralta	Completo	1	0	0	1	0	0	0
Lista de las últimas 15 estados de la puerta	Christian Loza Peralta	Completo	0	0	0	0	0	0	0
HU - REGISTRO EVENTOS - COORDINADOR									
Gestionar eventos academicos	Christian Loza Peralta	Completo	2	1	1	0	0	0	0
Ver los asistentes que se registraron en el evento	Christian Loza Peralta	Completo	1	1	0	0	0	0	0
Ver el total de asistentes que asistieron a un evento	Christian Loza Peralta	Completo	1	0	1	0	0	0	0
Obtener reporte de los asistentes a un evento	Christian Loza Peralta	Completo	0	0	0	0	0	0	0
Obtener los datos de un asistente que participo a un evento	Christian Loza Peralta	Completo	0	0	0	0	0	0	0
Buscar asistentes por DNI	Christian Loza Peralta	Completo	0	0	0	0	0	0	0
Buscar personas por DNI	Christian Loza Peralta	Completo	0	0	0	0	0	0	0
HU - REGISTRO EVENTOS - ESTUDIANTES									
Listar eventos academicos disponibles	Christian Loza Peralta	Completo	1	0	0	0	0	0	0
Listar eventos asistidos como asistente	Christian Loza Peralta	Completo	0	0	0	0	0	0	0
Ver el total de creditos extracurriculares generados por los eventos asistidos	Christian Loza Peralta	Completo	1	0	0	1	0	0	0
Obtener una notificación de haber alcanzado el limite de los creditos	Christian Loza Peralta	Completo	0	0	0	0	0	0	0
Inscribirse a un evento academico	Christian Loza Peralta	Completo	1	0	0	0	1	0	0
Listar eventos academicos finalizados	Christian Loza Peralta	Completo	0	0	0	0	0	0	0
HU - RIEGO DE PLANTA									
Gestonar plantas	Christian Loza Peralta	Completo	2	1	0	0	0	0	0
Listar plantas registradas	Christian Loza Peralta	Completo	1	0	0	1	0	0	0
Listar información de una planta según identificación	Christian Loza Peralta	Completo	1	0	0	0	1	0	0
Listar las útlimos 15 estados de la puerta	Christian Loza Peralta	Completo	1	0	0	0	0	1	0
HU - INTERESADO									
Visualizar servicios web	Christian Loza Peralta	Completo	3	1	1	0	1	0	0
Documentar servicios web	Christian Loza Peralta	Completo	2	1	0	1	0	0	0
Verificar entrega continua	Christian Loza Peralta	Completo	8	2	2	2	1	1	0
Verificación de productos	Christian Loza Peralta	Completo	7	1	1	1	2	3	2
TOTAL			**35**	**9**	**7**	**7**	**6**	**5**	**2**