Diseño de un framework software extensible para dispositivos tipo gateway integrados en plataformas IoT para Smart Campus.

Camilo Andrés Gutiérrez Velásquez

Trabajo de Grado para optar por el título de Ingeniero de Sistemas

Director
Gabriel Rodrigo Pedraza Ferreira
PhD. Ciencias de la Computación

Universidad Industrial de Santander
Facultad de Ingenierías Físico Mecánicas
Escuela de Ingeniería de Sistemas e Informática
Bucaramanga
2019

Agradecimientos

“En primera instancia quiero agradecer a Dios por acompañarme en cada paso de mi vida dándome sabiduría y calma.

A mis padres por hacer su mayor esfuerzo por ayudarme a salir adelante y ser una buena persona.

A mi hermana por ser un ejemplo a seguir demostrándome que las metas y los sueños se pueden hacer realidad.

A mis tíos y abuelos por siempre inculcarme buenos valores y estar presentes cuando más los he necesitado.

A mi pareja por acompañarme durante todo este tiempo brindándome su apoyo incondicional.

A mi director de proyecto Gabriel por su guía y acompañamiento durante el desarrollo de este proyecto.

Por último, a todos mis amigos con los que compartí gratos momentos durante toda mi estancia en la universidad.”

Infinitas Gracias

Tabla de Contenido

Pág.

Lista de Tablas

Lista de Figuras

Lista de Apéndices

(Ver apéndices adjuntos en el CD y pueden visualizarlos en la Base de Datos de la Biblioteca UIS)

Apéndice A. API REST del framework.

Apéndice B. Proyecto esqueleto para procesos.

Apéndice C. Configuración del framework.

Apéndice D. Vídeo explicativo funcionamiento de la plataforma IoT.

Lista de Siglas

AMQP:	Advanced Message Queuing Protocol
API:	Application Programming Interface
AWS:	Amazon Web Services
GPIO:	General Purpose Input/Output
GPS:	Global Positioning System
I2C:	Inter-Integrated Circuit
IoT:	Internet of Things
JSON:	JavaScript Object Notation
LED:	Light-emitting diode
MQTT:	Message Queue Telemetry Transport
NFC:	Near Field Communication
OSGi:	Open Services Gateway initiative
REST:	Representational State Transfer
TCP:	Transmission Control Protocol
Wi-Fi:	Wireless Fidelity

Resumen

TITULO: DISEÑO DE UN FRAMEWORK SOFTWARE EXTENSIBLE PARA DISPOSITIVOS TIPO GATEWAY INTEGRADOS EN PLATAFORMAS IOT PARA SMART CAMPUS*

AUTOR: CAMILO ANDRES GUTIERREZ VELASQUEZ**

PALABRAS CLAVE: IOT, FRAMEWORK, SOFTWARE, GATEWAY, SMART CAMPUS

DESCRIPCIÓN:

La gran afluencia de personas en las universidades ha creado un reto en cuanto a optimización de recursos físicos y control de dispositivos, por ejemplo, para controlar la temperatura, la luminosidad y la humedad de manera remota y/o automática, ya que con el pasar del tiempo el número de estos dispositivos incrementa con gran velocidad, complicando así cada vez más su gestión. Por ende, una posible solución en este contexto es la puesta en práctica de IoT con el objetivo de mejorar la calidad de vida de los individuos que hacen uso de las instalaciones universitarias, a través de la transformación digital de estas instituciones en universidades inteligentes (Smart Campus).

En este proyecto de grado se presenta el diseño de un framework de software extensible que permite a dispositivos tipo gateway la comunicación y el almacenamiento de datos producidos y recibidos por sensores y/o actuadores, al mismo tiempo que provee la capacidad de conectarse con plataformas IoT enfocadas en Smart Campus. Esto permite que los diferentes casos de uso de IoT sean implementados con menor dificultad, enfocándose en el manejo de los dispositivos finales (sensor y actuador) y delegando al framework lo referente al exterior y funciones generales.

Al final, se desarrolló un caso de uso para probar las funciones que cumple el framework, su extensibilidad, su portabilidad y su capacidad de conectarse a una arquitectura de IoT para Smart Campus.

---

*Trabajo de grado

**Facultad de Ingenierías Físicomecánicas. Escuela de Ingeniería de Sistemas e Informática

Director: Gabriel Rodrigo Pedraza Ferreira, PhD. Ciencias de la Computación

Abstract

TITULO: DESIGN OF AN EXTENSIBLE FRAMEWORK FOR GATEWAY DEVICES INTEGRATED ON IOT PLATFORMS FOR SMART CAMPUS.

AUTOR: CAMILO ANDRES GUTIERREZ VELASQUEZ**

PALABRAS CLAVE: IOT, FRAMEWORK, SOFTWARE, GATEWAY, SMART CAMPUS

DESCRIPCIÓN:

The huge influx of people at universities has created a challenge in terms of physical resources' optimization and control of devices, for instance, temperature, luminosity and humidity control either remotely or automatically, since the number of these devices increases quickly over time making more complex their management. Therefore, a possible solution to this problem is the implementation of IoT with the aim of improving the quality of life of the individuals who make use of university's facilities through the digital transformation of these institutions into smart universities (Smart Campus).

In this thesis, the design of an extensible software framework is presented, which allows gateway devices to communicate and store data produced by sensors and received by actuators, while providing the ability to connect with IoT platforms focused on Smart Campus. This allows the different IoT's use cases to be implemented easier focusing on the management of final devices (sensors and actuators) while delegating the external and general functionalities to the framework.

At the end of the thesis, a use case was developed in order to test the provided framework's functions, its extensibility, portability and ability to be connected to an IoT platform for Smart Campus.

---

*Bachelor Thesis

**Facultad de Ingenierías Físicomecánicas. Escuela de Ingeniería de Sistemas e Informática

Director: Gabriel Rodrigo Pedraza Ferreira, PhD. Ciencias de la Computación

Introducción

En los últimos años se ha visto un deseo latente en los humanos por sistematizar tantos objetos como sea posible para usar las ventajas que ofrece la tecnología, especialmente con la llegada del Internet. Por este motivo, nació un nuevo concepto llamado IoT (Internet of Things), que busca convertir los objetos o aparatos que se usan en la vida cotidiana en dispositivos inteligentes capaces de conectarse a Internet para que puedan ser manejados a través de esta red.

Algunos ejemplos de IoT son, la telemedicina, el control de pacientes en tiempo real y diagnósticos oportunos especialmente útil en zonas remotas donde hay escasez de médicos, la gestión del tráfico para mejorar la movilidad en las ciudades, la asistencia en la agricultura para monitorizar el estado de los cultivos permitiendo a los agricultores tomar mejores decisiones y reduciendo la pérdida de los productos, el control de inventarios por medio de dispositivos que rastreen su ubicación en tiempo real, el riego de zonas verdes de manera automatizada o controlada remotamente, el control de la toxicidad y de niveles de contaminación del ambiente para evitar efectos negativos en la vida de las personas y animales, entre otros (del Valle, s.f).

La gran afluencia de personas en las universidades ha creado un reto en cuanto a optimización de recursos físicos y control de dispositivos de manera remota y/o automática, ya que con el pasar del tiempo el número de estos dispositivos incrementa con gran velocidad, complicando así cada vez más su gestión. Por ende, una posible solución en este contexto es la’ puesta en práctica de IoT con el objetivo de mejorar la calidad de vida de los individuos que hacen uso de las instalaciones universitarias, a través de la transformación digital de estas instituciones en universidades inteligentes (Smart Campus).

No obstante, el desarrollo de aplicaciones IoT no es una tarea trivial, pues estas requieren la unión de una arquitectura hardware y software adecuada para soportar un entorno de ejecución óptimo. Uno de los componentes claves en el desarrollo de una aplicación IoT son los elementos conocidos como gateways, encargados de agrupar dispositivos con restricciones de recursos para permitirles interactuar con plataformas IoT entregando información valiosa a los usuarios.

En el siguiente proyecto de grado se presenta el diseño de un framework de software extensible que permite a dispositivos tipo gateway la comunicación y el almacenamiento de datos producidos por sensores y recibidos por actuadores, al mismo tiempo que provee la capacidad de conectarse con plataformas IoT enfocadas en Smart Campus. Esto permite que los diferentes casos de uso de IoT sean implementados con menor complejidad, enfocándose en el manejo de los dispositivos finales (sensor y actuador) y delegando al framework lo referente al exterior y funciones generales.

Posteriormente se desarrolló un caso de uso para probar las funciones que provee el framework, su extensibilidad, su portabilidad y su capacidad de conectarse a una plataforma IoT para Smart Campus.

1. Objetivos

1.1 Objetivo general

Diseñar un framework software extensible que permita a dispositivos tipo gateway que contengan sensores y actuadores ser integrados en plataformas IoT para Smart Campus.

1.2 Objetivos específicos

- Ofrecer mecanismos de extensión del framework para soportar la adición de nuevos sensores y actuadores en el dispositivo Gateway.
- Ofrecer un conjunto de servicios base para ser utilizados por las aplicaciones desplegadas en los dispositivos tipo Gateway como: comunicación, procesamiento y persistencia local.
- Ofrecer una interfaz programática (API) que permita registrar el Gateway y sus elementos asociados en una plataforma de Internet de las cosas.

2. Estado del arte

Es importante resaltar la labor que se ha desempeñado en cuanto al Internet de las Cosas junto con Edge Computing, los cuales son campos que están directamente relacionados con el presente trabajo de grado, por lo tanto, a continuación, se muestran algunos ejemplos de esto:

Eclipse Kura es un framework IoT Edge basado en Java/OSGi. Kura ofrece un API de acceso a las interfaces hardware de los Gateways IoT (puertos seriales, GPS, watchdog, GPIOs, I2C, etc.). Cuenta con protocolos de campo listos para usar (incluyendo ModBus, OPC-UA, S7), un contenedor de aplicación y un flujo de programación visual de datos basado en la web para adquirir datos desde el campo, procesarlos en el Edge y publicarlo en las principales plataformas IoT en la nube a través de conectividad MQTT (Eclipse Foundation, Inc. [Eclipse], s.f).

Node-RED es una herramienta de programación para conectar dispositivos hardware, APIs y servicios online de maneras nuevas e interesantes. Esta herramienta provee un editor basado en el navegador que hace fácil conectar flujos usando una amplia gama de nodos en la paleta y que pueden ser desplegados en tiempo de ejecución con un solo clic (Node-RED, s.f).

Como caso de uso, se menciona el trabajo de grado relacionado con el Internet de las Cosas llamado Diseño de una plataforma software que apoye los procesos de crianza en el área de la piscicultura (Montañez, 2017).

3. Marco de referencia

3.1 Internet de las cosas

El Internet de las cosas (IoT) es un concepto que puede parecer nuevo, pero que en realidad se ha venido mencionando desde hace algunos años; IoT trata acerca de convertir objetos de la vida cotidiana en objetos interconectados a través de Internet. Algunos ejemplos de IoT serían:

- Redes de sensores que miden el clima, la temperatura, la cantidad de personas, etc.
- Dispositivos implantados en humanos para tomar signos vitales en tiempo real.
- Vehículos conectados a Internet.
- Sistemas automatizados en el hogar y control de luminosidad.

Por consiguiente, se podría entender dicho concepto como un conjunto de dispositivos y sistemas que interconectan sensores y actuadores del mundo real a Internet. El crecimiento en el número y la variedad de dispositivos capaces de recoger datos valiosos para su posterior procesamiento es bastante acelerado. Un estudio realizado por Cisco estima que el número de dispositivos conectados en el 2020 será de 50 billones (Pastor, 2018).

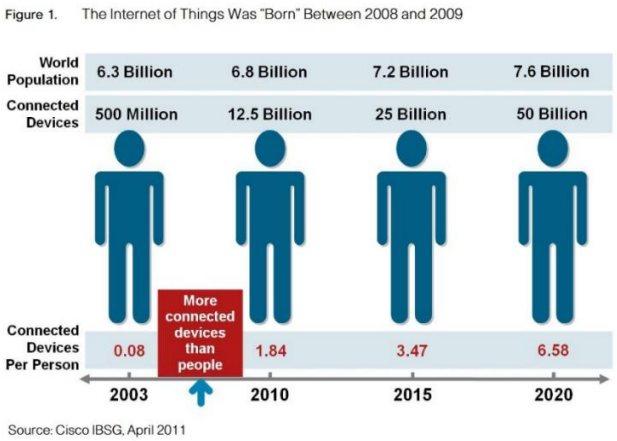

Figura 1. Proyección de Internet de las Cosas. Tomada de Pastor. (2018).

La conexión de estos dispositivos a Internet o a una pasarela (Gateway) puede realizarse, entre otros mediante:

- **Ethernet directo o Wi-Fi**
- **Bluetooth de baja energía:** Es una nueva tecnología digital de radio (inalámbrica) interoperable para pequeños dispositivos desarrollada por Bluetooth.
- **Comunicación de campo cercano (NFC):** Es una tecnología de comunicación inalámbrica, de corto alcance y alta frecuencia que permite el intercambio de datos entre dispositivos.
- **Zigbee:** Especificación de un conjunto de protocolos de alto nivel de comunicación inalámbrica para su utilización con radiodifusión digital de bajo consumo.

La siguiente imagen representa los dos modos de conexión más utilizados:

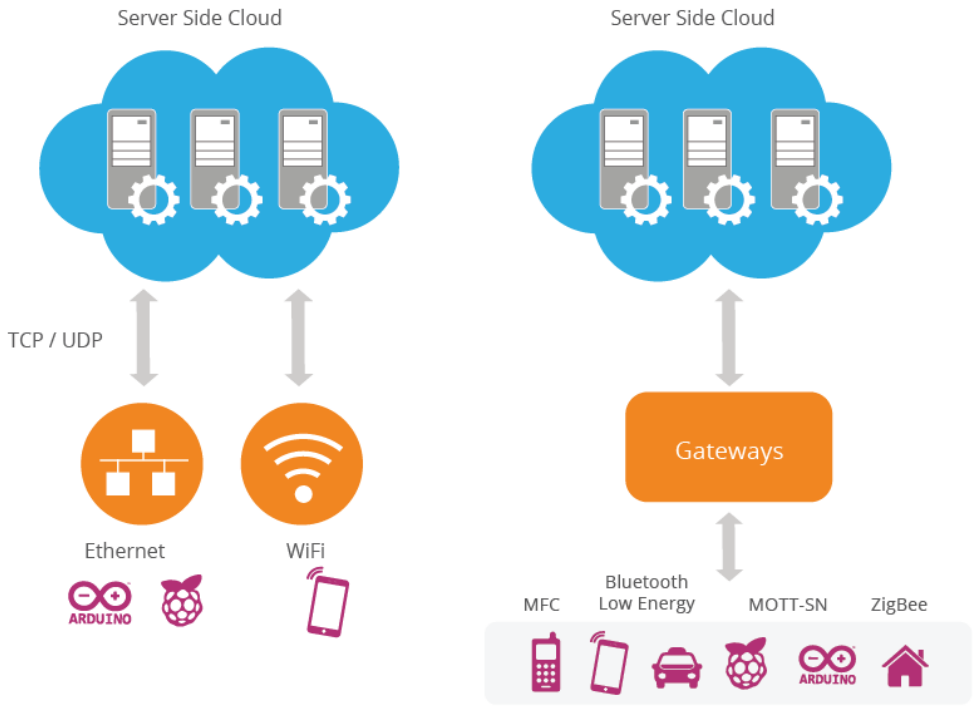

Figura 2. Modos de conexión más comunes. Tomada de Fremantle. (2015)

3.2 Smart Campus

Es muy común oír sobre ciudades inteligentes, en donde se habla acerca de avanzar consiguiendo que la tecnología y sus innovaciones formen parte del día a día de los ciudadanos, pero que a su vez se enfrenta a los problemas que como sociedad se deben asumir, enfocándose en una mejor gestión de los recursos medioambientales, del consumo energético, de la disminución de los residuos, etc.

Cuando hablamos de Smart Campus (Campus Inteligente) se aplican los mismos conceptos, pero enfocados al ámbito universitario, ya que muchas universidades hoy en día se establecen como una pequeña ciudad, en donde todos los implicados pueden encontrar casi todo lo que necesitan. Una solución que se puede aplicar en un Smart Campus sería, por ejemplo, crear un sistema de información geográfica y espacial del territorio que comprende un campus, que permita gestionar todo tipo de eventos, registrar el número de personas que utilizan determinado recurso y controlar con mayor eficiencia el estado de las instalaciones.

3.3 Edge computing

El modo de conexión mostrado en la Figura 2 a la derecha se denomina Edge computing, cuya funcionalidad es brindar procesamiento y almacenamiento con velocidades mayores a las actuales., tal como indica Butler (2017):

La llamada Edge Computing, permite que los datos producidos por los dispositivos de la internet de las cosas se procesen más cerca de donde se crearon en lugar de enviarlos a través de largos recorridos para que lleguen a centros de datos y nubes de computación.

Dicha velocidad de respuesta es de gran utilidad para diferentes industrias que necesiten un tiempo de respuesta pequeño para evitar que sus procesos se vean afectados.

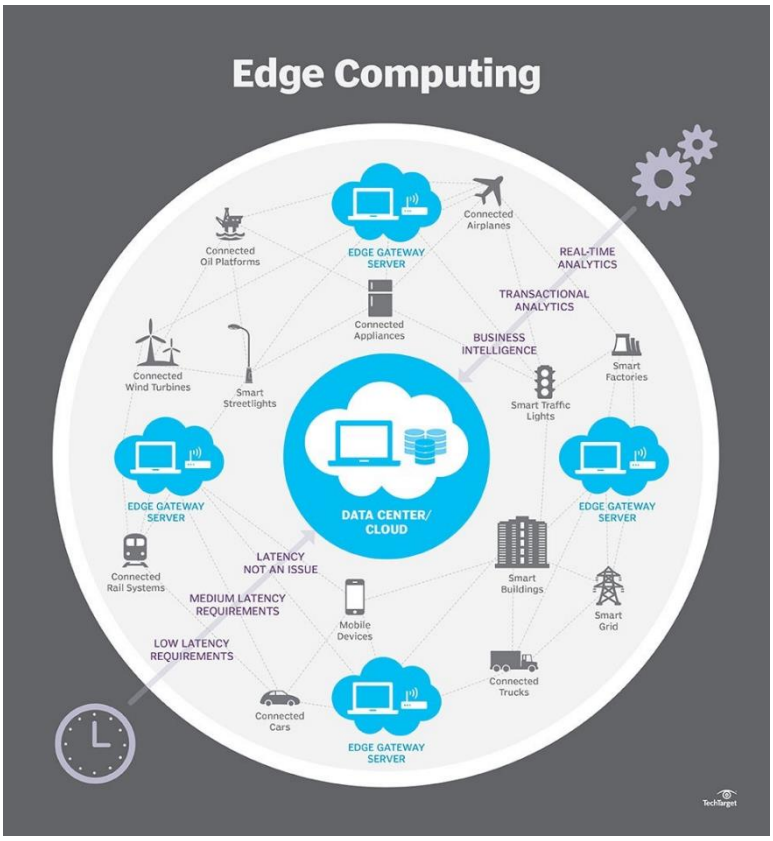

Figura 3. Representación de Edge Computing. Tomada de Pastor. (2018).

3.4 Gateways

Un gateway en una solución de internet de las cosas puede ser un protocolo gateway desplegado en la nube o un gateway de campo desplegado localmente con otros dispositivos como sensores o actuadores.

Dichos gateways tienen como principal característica la traducción de protocolos de comunicación además de proveer almacenamiento y procesamiento rápido, especialmente útiles para aquellos dispositivos restringidos en cuanto a sus recursos y capacidades, ofreciendo la posibilidad de comunicarse con Internet.

Un gateway de campo puede:

- Tomar decisiones rápidas para reducir la latencia.

- Proveer servicios de administración de dispositivos.
- Proveer almacenamiento local.
- Proveer procesamiento en el borde (edge).
- Mejorar la seguridad y privacidad de dispositivos restringidos.
- Desempeñar traducción de protocolos.

3.5 Sensores

Un sensor es un dispositivo capaz de detectar magnitudes físicas o químicas, llamadas variables de instrumentación, y transformarlas en variables eléctricas. Las variables de instrumentación pueden ser, por ejemplo: temperatura, intensidad lumínica, distancia, aceleración, inclinación, desplazamiento, presión, fuerza, torsión, humedad, movimiento, pH, etc. (Aprendiendoarduino, 2016).

Existen diferentes tipos de sensores dependiendo del tipo de variable que deban medir:

- Térmicos
- De humedad
- Magnéticos
- Infrarrojos
- Ópticos
- De contacto

3.6 Actuadores

Un actuador es un dispositivo capaz de transformar energía hidráulica, neumática o eléctrica en la activación de un proceso con la finalidad de generar un efecto sobre elemento externo. Este recibe la orden de un regulador, controlador y en función a ella genera la orden para activar un elemento final de control como, por ejemplo, una válvula (Aprendiendoarduino, 2016).

Existen varios tipos de actuadores utilizados en IoT como son:

- Hidráulicos
- Neumáticos
- Eléctricos

3.7 Interfaz

En el campo de la informática, el término interfaz se emplea para nombrar a la conexión funcional que existe entre dos programas, sistemas o dispositivos y que brinda una comunicación de diversos niveles, haciendo posible un intercambio de información. Existen dos tipos de interfaces: las interfaces de usuario y las interfaces físicas (Venemedia Comunicaciones C.A.).

En el campo del Internet de las Cosas las interfaces son bastante útiles ya que permiten que los datos intercambiados entre sensores, actuadores, gateways y cloud manejen formatos comunes facilitando la comunicación entre los mismos.

3.8 Extensibilidad

La extensibilidad en la ingeniería de software tiene que ver con la capacidad que tiene el diseño de un sistema o plataforma para incorporar nuevas funcionalidades sin afectar las características ofrecidas al inicio. Esto permite que dicho sistema no quede obsoleto con el pasar del tiempo.

3.9 Framework

Un Framework, que se podría traducir aproximadamente como marco de trabajo, es el esquema o estructura que se establece y que se aprovecha para desarrollar y organizar un software determinado. Esta definición, algo compleja, podría resumirse como el entorno pensado para hacer más sencilla la programación de cualquier aplicación o herramienta actual (NeoAttack).

De esta manera un framework está pensado para facilitar la vida de quienes lo utilizan prestando funcionalidades preprogramadas y reduciendo el tiempo necesario para llevar a cabo una solución determinada.

4. Marco metodológico

La metodología usada para el desarrollo de este trabajo de grado consistió en 5 fases representadas en el siguiente diagrama:

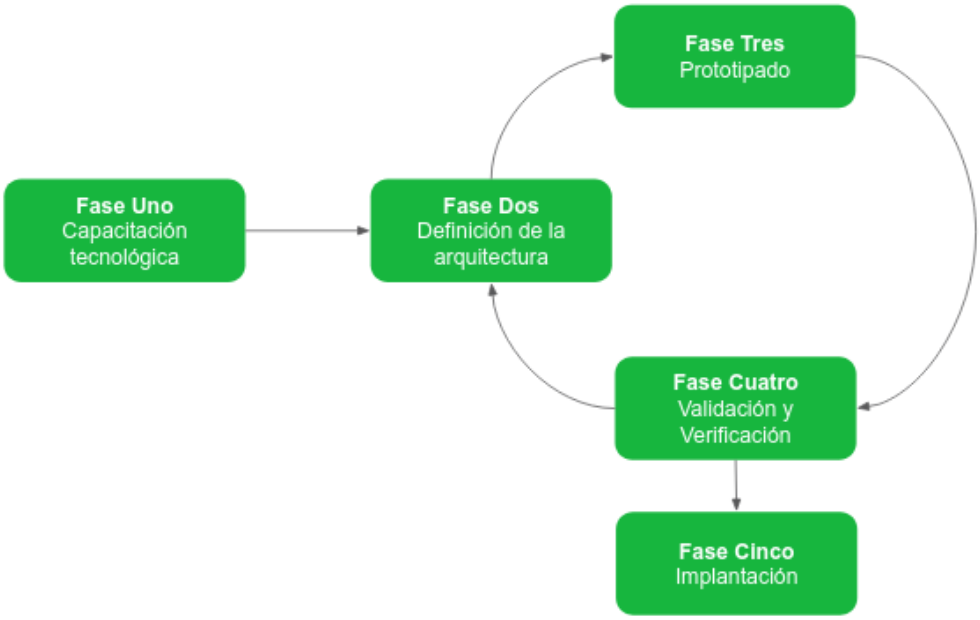

Figura 4. Diagrama metodológico

4.1 Fase uno: Capacitación tecnológica

En esta fase se investigó sobre los conceptos relevantes al Internet de las Cosas y soluciones existentes como Azure IoT y Aws IoT con el objetivo de tener una visión global de las posibles

arquitecturas y soluciones que se pueden brindar en este campo. Además, se llevaron a cabo cursos/tutoriales sobre herramientas, protocolos y estándares orientados a los dispositivos tipo gateway para permitir la correcta integración de los mismos en la plataforma IoT.

Las actividades realizadas en esta fase fueron las siguientes:

- Consulta y entendimiento de conceptos relacionados a IoT.
- Estudio de protocolos de comunicación teniendo en cuenta factores como consumo de recursos, velocidad y confiabilidad.
- Comparación de lenguajes de programación, paradigmas, tecnologías y frameworks orientados a IoT.
- Realización de cursos/tutoriales de herramientas y tecnologías con utilidad en IoT.

4.2 Fase dos: Definición de la arquitectura

En esta fase se definió la arquitectura del framework gateway incluyendo tecnologías, medios o protocolos de comunicación, el flujo de los datos desde su generación hasta su emisión, así como la integración con la plataforma IoT orientada a Smart Campus.

Las actividades realizadas en esta fase fueron las siguientes:

- Definición del alcance del proyecto y las características de la arquitectura.
- Definición de las funcionalidades provistas por el framework Gateway.

4.3 Fase tres: Prototipado

Con base en las decisiones tomadas en la fase dos, en esta fase se prototipó el framework gateway con sus funcionalidades completas y sus características definidas, teniendo en cuenta la capacidad de integración con la plataforma IoT.

Las actividades realizadas en esta fase fueron las siguientes:

- Diseño del prototipo del framework software gateway.

- Integración del framework gateway con la arquitectura IoT.

4.4 Fase cuatro: Verificación y validación

En esta fase se verificó que los métodos y pasos utilizados para llevar a cabo la posible solución fueron correctos, es decir, se verificó que la lógica e implementación cumplieran con los requisitos establecidos inicialmente y además se validó que la solución implementada resuelve satisfactoriamente el problema en cuestión. Para el framework software gateway esto se tradujo a verificar que la lógica está bien implementada y que cumple con las funcionalidades descritas anteriormente.

Las actividades realizadas en esta fase fueron las siguientes:

- Verificación de la lógica e implementación.
- Validación de funcionalidades propuestas.

4.5 Fase cinco: Implantación

Una vez se validó la integración del framework gateway en la plataforma IoT, empezó la fase de despliegue e implantación en minicomputadoras, en donde se hicieron pruebas de funcionamiento y portabilidad propias del framework.

Las actividades realizadas en esta fase fueron las siguientes:

- Despliegue del framework gateway en minicomputadoras.
- Pruebas de funcionamiento y portabilidad.
- Validación de integración gateway con plataforma IoT.

5. Arquitectura de referencia y requerimientos

5.1 Arquitectura de referencia

A continuación, se muestra la arquitectura de referencia, creada con base en las arquitecturas estudiadas durante la etapa de Capacitación tecnológica y representa la plataforma Smart Campus desde el punto de vista del usuario, es decir, es una guía para los usuarios que deseen implementar una aplicación utilizando la plataforma desarrollada.

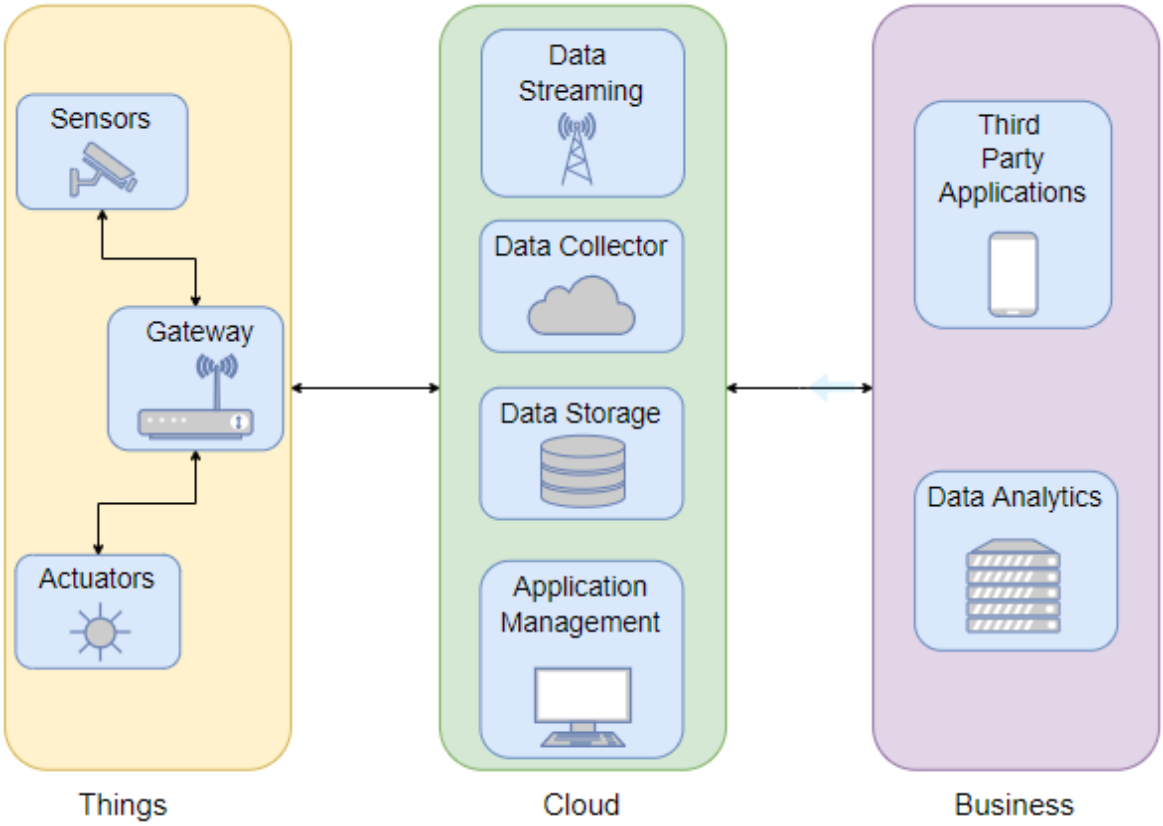

Figura 5. Arquitectura de referencia

Esta se compone por tres capas, la capa Things que representa la capa sensorial, y está compuesta por sensores, actuadores y gateways; la segunda es la capa Cloud donde se encuentran las aplicaciones de transmisión de datos, almacenamiento de información y gestión de aplicaciones;

finalmente la capa Business donde se encuentran las aplicaciones de terceros o externas y las aplicaciones de analítica de datos desarrolladas por los usuarios.

Para el desarrollo de una aplicación o caso de uso, los usuarios deben construir software para extraer información de sensores y recibirla en actuadores; esta información es enviada utilizando el software Gateway que se provee en la plataforma Smart Campus donde se procesa y envía la información al software Backend o Cloud, el cual a su vez la expone para que otras aplicaciones de usuarios transformen o procesen la información recibida y, finalmente ser mostrada mediante aplicaciones externas o ser enviada a aplicaciones de analítica de datos.

5.1 Requisitos funcionales

Identificación del requisito	RF01
Nombre del requisito	Servicio web para gestión de dispositivos
Descripción	Implementación de un servicio web tipo REST que permita la consulta, creación, actualización y eliminación de los dispositivos (gateway, sensores y actuadores) y sus propiedades.
Prioridad	Alta

Identificación del requisito	RF02
Nombre del requisito	Servicio web para gestión de procesos
Descripción	Implementación de un servicio web tipo REST que permita la consulta, creación, actualización y eliminación de procesos y sus propiedades.
Prioridad	Alta

Identificación del requisito	RF03

Nombre del requisito	Servicio web para gestión de mensajes
Descripción	Implementación de un servicio web tipo REST que permita consultar y eliminar mensajes en el gateway.
Prioridad	Alta

Identificación del requisito	RF04
Nombre del requisito	Servicio web para enviar mensajes a procesos
Descripción	Implementación de un servicio web tipo REST que permita dirigir mensajes hacia los procesos que contienen sensores y/o actuadores desde el exterior.
Prioridad	Alta

Identificación del requisito	RF05
Nombre del requisito	Comunicación de mensajes emitidos por procesos
Descripción	Implementación de mecanismos de comunicación de datos provenientes de procesos que contienen sensores y/o actuadores.
Prioridad	Alta

Identificación del requisito	RF06
Nombre del requisito	Persistencia local de mensajes emitidos por procesos
Descripción	Implementación de mecanismos en el framework que permitan almacenar y consultar mensajes generados por los procesos.
Prioridad	Alta

Identificación del requisito	RF07

Nombre del requisito	Consulta del estado del gateway y los procesos
Descripción	Implementación de un servicio web REST que permita consultar el estado del gateway y los procesos para determinar si están funcionando o no.
Prioridad	Media

Identificación del requisito	RF08
Nombre del requisito	Despliegue dinámico de procesos
Descripción	Implementación de un servicio web REST que permita desplegar dinámicamente los procesos compatibles con la JVM y que están ubicados en el mismo hardware gateway.
Prioridad	Media

5.2 Requisitos no funcionales

Por motivos de alcance del proyecto no se definieron requisitos relacionados con la seguridad en la manipulación de los datos.

Identificación del requisito	RNF01
Nombre del requisito	Extensibilidad del framework
Descripción	Ofrecer mecanismos de extensibilidad del framework de manera que se puedan añadir procesos que pueden contener sensores y actuadores y/u otras funcionalidades a este.
Prioridad	Alta

Identificación del requisito	RNF02
Nombre del requisito	Portabilidad del framework
Descripción	Permitir que el framework se pueda ejecutar en hardware diferentes.

Prioridad	Media

Identificación del requisito	RNF03
Nombre del requisito	Soporte de múltiples lenguajes en el framework
Descripción	Ofrecer la posibilidad de usar el framework desde diferentes lenguajes de programación con el objetivo de facilitar el desarrollo de los diferentes casos de uso.
Prioridad	Media

6. Desarrollo del proyecto

6.1 Contexto del proyecto

La visión de este proyecto es el diseño de un framework software extensible para dispositivos tipo gateway integrados en plataformas IoT para Smart Campus, por consiguiente, es importante resaltar que este proyecto de grado se desarrolló paralelamente junto con tres trabajos de grado más que consisten en la solución backend o cloud, la plataforma web de administración y el despliegue de la plataforma con el objetivo de facilitar la implementación de los casos de uso relacionados con el Internet de las Cosas orientados a Smart Campus como lo podría ser el control de la luminosidad de instalaciones, riego de zonas verdes controladas remotamente, control de temperatura en aulas de clase de manera centralizada y recolección de datos como humedad, emisión de gases, cantidad de personas en un determinado lugar, geolocalización de inventario, entre otros.

6.2 Capacitación tecnológica

6.2.1 Arquitecturas IoT

Para lograr resultados satisfactorios al momento de llevar a cabo el diseño del framework gateway junto con la plataforma general orientada a Smart Campus, fue importante analizar diferentes soluciones presentes hoy en día en cuanto a IoT.

Es en este punto donde aparece Azure IoT como la solución desarrollada por Microsoft para IoT, en donde se encontraron ideas con respecto a lo que las arquitecturas IoT deben proveer, como lo son:

- **Portabilidad:** Como en la página principal dice, se refiere a la posibilidad de usar cualquier dispositivo, sistema operativo, origen de datos, software o servicio, en el entorno local, en el perímetro o en la nube.
- **Escalabilidad:** Se refiere a la capacidad para poder añadir más dispositivos a la arquitectura sin temor de que esta falle o se haga más lenta.

La arquitectura IoT propuesta por Azure se muestra en la siguiente figura:

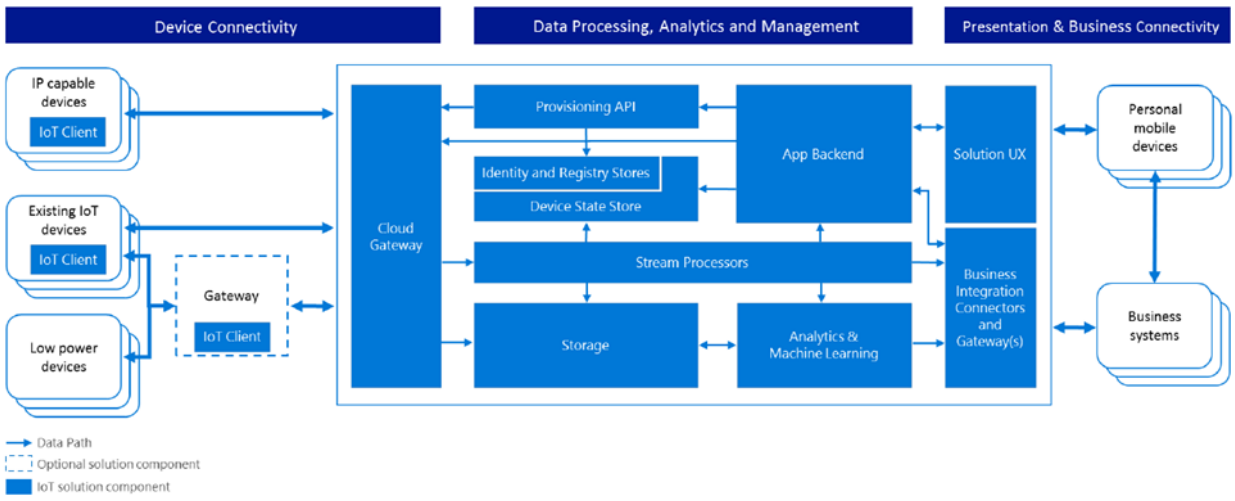

Figura 6. Arquitectura de referencia de Azure IoT. Tomada de Malone T. (2016).

Ver esta arquitectura fue realmente útil para obtener una idea de cómo modularizar la plataforma IoT para Smart Campus. Además, muestra a los gateways como una parte importante de la misma.

Posterior al estudio de Azure IoT se hizo el análisis de AWS IoT, la arquitectura IoT de Amazon en la cual se encontraron varias similitudes con la solución ofrecida por Microsoft, especialmente en algunos componentes de su estructura, listados a continuación:

- **Capa sensorial:** En esta existen todos los dispositivos finales como lo son sensores y actuadores encargados de interactuar con el mundo físico y de transformar lo que detectan en información útil para las plataformas IoT.
- **Capa de gateway:** En esta capa se muestran las funciones que pueden prestar los dispositivos tipo gateway a las soluciones IoT, sirviendo como canal de comunicación, procesamiento y almacenamiento local para la información de dispositivos finales, de manera que facilita la interacción de estos componentes con la arquitectura general.
- **Capa de backend o cloud:** Encargada de tener la visión global de todos los dispositivos desplegados. Es el punto de encuentro de todos los datos generados en los diferentes gateways, por lo cual es una estructura escalable y altamente disponible que puede permitir varios servicios como analítica de datos, aprendizaje de máquina, etc.
- **Capa de aplicación:** Permite gestionar los dispositivos gateways, sensores y actuadores desplegados en la plataforma, conocer su estado y gestionar reglas de tratamientos de información. Además, en esta capa también aparecen aplicaciones creadas por los usuarios para uso y visualización de los datos generados por sus casos de uso en la plataforma.

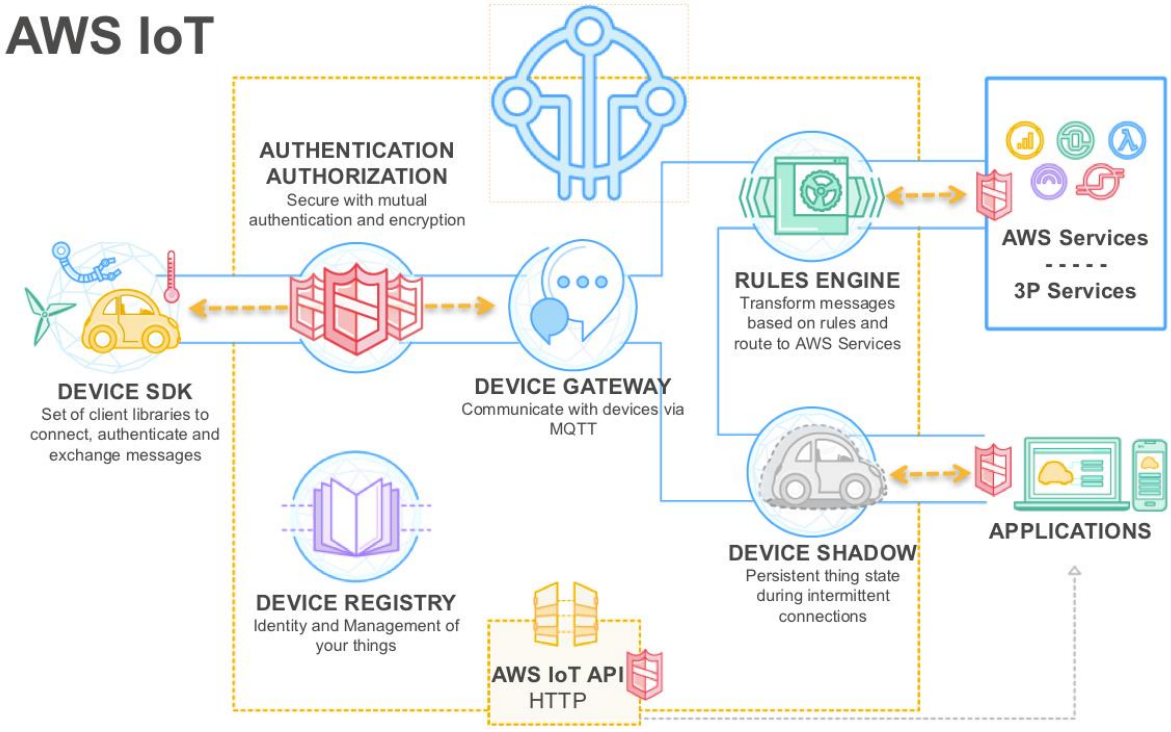

Figura 7. Arquitectura de AWS IoT. Tomada de McCauley, R. (2016).

6.2.2 Tecnologías y herramientas. En esta subsección se hablará del lenguaje de programación, los protocolos de comunicación, la base de datos y el framework seleccionados para el desarrollo del proyecto.

- Lenguaje de programación

Después de analizar algunos lenguajes de programación se encontró que JAVA es uno de los mejores exponentes cuando se trata de IoT por las siguientes razones:

- Java es un lenguaje de programación orientado a objetos.
- Java fue construido con capacidades útiles para aplicaciones IoT.
- Java es altamente portable y no tiene limitaciones de hardware.
- Al tener una larga historia cuenta con mucha documentación.

Según el subdominio o capa de IoT, la siguiente imagen muestra los lenguajes de programación preferidos:

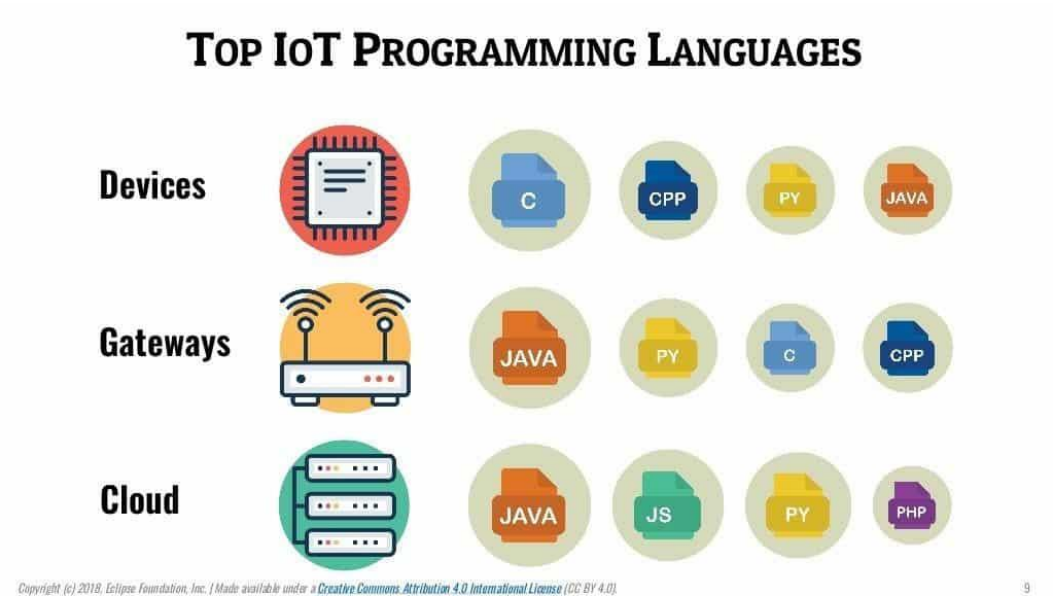

Figura 8. Lenguajes de programación preferidos para IoT. Tomada de Cuelogic. (2018).

Según Cuelogic (2018) estos resultados se obtuvieron de una encuesta realizada por Eclipse Foundation en la cual participaron 502 individuos, arrojando los siguientes porcentajes de preferencia en cuanto a lenguajes de programación para construir soluciones IoT:

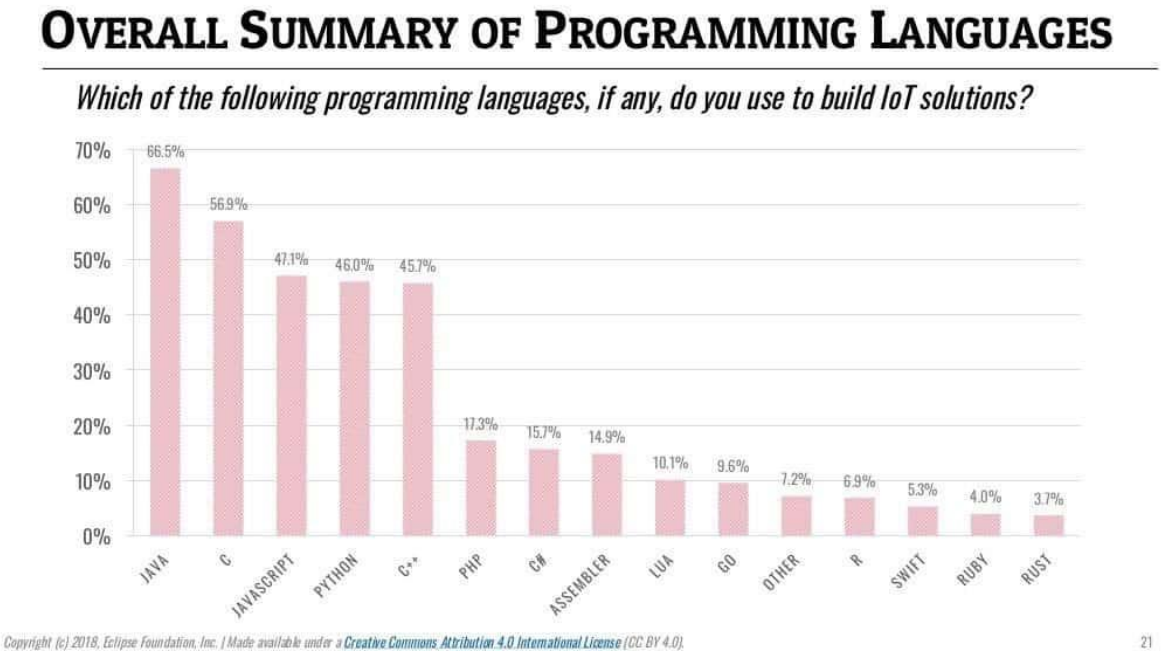

Figura 9. Lenguajes de programación preferidos para IoT en porcentaje. Tomada de Cuelogic. (2018).

- Protocolos de comunicación

HTTP: es un protocolo de la capa de aplicación para la transmisión de documentos hipermedia, como HTML. Fue diseñado para la comunicación entre los navegadores y servidores web, aunque puede ser utilizado para otros propósitos también. Sigue el clásico modelo cliente-servidor, en el que un cliente establece una conexión, realiza una petición a un servidor y espera una respuesta del mismo. Se trata de un protocolo sin estado, lo que significa que el servidor no guarda ningún dato entre dos peticiones. Aunque en la mayoría de casos se basa en una conexión del tipo TCP/IP, puede ser usado sobre cualquier capa de transporte segura o de confianza, es decir, sobre cualquier protocolo que no pierda mensajes silenciosamente, tal como UDP (Mozilla, s.f).

AMQP: (Advanced Message Queueing Protocol) es un protocolo de conexión de código abierto para mensajería asíncrona.

AMQP es eficiente, portable, multicanal y seguro. El protocolo binario ofrece autenticación y encriptación vía SASL o TLS, basado en un protocolo de transporte como lo es TCP. El protocolo de mensajería es rápido y garantiza la entrega de mensajes con acuse de recibo. AMQP funciona bien en entornos multi-cliente y provee un medio para delegar tareas y hacer que los servidores manejen solicitudes inmediatas más rápido. (Rouse, 2018).

TCP: El Protocolo de Control de Transmisión (TCP) permite a dos anfitriones establecer una conexión e intercambiar datos. TCP garantiza la entrega de datos, es decir, que los datos no se pierdan durante la transmisión y también garantiza que los paquetes sean entregados en el mismo orden en el cual fueron enviados (Masadelante, s.f).

- Base de datos

MongoDB: Es una base de datos orientada a documentos. Esto quiere decir que, en lugar de guardar los datos en registros, guarda los datos en documentos. Estos documentos son almacenados en BSON, que es una representación binaria de JSON (JavaScript Object Notation). Una de las diferencias más importantes con respecto a las bases de datos relacionales, es que no es necesario seguir un esquema. Los documentos de una misma colección pueden tener esquemas diferentes. (Genbeta, 2014).

- Frameworks

SpringBoot: Es un framework basado en Spring cuyo lenguaje de programación es JAVA. Se puede utilizar para desarrollar aplicaciones con menor dificultad, puesto que provee características que permiten a los desarrolladores preocuparse sólo por la lógica de negocio de sus aplicaciones y no de cosas como configuración, arranque y ejecución. Sus características principales son:

- Creación de aplicaciones stand-alone.
- Tomcat, Jetty o Undertow embebidos directamente.

- Dependencias de arranque provistas para simplificar la configuración del build.
- Dependencias de Spring y de terceros configuradas automáticamente.
- Características de producción provistas automáticamente.

Vert.x: Eclipse Vert.x es un conjunto de herramientas para construir aplicaciones reactivas en la JVM. Entre sus principales características se encuentra que:

- Es bastante ligero.
- Es rápido.
- No es de instancia monolítica.
- Es modular.
- Permite crear aplicaciones poderosas de manera sencilla.
- Es ideal para crear microservicios ligeros con alto desempeño.

6.3 Definición de la arquitectura

Se hace indispensable mencionar nuevamente la participación de los otros trabajos de grado que se desarrollaron paralelamente a este proyecto con el objetivo de buscar una solución IoT para Smart Campus de inicio a fin. Por ende, a continuación, se describe la arquitectura diseñada globalmente y más adelante se mostrará a detalle lo referente al framework gateway.

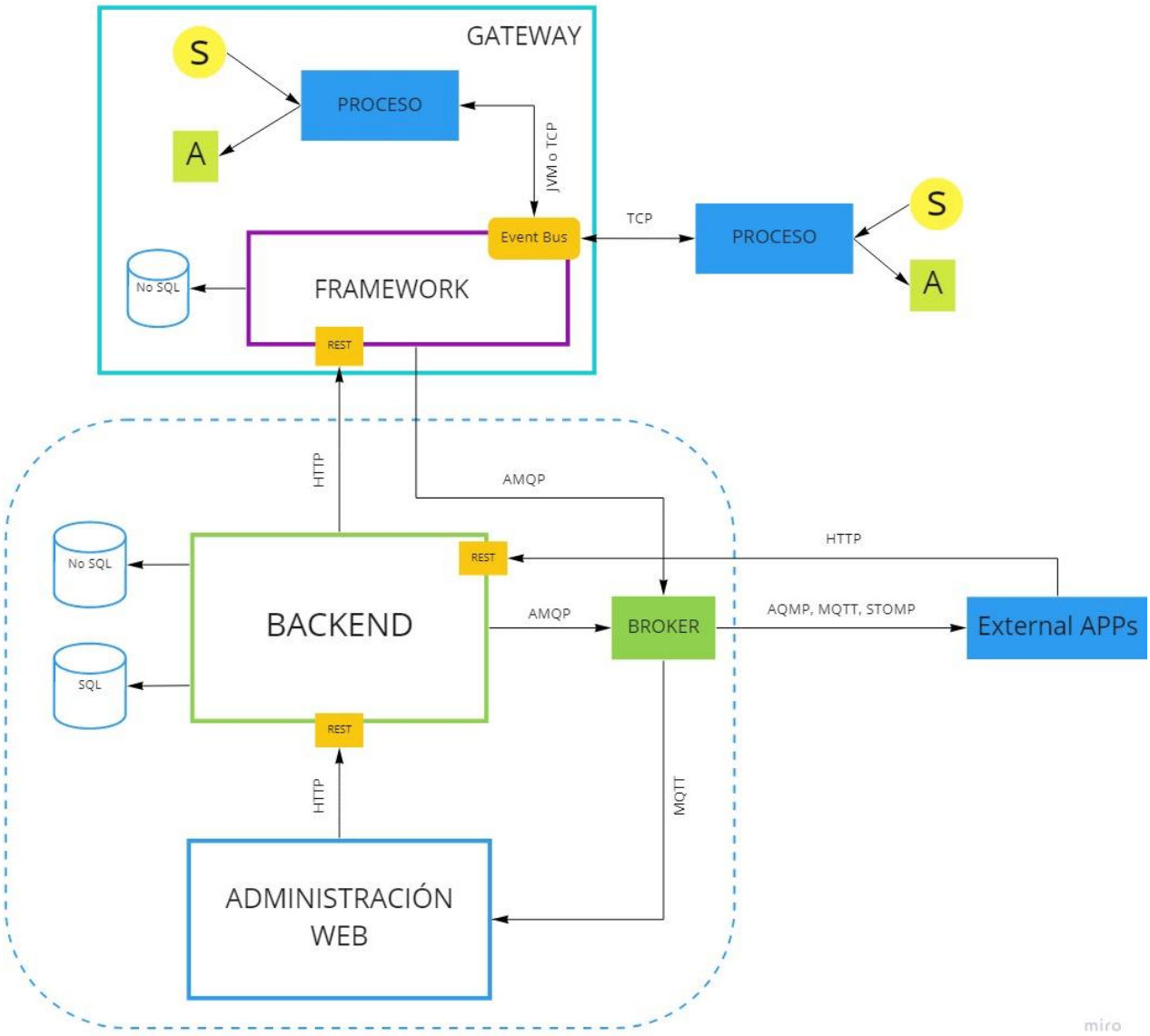

Figura 10. Arquitectura IoT para Smart Campus

Dichos trabajos de grado son listados a continuación:

- **Backend:** En este trabajo se presenta una solución Backend para una plataforma IoT, este software se desarrolló con una arquitectura de microservicios de alta disponibilidad elaborada en Java usando el framework Spring Boot, permite la integración de dispositivos y gateways a través de un broker y gracias a su escalabilidad puede manejar grandes

volúmenes de datos que se generen a partir de estos, almacenarlos y a través de un API REST ser consultados para el uso que se les quiera dar, además, cuenta con una unidad de persistencia para almacenar la información de los elementos que estén presentes en la infraestructura (dispositivos y gateways) (Arias y Estupiñan, 2019).

- **Administración Web:** En este proyecto de grado se presenta el diseño de una aplicación web que permite gestionar una arquitectura Smart Campus mediante la cual los usuarios pueden registrar y gestionar sus casos de uso y los dispositivos asociados a los mismos permitiéndoles de una manera sencilla crear aplicaciones que contribuyan a generar esta transformación digital (Camacho, 2019).
- **Despliegue:** En este trabajo de investigación se presenta el diseño de una infraestructura software para el despliegue de una plataforma IoT en una infraestructura cloud de alta disponibilidad en un entorno distribuido con el fin de proveer un entorno que pueda soportar cantidades masivas de solicitudes permitiendo una escalabilidad horizontal en la infraestructura hardware. (Rojas, 2019).

Ahora bien, se explicará a fondo el framework gateway cuyo desarrollo es el aporte que realiza este proyecto de grado en la arquitectura IoT enfocada a Smart Campus.

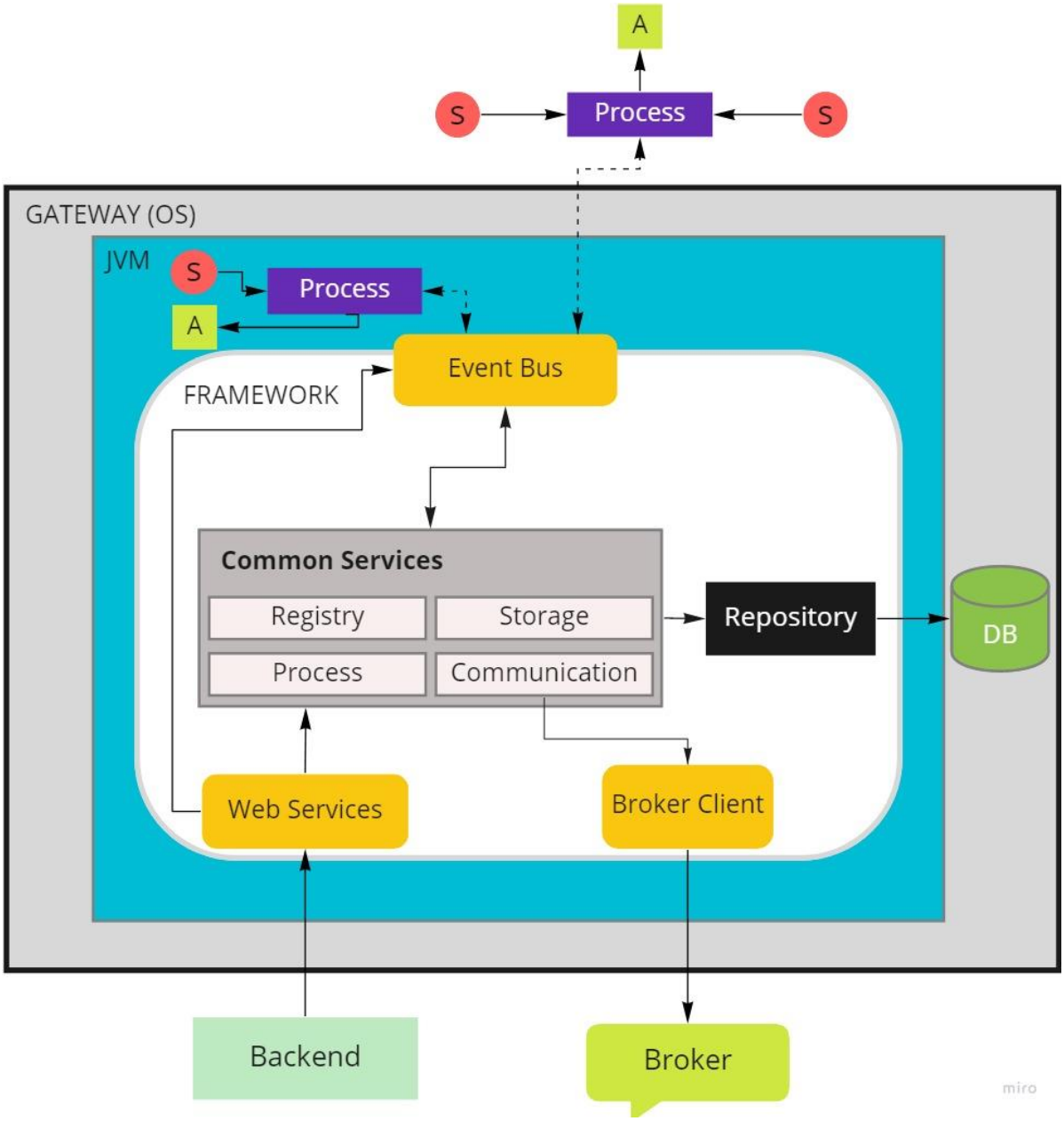

Figura 11. Arquitectura Gateway

Como se aprecia en la Figura 9, la arquitectura está compuesta de módulos bien definidos con sus labores específicas y serán explicados a continuación:

6.3.1 Web Services. Este módulo proporciona un canal de comunicación (puerta de entrada) para que usuarios externos (backend o cloud) puedan utilizar los servicios provistos por el framework, como lo son la gestión de los registros (gateway, sensores y actuadores), los procesos, los mensajes generados por los dispositivos finales y el envío de datos hacia los actuadores.

6.3.2 Common Services. Este módulo contiene cuatro submódulos listados a continuación con la lógica de negocio necesaria para ofrecer las funcionalidades definidas por el framework:

- **Registry:** Este submódulo se encarga de ofrecer consulta, inserción, actualización y eliminación de gateways, sensores y actuadores junto con sus propiedades, permitiendo la existencia de la representación lógica de estos dispositivos físicos. Vale la pena mencionar que una de las propiedades del gateway es la capacidad de limpiar periódicamente los mensajes ya enviados de la base de datos con el objetivo de liberar espacio.
- **Process:** Este submódulo se encarga de ofrecer consulta, inserción, actualización y eliminación de los procesos junto con sus propiedades de manera que los usuarios puedan tener control lógico sobre los mismos. Es importante mencionar que una de las propiedades de los procesos es que pueden indicar con qué frecuencia enviar los mensajes generados por los dispositivos finales; además este servicio ofrece la capacidad de desplegar procesos sobre el gateway de manera remota.
- **Storage:** Este submódulo se encarga del almacenamiento de los mensajes provenientes de los procesos que contienen dispositivos finales (sensores y actuadores) para su posterior uso.
- **Communication:** Este submódulo permite el envío de datos provenientes de los procesos que contienen sensores hacia el exterior y a su vez facilita la recepción de mensajes dirigidos

hacia los actuadores. Además, ofrece la capacidad de conocer el estado del gateway y los procesos con el fin de saber si están funcionando o no.

6.3.3 Repository. Este módulo es el encargado de establecer la conexión con la base de datos sirviendo de puente entre esta y el módulo Common Services con el fin de permitir que los datos que fluyen por el framework sean almacenados en dicha unidad de memoria no volátil.

6.3.4 DB (Base de datos). Este módulo es el encargado de almacenar y gestionar la información relacionada con el framework gateway a corto y largo plazo, facilitando la administración de los componentes involucrados en la arquitectura.

6.3.4 Broker. Si bien este módulo hace parte de la arquitectura externa al gateway, es necesario mencionarlo para entender mejor el flujo de los datos. El broker (o intermediario) cumple la función de recibir y entregar mensajes de diferentes usuarios (gateways) haciendo uso de tópicos y/o colas, por lo tanto, es acá a donde se envían los mensajes provenientes de los procesos que contienen sensores conectados al gateway para su posterior uso por parte del backend.

6.3.5 Broker Client. Este módulo contiene el cliente del broker, el cual permite el envío de mensajes generados por los procesos que contienen sensores a las colas (queues) de dicho broker por medio de un protocolo confiable y ligero.

6.3.6 Event Bus. El Event Bus (o bus de eventos) permite que exista comunicación entre componentes que no están previamente registrados, a través del estilo de comunicación publish-subscribe. Para el framework gateway esto habilita la comunicación bidireccional entre el framework en sí y los procesos desarrollados posteriormente sin importar el lenguaje en el que son implementados.

6.3.7 Process. Los procesos son las aplicaciones desarrolladas por usuarios de la plataforma IoT, es decir los casos de uso. Estos contienen toda la lógica necesaria para el manejo de sensores y actuadores, así como la capacidad de usar el framework gateway para comunicar, almacenar y consultar información o ser desplegados remotamente en el caso de que sean programados en lenguajes compatibles con la JVM. Estos procesos tienen la posibilidad de ser implementados en diferentes lenguajes de programación y pueden ser desplegados en el mismo hardware donde se ejecuta el framework gateway o por fuera de este.

6.3.8 Sensores. Los sensores son dispositivos finales que se conectan a los procesos para capturar los datos generados en el ambiente físico donde se encuentran ubicados y permitir su posterior procesamiento. El framework provee la capacidad de almacenar la representación lógica de dichos sensores en la base de datos.

6.3.9 Actuadores. Los actuadores son dispositivos finales que se conectan a los procesos para realizar determinadas acciones en el ambiente físico donde se encuentran ubicados, las instrucciones para su funcionamiento son entregadas por medio de los procesos. El framework provee la capacidad de almacenar la representación lógica de dichos actuadores en la base de datos.

6.4 Prototipado

En esta fase se tuvo en cuenta la arquitectura diseñada anteriormente y se implementó cada uno de los módulos propuestos.

6.4.1 Web Services. Para la implementación de los Web Services se hizo uso de Spring Boot a través de clases con la anotación @RestController que se encarga de proveer las herramientas necesarias para exponer endpoints a usuarios externos a través del protocolo de intercambio y manipulación de datos REST. El formato de los mensajes intercambiados es JSON. A continuación, se lista la tabla que contiene los endpoints implementados con detalles adicionales para su uso.

Tabla 1.
Endpoints REST ofrecidos por el framework gateway.

Servicio	**Método**	**Endpoint**	**Descripción**
Registry	GET	*/registry/registries*	Permite consultar los registros (gateway, sensores y actuadores) de la base de datos del gateway.
Registry	POST	*/registry*	Permite insertar registros (gateway, sensores y actuadores) en la base de datos del gateway.
Registry	PUT	*/registry*	Permite actualizar registros (gateway, sensores y actuadores) en la base de datos del gateway.
Registry	DELETE	*/registry*	Permite eliminar registros (gateway, sensores y actuadores) de la base de datos del gateway.
Process	GET	*/process/processes*	Permite consultar los procesos de la base de datos del gateway.
Process	POST	*/process*	Permite insertar procesos en la base de datos del gateway.
Process	PUT	*/process*	Permite actualizar procesos en la base de datos del gateway.
Process	DELETE	*/process/{processId}*	Permite eliminar procesos de la base de datos del gateway basado en el id.
Process	PUT	*/process/{processId}/ deploy/{deploy}*	Permite desplegar procesos sobre el gateway de manera remota. Estos procesos deben ser compatibles con la JVM y deben estar empaquetados con Maven. {processId} es el id del proceso y {deploy} es un booleano que indica si se debe prender o apagar el proceso.

Storage	GET	*/storage/messages*	Permite obtener los mensajes relacionados a un proceso que se encuentra almacenados en la base de datos del gateway.
Storage	DELETE	*/storage/messages*	Permite eliminar mensajes asociados a un proceso especifico de la base de datos del gateway.
Communication	POST	*/communication/process/ message*	Permite el envío de mensajes hacia los procesos desde el exterior.
Communication	GET	/communication/keepAlive	Retorna el estado del gateway y los procesos asociados a este para indicar si están funcionando correctamente o no.

Nota: Cada endpoint comienza con la ip o la dirección del server y el puerto de aplicación seguido de la ruta en la tabla, por ejemplo: *http://localhost:8080/registry/registries*. El Apéndice A contiene el API REST con información más detallada de los servicios.

6.4.2 Broker Client. Antes de hablar de la implementación de los servicios, primero se debe mencionar la implementación de este módulo, pues dichos servicios requieren de las funcionalidades presentes aquí.

Para este módulo se implementó una clase marcada con la anotación @Service pues se necesita sólo una instancia a lo largo de todo el framework para utilizar las funcionalidades provistas por el mismo; a su vez se creó una interfaz que define los métodos públicos del módulo cómo lo es el método para crear la conexión al broker, el método que permite enviar mensajes a las colas de dicho broker y un método para desconectarse del mismo.

Todo lo anterior se logró haciendo uso de una librería de SpringBoot llamada *spring-boot-starter-activemq* que provee los mecanismos necesarios de interacción con el broker por medio del protocolo AMQP.

6.4.3 Common Services. Para la implementación de los servicios también se usó SpringBoot, las clases fueron marcadas con la anotación @Service, la cual les da la característica de ser clases de una sola instancia que pueden ser inyectadas en cualquier lugar del framework para usar sus métodos. Además, para cada servicio se creó una interfaz que define los métodos que deben ser implementados y que son expuestos al resto de las clases. Esta capa es llamada directamente por los controllers (Web Services) cuando se hace una petición REST o por el Event Bus cuando los procesos necesitan enviar o recibir información.

Existen dos tipos de objetos especiales que se utilizan a lo largo de todo el framework:

- **DTOs:** Los Data Transfer Objects (Objetos de transferencia de datos) son objetos que contienen sólo los atributos útiles y públicos que pueden ser compartidos sin importar si el ámbito es interno o externo. La nomenclatura de estos objetos siempre es el nombre de la clase terminado en DTO, por ejemplo, *MessageDTO*.
- **Entities:** Las entities (entidades) son los objetos que representan el modelo en la base de datos, por lo tanto, a diferencia de los DTOs estos contienen atributos propios de las bases de datos y deben ser manejados sólo en el interior del framework para evitar exponer información sensible a los usuarios finales.

En este módulo se encuentra la lógica de negocio necesaria para que el framework realice las funcionalidades definidas previamente. Para no hacer larga la lectura a continuación se van a listar los submódulos con sólo las funcionalidades que complementan las descripciones de los Web Services:

- **Registry:** Cuando un gateway es creado o actualizado el framework realiza las siguientes tareas adicionales:

- Busca dentro de sus propiedades una propiedad llamada *broker_url*, esta propiedad indica la dirección del broker al cual deben ser enviados los datos generados por los procesos. Después revisa si no existe una conexión activa con ese broker para crearla a través del módulo Broker Client.
- En segundo lugar, busca una propiedad llamada *db_cleanup_time* (propiedad no obligatoria) la cual contiene el intervalo de tiempo (en minutos) para ejecutar la limpieza de los mensajes (ya enviados) en la base de datos con el objetivo de optimizar el uso de la memoria del gateway.
- Por último, se hace una consulta de los recursos del hardware donde se está ejecutando el framework con el objetivo de proveerle información útil al usuario. Estas propiedades se añaden a las propiedades enviadas por el usuario y son marcadas de tipo: *REPORTED*.
- **Process:** Cuando un proceso es creado o actualizado, el framework revisa si contiene la propiedad *batch_frequency* (opcional) para crear un job (trabajo) de la clase *MessageSenderRunnable*, la cual se encarga de enviar un conjunto de mensajes de un proceso especifico cuando pasa el intervalo de tiempo (en minutos) indicado en esta propiedad. Por otro lado, cuando un proceso es eliminado se revisa si este tenía un job de envío de mensajes asociado para eliminarlo; sí además este es un proceso que corre sobre la misma JVM del framework se detiene su ejecución.
- **Storage:** En este servicio no hay nada adicional a la descripción de los Web Services.
- **Communication:** Cuando un mensaje es enviado de un proceso, el framework consulta información relacionada con dicho proceso para buscar dentro de sus propiedades si fue configurado en frecuencia (*batch_frequency*) o en cantidad (*batch_amount*). Si ninguna de las dos propiedades existe esto quiere decir que el mensaje debe ser enviado

inmediatamente, por lo tanto, se envía al broker a través del Broker Client por AMQP y posteriormente el mensaje es guardado en base de datos con estado *ya enviado*. En cambio, si tiene la propiedad *batch_frequency* el mensaje se guarda en base de datos con estado *no enviado*, de manera que el job programado lo pueda consultar en el futuro para su envío junto con los demás mensajes no enviados del proceso cuando el tiempo configurado se cumpla. Por último, si tiene la propiedad *batch_amount* configurada se hace una consulta en la base para saber cuántos mensajes hay en cola y en caso de que el mensaje que acaba de llegar complete la cantidad preestablecida, todos estos mensajes son enviados en conjunto. En los dos últimos casos después de que ocurre el envío, los mensajes son actualizados en base datos con estado *ya enviado*.

6.4.4 Repository. Para lograr la implementación de este módulo se crearon 3 interfaces *IRegistryRepository*, *IProcessRepository* y *IStorageRepository* cada una de ellas extendió una clase provista por SpringBoot de la manera *MongoRepository<T, ID>* donde *T* es la entidad con la que se quiere trabajar e *ID* es el tipo del identificador del objeto T. Por ejemplo:

MongoRepository<Registry, Long>

Esta clase le indica a SpringBoot que la base de datos con la que se va a trabajar es MongoDB permitiéndole crear y optimizar consultas específicas para la misma, sin tener que usar explícitamente las consultas NoSQL de mongo.

A continuación, se hablará de un Repository de ejemplo que ilustra lo explicado anteriormente:

```
public interface IRegistryRepository extends MongoRepository<Registry, Long> {

  public Optional<Registry> findByIdBackendAndType(Long idBackend, EThingType type);
  public Long deleteByIdBackendAndType(Long idBackend, EThingType type);
```

```
}
```

Como se observa, la entidad con la que se está trabajando en esa clase es Registry (Gateway, Sensores y Actuadores) y el identificador de dichos dispositivos es de tipo *Long*.

El primer método se encarga de consultar un dispositivo basado en su id asignado desde el Backend y su tipo (*GATEWAY, SENSOR o ACTUATOR*).

El segundo método se encarga de eliminar un dispositivo basado en su id y su tipo (*GATEWAY, SENSOR o ACTUATOR*).

Como se aprecia, la consulta de las entidades en la base de datos se puede hacer de manera sencilla si se extiende la clase de *MongoRepository* y se usan sus métodos base.

6.4.5 DB (Base de datos). Para este módulo, como se ha mencionado anteriormente se utilizó MongoDB, pues en esta base de datos se puede almacenar diferentes estructuras de datos, lo cual permite una mayor flexibilidad a la hora de gestionar la información. En Mongo, las tablas se denominan Colecciones y los registros se denominan Documentos. La siguiente gráfica muestra la estructura de la base de datos que requiere el framework gateway para su funcionamiento:

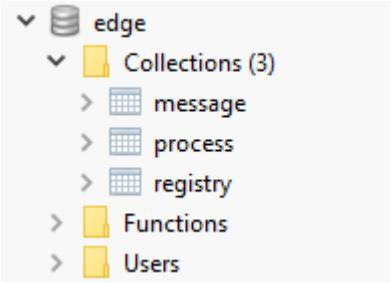

Figura 12. Representación de la base de datos

SpringBoot provee la capacidad de crear esta representación a partir del código mediante clases denominadas entidades o persistencias. Cada clase fue marcada con la anotación

@Document(collection=”nombredelacolección”) y los atributos definidos en su interior son campos de cada documento perteneciente a la colección *“nombredelacolección”* en la base de datos. Para cada colección existe un id que la identifica en la base de datos.

Vale la pena mencionar que los documentos de las colecciones process y registry tienen una lista de propiedades embebidas que sirven para que el usuario almacene información útil con respecto a cada documento y su estructura es la siguiente:

```
"properties": [
      {
         "name": "nombredepropiedad",
         "value": "valordepropiedad",
         "type": "tipodepropiedad"
      },
      {
         "name": "nombredepropiedad",
         "value": "valordepropiedad",
         "type": "tipodepropiedad"
      }
  ]
```

6.4.6 Event Bus. En este módulo se introduce una de las partes más importantes del framework gateway como lo es el Event Bus para permitir la comunicación bidireccional con los procesos. Para lograr esto se usó Vert.x en donde se creó una clase *VertxHandler* que implementa la interfaz *IVertxHandler*.

La interfaz obliga a la clase a implementar 3 métodos específicos:

- publishToTopic: Es el método encargado de enviar datos que provienen del exterior hacia los procesos.
- deployProcess: Es el método que contiene la lógica para desplegar procesos programáticamente.

- checkProcessAlive: es el método que le permite al framework gateway conocer el estado de los procesos (activo o inactivo).

Por su lado la clase es la encargada de la manipulación del Event Bus que se ejecuta sobre la JVM donde se encuentra alojado el framework, exponiéndolo sobre un puerto del gateway (puerto 7000 en este caso) para que pueda ser accedido por medio de TCP.

El funcionamiento de este Event Bus es a través de tópicos siguiendo la metodología "publish/subscribe". Mediante Vert.x se expone 5 tópicos para proveer ciertas funcionalidades a los procesos y del mismo modo los procesos exponen tópicos del lado de ellos para permitir la comunicación bidireccional.

A continuación, se muestra la tabla que describe los tópicos expuestos por el framework:

Tabla 2.
Tópicos del Framework en el Event Bus

Tópico	**Descripción**
sendMessage	Se encarga de recibir los mensajes provenientes de los procesos que quieren ser enviados hacia el exterior.
getRegistries	Retorna los registros existentes en la base de datos de manera que los procesos pueden hacer uso de la información almacenada en ellos.
getProcessById	Retorna la representación lógica de proceso especifico basado en su id, de manera que los procesos desplegados pueden usar las propiedades almacenadas en él.
getMessagesByProcessId	Retorna los mensajes asociados a un proceso especifico, permitiéndole a los procesos tomar decisiones basadas en un conjunto histórico de mensajes.
saveMessage	Almacena el mensaje enviado desde los procesos en la base de datos para su posterior uso.

6.4.7 Process. Como se dijo anteriormente, los procesos son la representación de los casos de uso finales desarrollados por los usuarios del framework gateway y por ende de la plataforma IoT. Dichos procesos pueden ser implementados con cualquier lenguaje de programación que tenga compatibilidad con la librería de Vert.x y por ende acceso al Event Bus.

Los procesos pueden ser desplegados bien sea dentro de la JVM donde se ejecuta el framework, por fuera de la JVM, pero sobre el mismo hardware en donde se encuentra el framework o totalmente afuera del gateway.

Para el primer caso se decidió crear un proyecto esqueleto en JAVA que facilita la implementación de los casos de uso hechos con este lenguaje de programación. Este tipo de proceso se puede comunicar directamente con el framework sin necesidad de protocolos intermedios lo cual aumenta la velocidad de comunicación y da la posibilidad de ser desplegado remotamente. Los detalles acerca de este proyecto se adjuntan en el Apéndice B.

Para el segundo y el tercer caso, la implementación de los procesos queda totalmente a juicio de los usuarios, pero se debe tener en cuenta que la comunicación con el framework se debe realizar mediante TCP.

En general, cualquier proceso debe subscribirse a dos tópicos esenciales para el correcto funcionamiento de los casos de uso. La tabla a continuación describe estos tópicos:

Tabla 3.
Tópicos de los procesos en el Event Bus

Tópico	**Descripción**
{processTopic}	El *processTopic* es un atributo perteneciente al proceso que indica la dirección a la cual estará subscrito para recibir mensajes dirigidos hacia él. Un ejemplo sería *temperatura/centic* por ende este proceso es la representación del caso de uso de temperatura.

keepAlive/{processId}	El valor processId representa el id del proceso al momento de su creación mediante los Web Services. Un ejemplo sería *keepAlive/2* lo que indica que el proceso con id 2 está recibiendo peticiones de estado en este tópico. Los usuarios pueden crear lógica específica para responder si están funcionando o no.

6.4.8 Extensibilidad. La extensibilidad del framework gateway fue lograda mediante el uso de Vert.x, el cual genera unidades de despliegue independientes denominados Verticles que permiten añadir o retirar funcionalidades. Para explicar mejor esto, se hará uso de la siguiente gráfica:

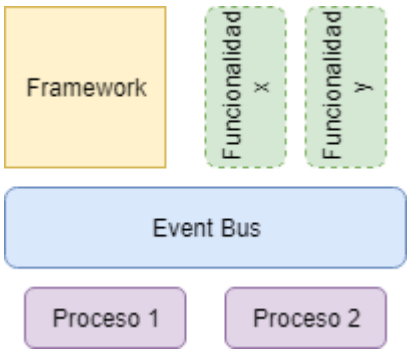

Figura 13. Extensibilidad del framework

En la figura anterior vemos que el intercambio de mensajes entre procesos y framework ocurre mediante el Event Bus permitiendo ejecutar nuevas unidades de despliegue (verticles) que representen casos de uso finales capaces de interactuar con el framework. Este mismo concepto se puede aplicar para agregar nuevas funcionalidades o características al framework gateway comunicando lo nuevo con lo antiguo mediante el Event Bus.

Un detalle importante es que las nuevas funcionalidades pueden ser programadas en diferentes lenguajes de programación, por lo cual el usuario tiene la posibilidad de escoger el que mejor le parezca, eso sí, siempre y cuando sea compatible con Vert.x y el Event Bus.

6.5 Validación

En esta fase se describen los escenarios planteados para validar el funcionamiento del framework gateway con base en los requerimientos funcionales y no funcionales descritos en el Capítulo 5. Sin embargo, la validación de los requisitos no funcionales RNF02 - Portabilidad del framework y RNF03 - Soporte de múltiples lenguajes en el framework se mostrará en el siguiente capítulo (Caso de uso: Implementación e implantación).

Para la validación del prototipo se utilizó el mismo equipo de cómputo usado para el diseño y desarrollo del framework gateway. A continuación, se muestran las especificaciones del mismo:

Tabla 4.
Especificaciones equipo de cómputo

Tipo	**Referencia**
Procesador	Intel Core i7-5500U 2.4GHz
Arquitectura del procesador	64 bits, procesador x64
Memoria RAM	8GB DDR3
Almacenamiento	SSD 240GB

Tabla 5.
Versiones del software utilizado

Software	**Versión**
Sistema operativo	Windows 10 Pro
Eclipse	Eclipse Photon

Robo 3T	1.2.1
Postman	6.7.4
ActiveMQ Broker	5.15.8

Nota: La validación sobre minicomputadoras se verá en el siguiente capítulo junto con su conexión a la plataforma IoT orientada a Smart Campus.

A continuación, se muestra el estado inicial de la base de datos:

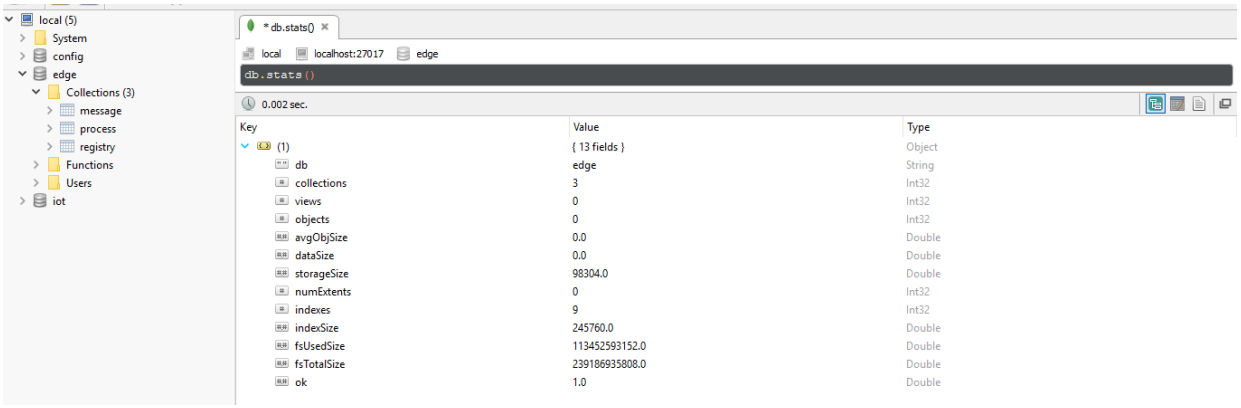

Figura 14. Estado inicial de la base de datos

Como se observa, la base de datos del gateway (edge) se encuentra vacía a excepción de las 3 colecciones creadas automáticamente por medio de SpringBoot al encender el framework gateway.

6.5.1 Pruebas de gestión de dispositivos. En esta prueba se validó el requisito funcional RF01 – Servicio web para gestión de dispositivos. Para efectos de la prueba, se trabajó con dispositivos tipo gateway, sin embargo, se deja claro que los sensores y los actuadores funcionan de la misma manera.

Creación del gateway:

Para la creación del gateway se utilizó el método *POST* en el endpoint *http://localhost:8080/registry*, a continuación, se encuentra el body de la solicitud:

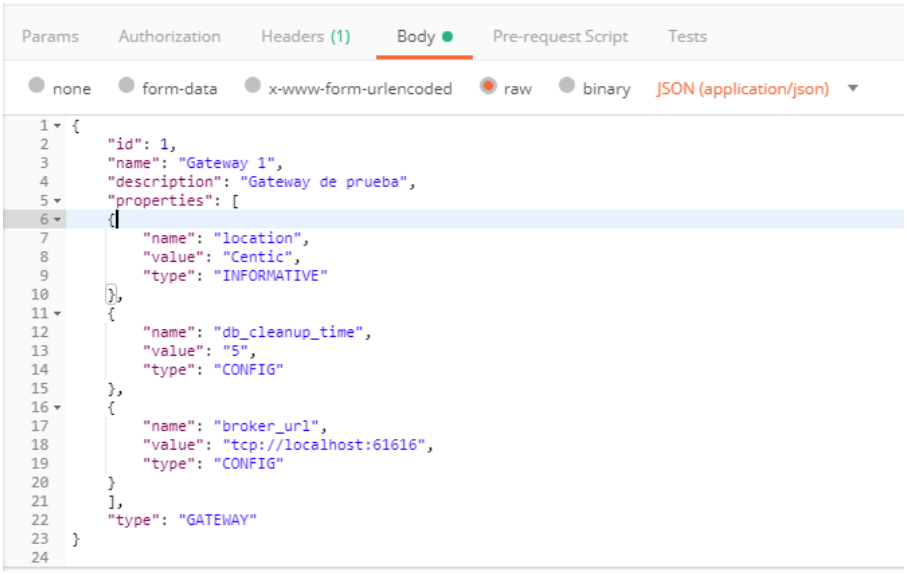

Figura 15. Solicitud de creación de un gateway

El gateway creado contiene dos propiedades de configuración *db_cleanup_time* y *broker_url* y una informativa *location.*

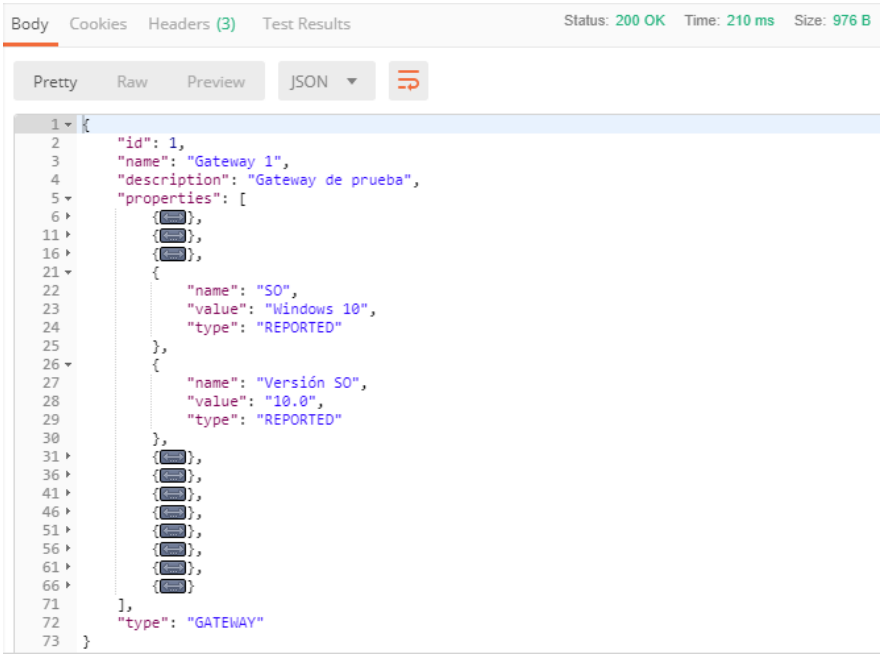

Figura 16. Respuesta de creación de un gateway

En la figura anterior se puede apreciar que al momento de crearse el gateway como se mencionó en capítulos anteriores, el framework agregó propiedades de tipo REPORTED que ofrecen información acerca del hardware y software del dispositivo gateway en donde se está ejecutando.

Por otro lado, como se aprecia en la siguiente figura al momento de crear el gateway, la conexión con el broker fue establecida y se configuró el job para limpiar los mensajes con estado *ya enviado* de la base de datos cada 300000ms es decir cada 5 minutos como se indicó en la solicitud.

```
o.s.data.mongodb.core.MongoTemplate      : find using query: { "type" : "GATEWAY" } fields: Document{{}} for cl
o.s.data.mongodb.core.MongoTemplate      : Inserting Document containing fields: [name, description, type, prop
c.u.i.edge.core.service.ActiveMQService  : Connection established with the broker:tcp://localhost:61616
c.u.i.edge.core.service.RegistryService  : The Db Cleaner task was configured every 300000 ms.
o.s.data.mongodb.core.MongoTemplate      : Remove using query: { "alreadySent" : true } in collection: message.
org.zalando.logbook.Logbook              : Outgoing Response: b5db0a684fee26d4
```

Figura 17. Log de la creación del gateway

Para concluir esta prueba, ahora se muestra en la base de datos el documento del nuevo gateway creado dentro de la colección *registry:*

```
db.getCollection('registry').find({})
```

registry 0.001 sec.

Key	Value
(1) ObjectId("5cda0d311904e71454bee5d7")	{ 7 fields }
_id	ObjectId("5cda0d311904e71454bee5d7")
name	Gateway 1
description	Gateway de prueba
type	GATEWAY
properties	[13 elements]
idBackend	1
_class	co.uis.iot.edge.core.persistence.Registry

Figura 18. Gateway creado en la base de datos

Consulta de registros:

Para la consulta de los registros se utilizó el método *GET* en el endpoint *http://localhost:8080/registry/registries*, a continuación, se encuentra la respuesta:

Figura 19. Respuesta de la consulta de registros

En la respuesta se verifica que el framework retorna los dispositivos agregados en la base de datos, que por el momento sólo era uno. Las propiedades fueron comprimidas para efectos de la figura, pero se asegura que se recibieron tanto las propiedades creadas por el usuario como las que el framework reporta.

Actualización de gateway:

Para la actualización del gateway se utilizó el método *PUT* en el endpoint *http://localhost:8080/registry*, a continuación, se encuentra el body de la solicitud:

Figura 20. Solicitud de actualización de gateway

Se actualizaron los atributos *name*, *description* y la propiedad *db_cleanup_time* para que el framework hiciera la limpieza cada 10 minutos en vez de cada 5. A continuación, se muestra la respuesta del servicio:

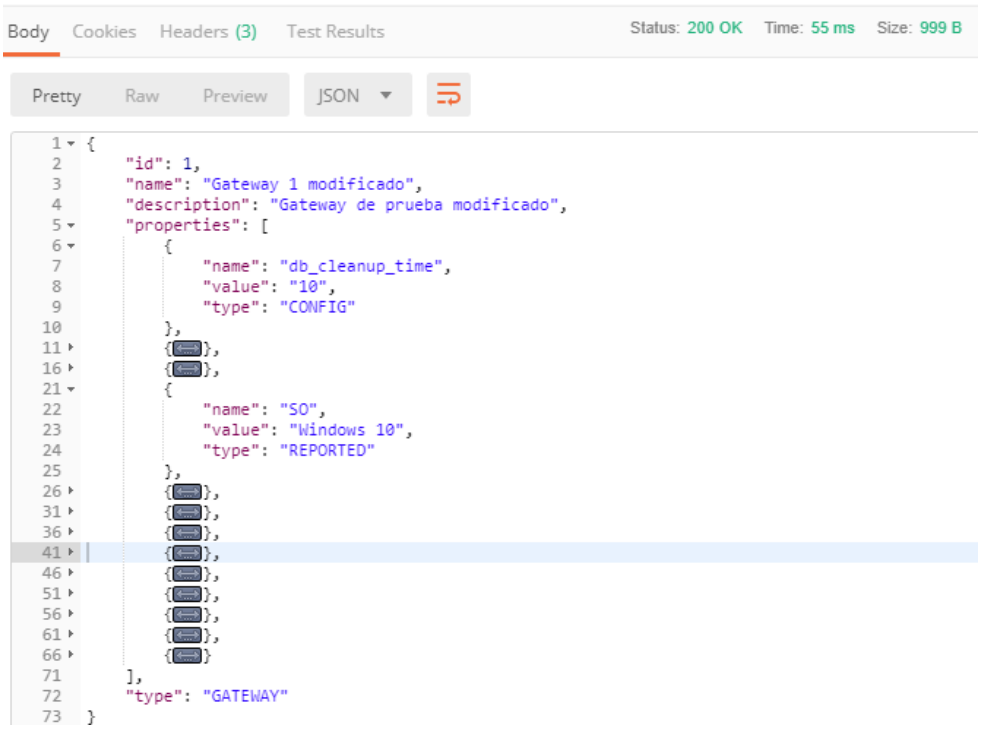

Figura 21. Respuesta de la actualización del gateway

Al igual que en la creación se ve que el framework agregó de nuevo las propiedades reportadas del gateway.

Al mirar de nuevo la consola se encontró que el anterior job de limpieza de base de datos fue cancelado y uno nuevo fue creado, pero esta vez configurado cada 10 minutos tal cual como se indicó en la solicitud.

```
o.s.data.mongodb.core.MongoTemplate      : Saving Document containing fields: [_id, name, description, ty
c.u.i.edge.core.service.RegistryService  : The Db Cleaner task was canceled.
c.u.i.edge.core.service.RegistryService  : The Db Cleaner task was configured every 600000 ms.
```

Figura 22. Log de actualización del gateway

Para terminar esta prueba se mostrará a continuación una imagen del estado de la base de datos en donde se demuestra que se realizaron los cambios enviados en la solicitud:

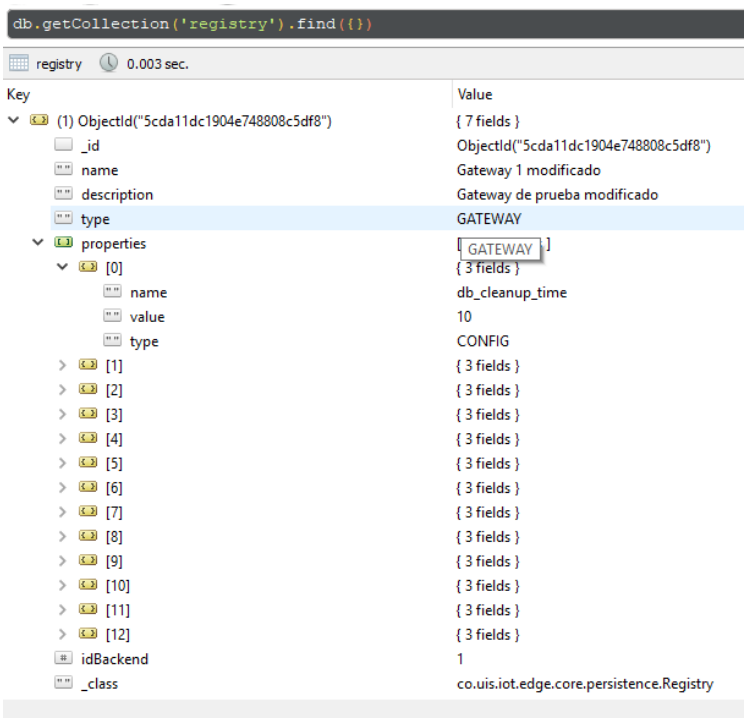

Figura 23. Gateway actualizado en la base de datos

Eliminación del Gateway:

Para la eliminación del gateway se utilizó el método DELETE en el endpoint *http://localhost:8080/registry?id=1&type=GATEWAY*.

Como se puede observar en el mismo endpoint se pasan los parámetros del registro a eliminar, los cuales son id y tipo. La respuesta de este servicio es un body vacío pero un *Status 200 ok.*

Para verificar el correcto funcionamiento de este método a continuación se muestra la base de datos con la colección *registry* vacía:

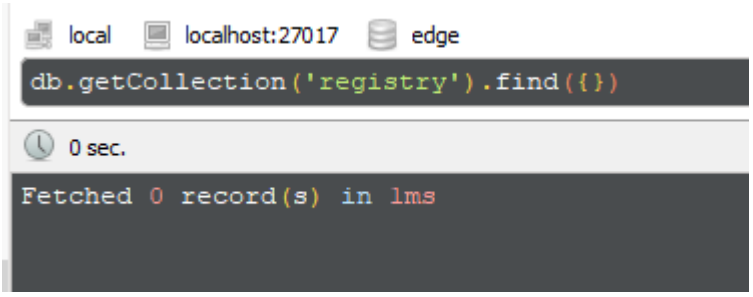

Figura 24. Gateway eliminado de la base de datos.

6.5.2 Pruebas de gestión de procesos. En esta prueba se validó el requisito funcional RF02 – Servicio web para gestión de procesos. Para efectos de la prueba, se trabajó un proceso compatible con la JVM.

El estado inicial de la base de datos para esta prueba contiene un gateway creado, pues esta representación lógica es necesaria para poder operar sobre el framework.

Creación de proceso:

Para la creación del proceso se utilizó el método *POST* en el endpoint *http://localhost:8080/process*, a continuación, se encuentra el body de la solicitud y la respuesta:

Figura 25. Solicitud para crear un proceso

Como se observa en la petición, el proceso fue configurado para recibir mensajes por medio del Event Bus en el tópico *proceso1*, además se configuró la propiedad *batch_amount*, lo que quiere decir que el framework recibirá mensajes emitidos por este proceso, pero no los enviará hacia el broker externo inmediatamente sino hasta que se completen bloques de cinco mensajes. Por último, se configuró la propiedad *process_jar* con la ruta del repositorio de Maven donde se encuentra empaquetado la lógica del proceso, en el Apéndice B se habla sobre este tipo de procesos.

A continuación, se muestra la respuesta del framework para la creación:

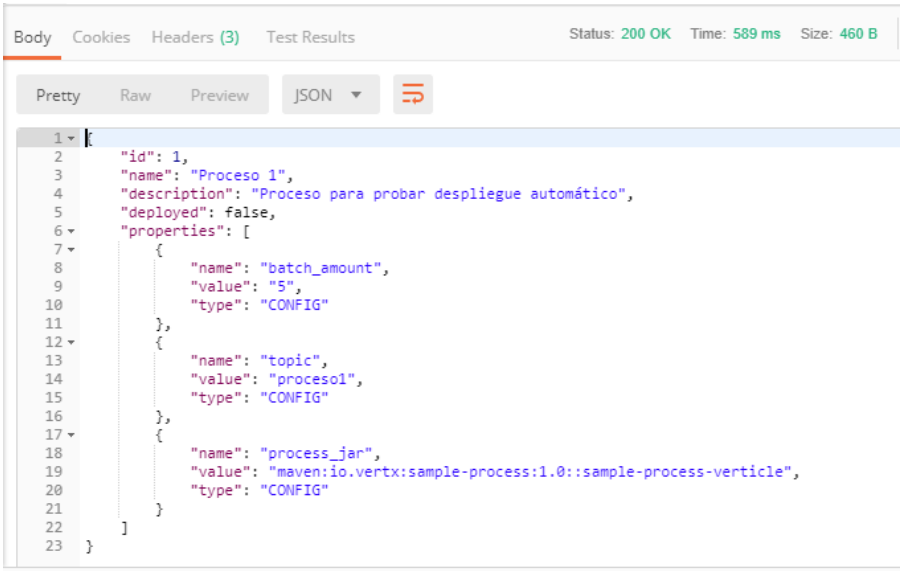

Figura 26. Respuesta de la creación de un proceso

La respuesta del framework confirma la correcta creación del gateway y además retorna un atributo llamado *deployed* con valor igual a *false*, lo que quiere decir que el proceso fue creado, pero aún no ha sido desplegado.

Por último, se muestra la imagen de la base de datos, en donde se encuentra el proceso creado:

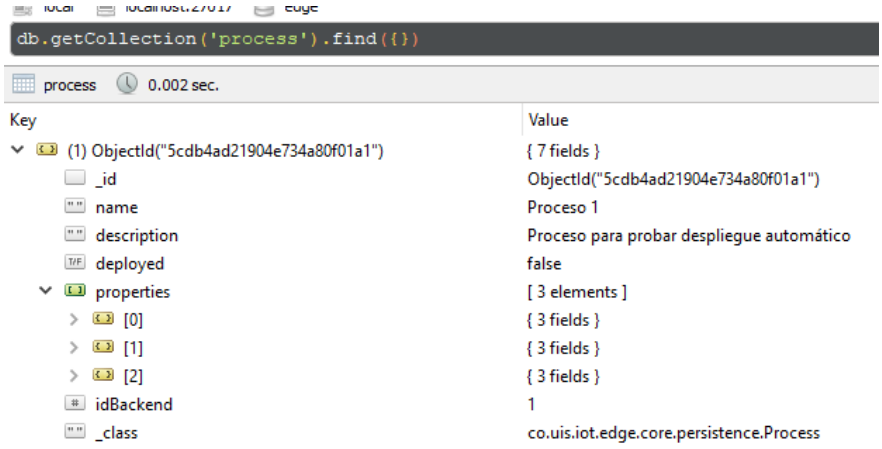

Figura 27. Base de datos con proceso creado

Actualización de proceso:

Para la actualización del proceso se utilizó el método *PUT* en el endpoint *http://localhost:8080/process*, a continuación, se encuentra el body de la solicitud y la respuesta:

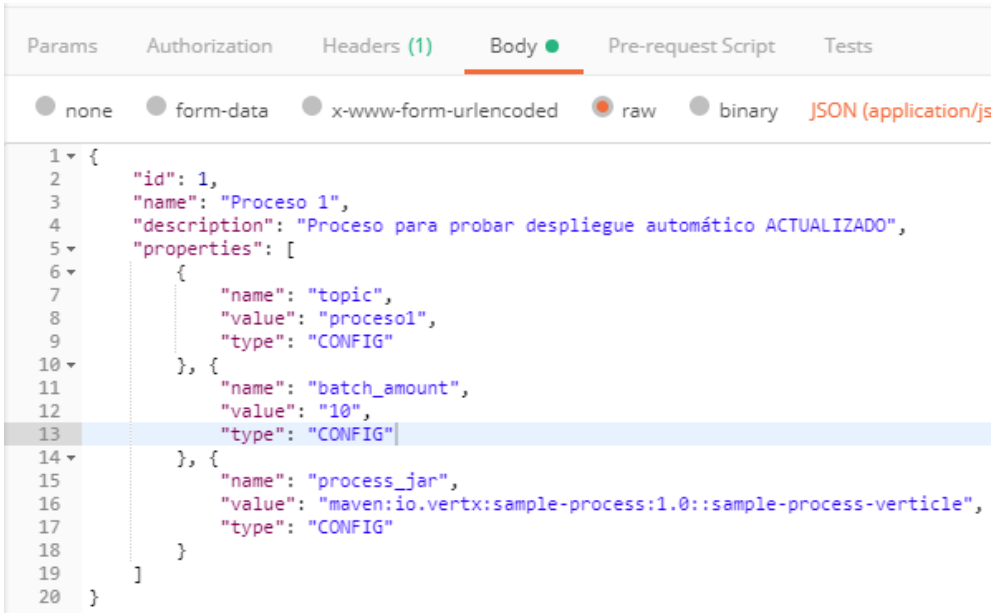

Figura 28. Actualización de un proceso

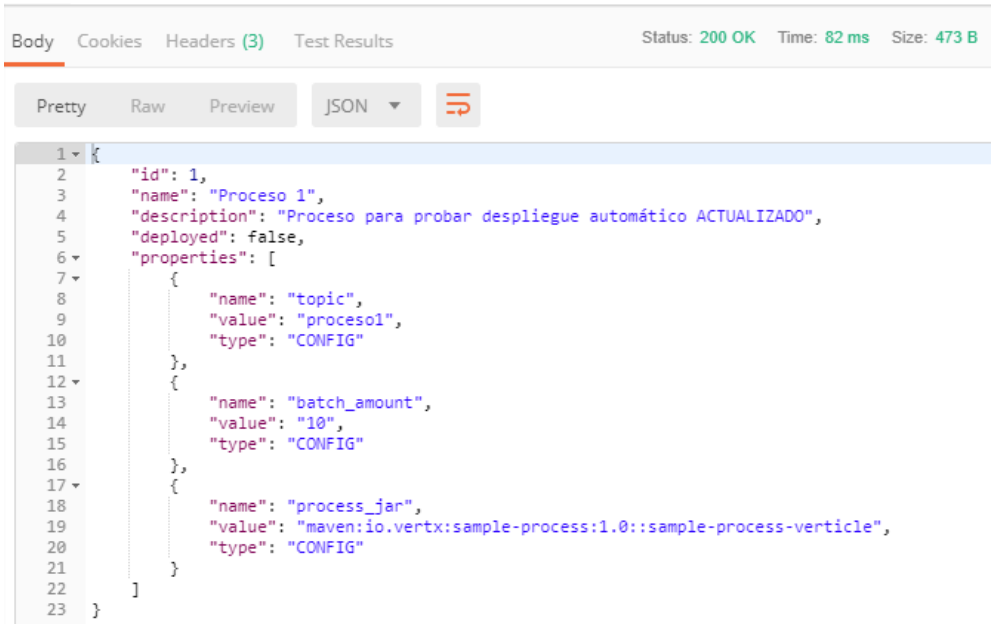

Figura 29. Respuesta de actualización de un proceso

Como se aprecia en la respuesta la descripción y la propiedad *batch_amount* fueron actualizados satisfactoriamente. A continuación, se muestra la imagen de la base de datos con dicha información actualizada:

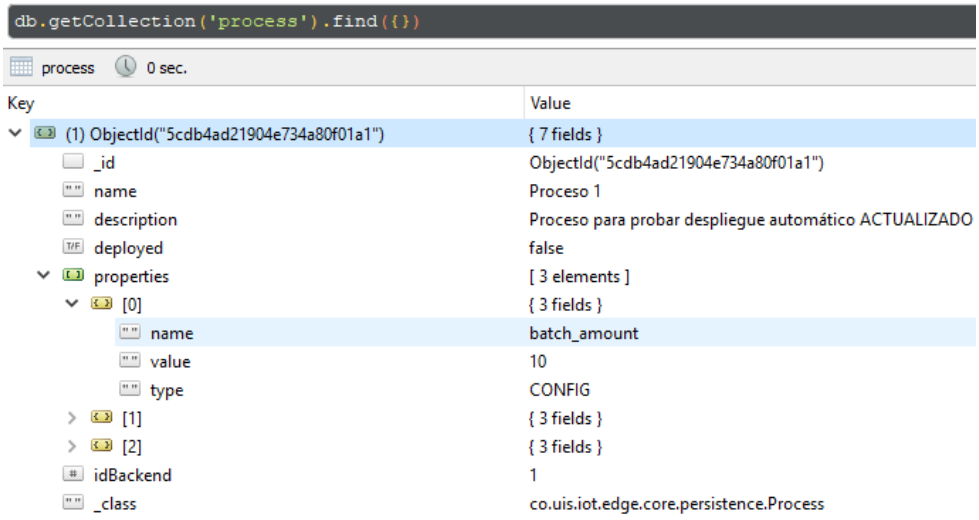

Figura 30. Proceso actualizado en base de datos

Consulta de procesos:

Para la consulta de los procesos se utilizó el método GET en el endpoint *http://localhost:8080/process/processes?id=1*, en donde *id=1* significa que se quiere consultar el proceso con ese id, a continuación, se encuentra la respuesta a esta solicitud:

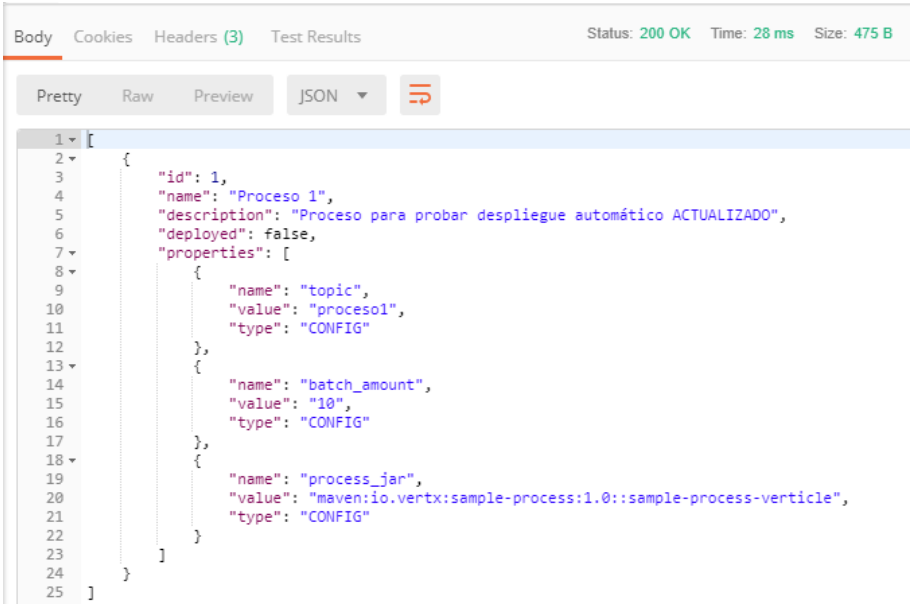

Figura 31. Respuesta de la consulta de un proceso

En la respuesta se observa que los datos retornados corresponden a los ingresados anteriormente. Si se desean consultar todos los procesos registrados en el framework gateway, se debe utilizar el método *GET* en el endpoint *http://localhost:8080/process/processes*.

Eliminación del proceso:

Para la eliminación del proceso se utilizó el método *DELETE* en el endpoint *http://localhost:8080/process/1*.

El valor *1* en el endpoint indica el id del proceso que se desea eliminar. La respuesta de este servicio es un body vacío pero un *Status 200 ok.*

Para verificar el correcto funcionamiento de este método a continuación se muestra la base de datos con la colección *process* vacía:

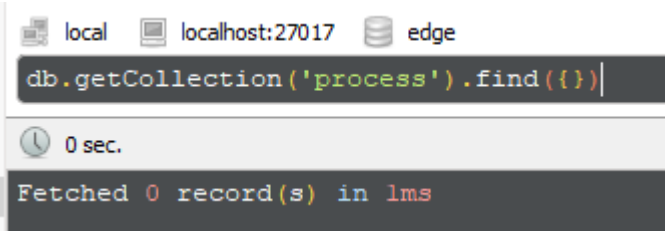

Figura 32. Proceso eliminado de la base de datos.

6.5.3 Prueba de despliegue dinámico. En esta prueba se validó el requisito funcional RF08 – Despliegue dinámico de procesos y el requisito no funcional RNF01 - Extensibilidad del framework.

El punto inicial de esta prueba es el gateway creado en la base de datos junto con el proceso, pero con la propiedad *batch_amount* configurada con el valor 2.

Una vez el proceso fue empaquetado en el repositorio local de Maven, para el despliegue se utilizó el método *PUT* en el endpoint *http://localhost:8080/process/1/deploy/true*, el body de la solicitud se envió vacío. A continuación, se muestra la respuesta:

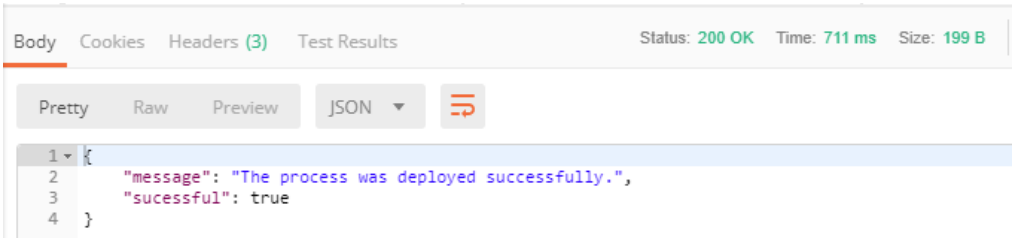

Figura 33. Despliegue de proceso dinámicamente

Como se observa el resultado es que el proceso fue desplegado correctamente, esto se puede comprobar revisando la consola, pues allí la lógica del proceso indica que fue desplegado correctamente desde la clase *SampleProcessVerticle:*

```
o.s.data.mongodb.core.MongoTemplate      : find using query: { "idBackend" : { "$numberLong" : "1" } } fields:
vertx-stack-resolver                     : Resolving uis.vertx.sample.process:sample-process:jar:1.0
c.u.i.edge.common.model.ProcessVerticle  : [consumeMessages] Waiting for messages in the topic: proceso1
c.u.i.edge.common.model.ProcessVerticle  : [keepAliveListener] Waiting for keepalive request for the process: 1
u.v.s.process.SampleProcessVerticle      : The process 1 has been deployed.
```

Figura 34. Proceso desplegado satisfactoriamente

En la figura anterior se puede observar que el framework gateway busca el proceso para desplegarlo y cuando lo despliega, el proceso empieza a ejecutar la lógica con la que fue programado. Hay 3 mensajes importantes que el proceso imprime en la consola, el primero es que está listo para recibir mensajes en el tópico *proceso1*, el segundo mensaje dice que está listo para responder peticiones de keepAlive (estado) y por último indica que fue desplegado.

La última tarea que realizó el framework en este despliegue, fue actualizar el proceso en base de datos para configurar la propiedad *deployed = true*.

6.5.4 Envío de mensajes desde los procesos. En esta prueba se validó el requisito funcional RF05 - Comunicación de mensajes emitidos por procesos.

El punto inicial de esta prueba es la colección de mensajes vacía en la base de datos y la cola de mensajes vacía en el broker como se muestra a continuación:

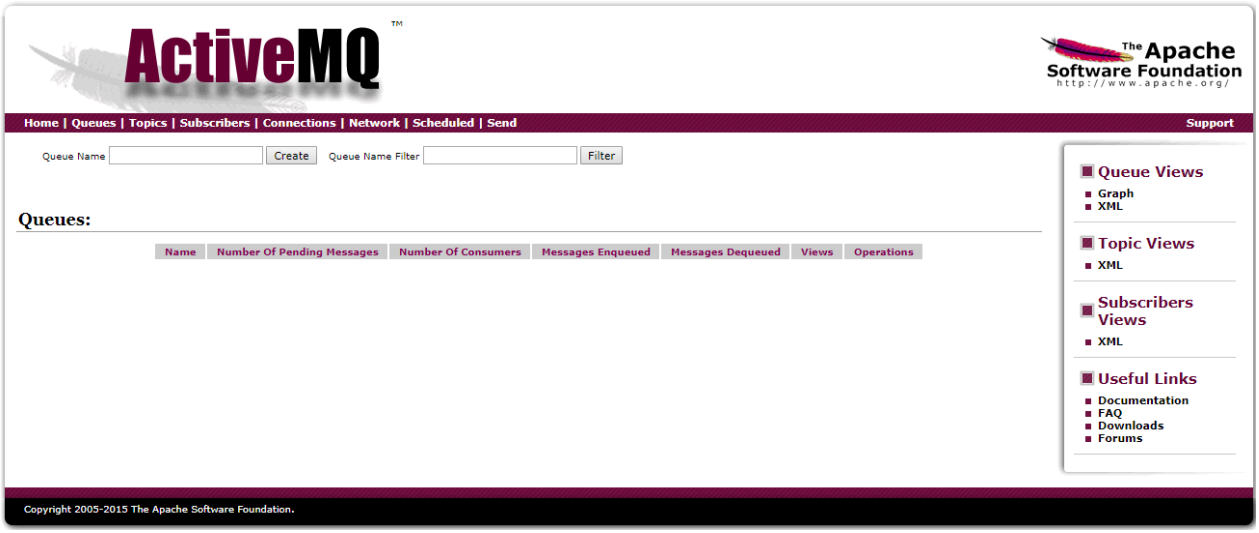

Figura 35. Estado inicial del broker

Una vez fue desplegado el proceso creado en la subsección anterior (con *batch_amount = 2)*, este empezó a generar datos para ser enviados hacia el exterior a través del Event Bus como se aprecia en la consola mostrada a continuación:

```
co.uis.iot.edge.core.vertx.VertxHandler  : sendMessage method called with message: {"gatewayId":null,"processId":1,"payl
u.v.s.process.SampleProcessVerticle      : [sendMessage] Successfuly Response: "The message was processed successfully"
o.s.data.mongodb.core.MongoTemplate      : find using query: { "type" : "GATEWAY" } fields: Document{{}} for class: clas
o.s.data.mongodb.core.MongoTemplate      : find using query: { "processId" : { "$numberLong" : "1" }, "alreadySent" : fa
o.s.data.mongodb.core.MongoTemplate      : Saving Document containing fields: [_id, alreadySent, processId, payload, tim
o.s.data.mongodb.core.MongoTemplate      : Inserting Document containing fields: [alreadySent, processId, payload, times
c.u.i.e.c.service.CommunicationService   : Batch amount process with id 1 was sent successfully.
co.uis.iot.edge.core.vertx.VertxHandler  : sendMessage method called with message: {"gatewayId":null,"processId":1,"payl
u.v.s.process.SampleProcessVerticle      : [sendMessage] Successfuly Response: "The message was processed successfully"
o.s.data.mongodb.core.MongoTemplate      : find using query: { "type" : "GATEWAY" } fields: Document{{}} for class: clas
o.s.data.mongodb.core.MongoTemplate      : find using query: { "processId" : { "$numberLong" : "1" }, "alreadySent" : fa
o.s.data.mongodb.core.MongoTemplate      : find using query: { "idBackend" : { "$numberLong" : "1" } } fields: Document{
o.s.data.mongodb.core.MongoTemplate      : Inserting Document containing fields: [alreadySent, processId, payload, times
co.uis.iot.edge.core.vertx.VertxHandler  : saveMessage method called with message: {"gatewayId":null,"processId":1,"payl
u.v.s.process.SampleProcessVerticle      : [sendMessage] Successfuly Response: "The message was processed successfully"
```

Figura 36. Envío de mensajes desde el proceso

El proceso envió 3 mensajes. Cuando envió el segundo, la lógica del framework detecto que se cumplió la cantidad de mensajes establecidos en la propiedad *batch_amount* por ende procedió a enviar los dos mensajes. Por otro lado, con el tercer mensaje lo único que hizo fue guardarlo en base de datos puesto que el proceso especifica que se deben enviar mensajes de dos en dos. La

siguiente imagen de la base de datos muestra a los dos primeros mensajes enviados y el tercero no enviado.

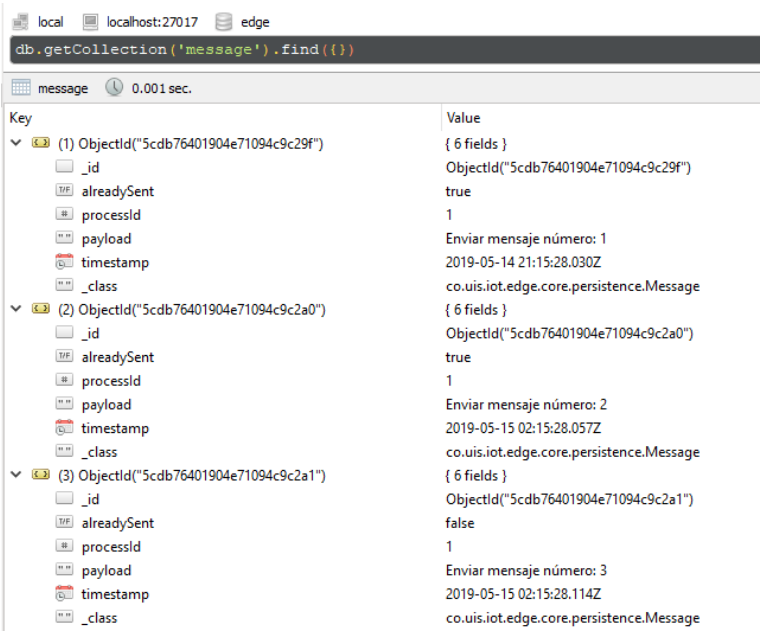

Figura 37. Estado de los mensajes en base de datos

Por último, se muestran los mensajes ya entregados en el broker en la siguiente figura:

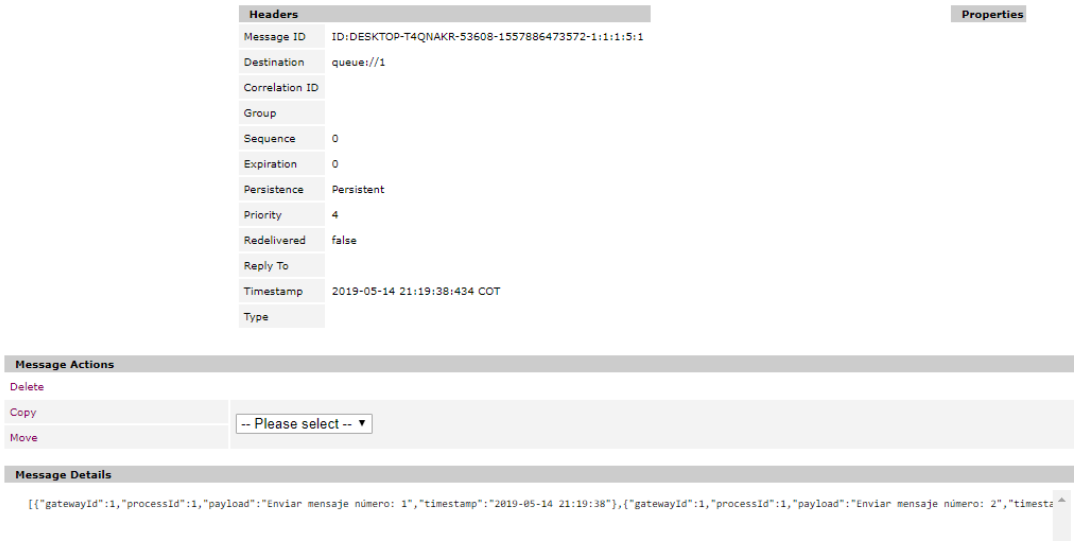

Figura 38. Mensajes recibidos por el broker

6.5.5 Guardado y consulta de mensajes mediante el Event Bus. En esta prueba se validó el requisito funcional RF06 - Persistencia local de mensajes emitidos por procesos.

El estado inicial de esta prueba es justo después de que el proceso enviara los 3 mensajes de la anterior prueba.

Para esta validación el proceso utilizó el tópico *saveMessage* al cual está subscrito el framework y envió el siguiente mensaje:

{"gatewayId":null,"processId":1,"payload":"Mensaje de prueba guardado desde proceso 1 a través del Event Bus.","timestamp":null}

Acto seguido el framework tomó el mensaje, lo insertó en la base de datos con estado ya enviado y le respondió al proceso informando que el guardado fue exitoso, a continuación, se muestra la base de datos con el mensaje:

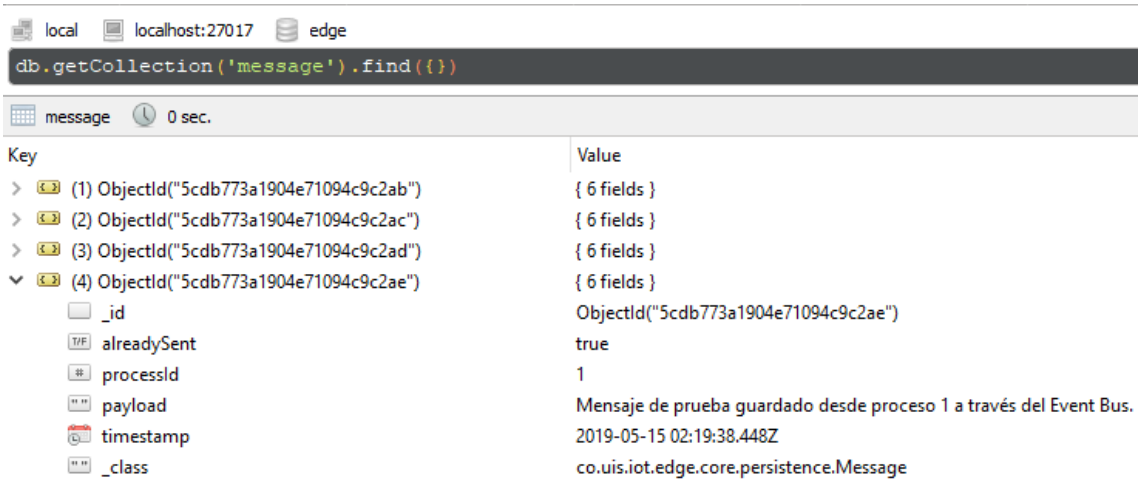

Figura 39. Mensaje almacenado a través del Event Bus

La siguiente figura muestra la ejecución de los eventos en la consola:

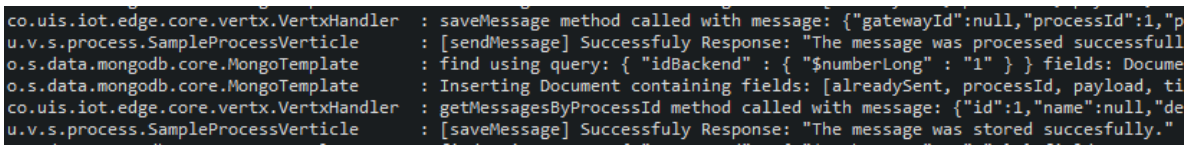

Figura 40. Ejecución de guardado de mensaje mediante el Event Bus

Consulta de mensajes

El proceso hizo una consulta de los mensajes pertenecientes al proceso con id 1, a través del tópico *getMessagesByProcessId* y el framework le respondió lo siguiente:

[{"gatewayId":null,"processId":1,"payload":"Enviar mensaje número:1",
"timestamp":"2019-05-14 16:19:38"},
{"gatewayId":null,"processId":1,"payload":"Enviar mensaje número:2",
"timestamp":"2019-05-14 21:19:38"},
{"gatewayId":null,"processId":1,"payload":"Enviar mensaje número:3",
"timestamp":"2019-05-14 21:19:38"},
{"gatewayId":null,"processId":1,"payload":"Mensaje de prueba guardado desde proceso 1 a través del Event Bus.","timestamp":"2019-05-14 21:19:38"}]

Lo cual representa la lista con los cuatro mensajes almacenados en la base de datos. A continuación, se muestra la ejecución de este procedimiento en la consola:

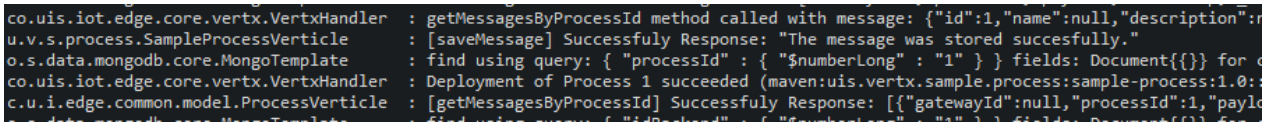

Figura 41. Ejecución de consulta de mensajes a través del Event Bus

6.5.6 Envío de un mensaje hacia un proceso. En esta prueba se validó el requisito funcional RF04 - Servicio web para enviar mensajes a procesos.

Para esta prueba se tomó como punto de partida el estado de la base de datos que quedó después de realizar las pruebas anteriores y se hizo uso del proceso que ya estaba desplegado.

Para el envío del mensaje hacia el proceso 1, se utilizó el método *POST* en el endpoint *http://localhost:8080/communication/process/message*, a continuación, se encuentra el body de la solicitud:

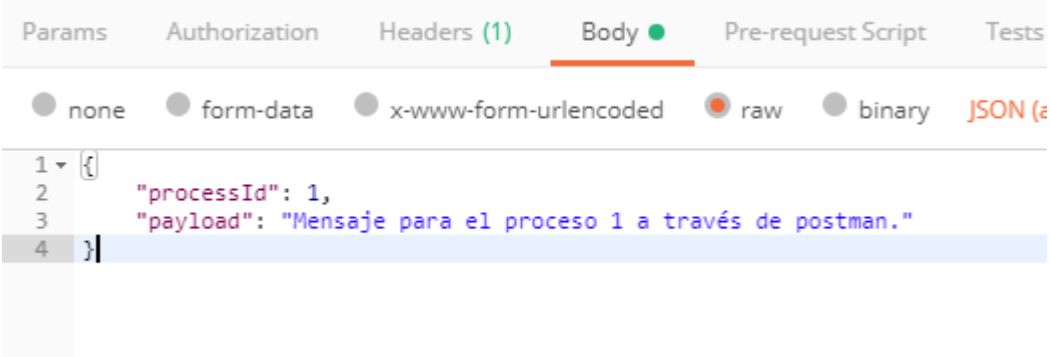

Figura 42. Solicitud para enviar mensaje a un proceso

El body muestra el proceso al cual se le va a enviar el mensaje y el *payload* que es el mensaje en sí. La respuesta de la solicitud es un body vació, pero con un *Status 200 ok*. Para validar que el mensaje fue entregado correctamente al proceso, a continuación, se muestra la consola en donde se ve la clase *SampleProcessVerticle* imprimiendo el mensaje que acaba de recibir:

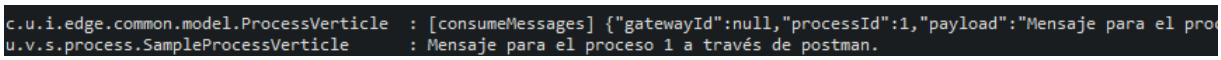

Figura 43. Ejecución del envío de un mensaje a un proceso

6.5.7 Consulta de estado de gateway y procesos. En esta prueba se validó el requisito funcional RF07 - Consulta del estado del gateway y los procesos.

Esta prueba se ejecutó con un solo proceso ejecutándose, sin embargo, el funcionamiento para más procesos es el mismo.

El servicio web fue llamado a través del método *GET* en el endpoint *http://localhost:8080/communication/keepAlive*, el body de la petición se envió vacío y la respuesta fue la siguiente:

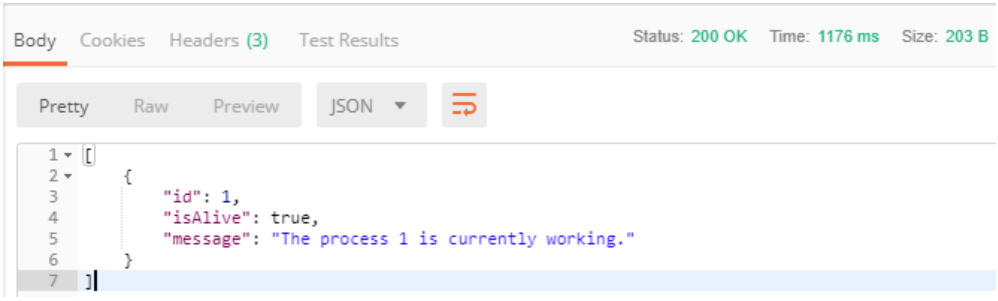

Figura 44. Respuesta del estado del gateway y los procesos

La respuesta muestra una lista de procesos con sólo un objeto el cual representa el proceso que se encontraba en ejecución. Este objeto indica que el proceso con *id* 1 está funcionando correctamente *isAlive: true* y un mensaje que re confirma su estado. Por otro lado, el estado del gateway se asume como activo, pues respondió la solicitud.

A continuación, se muestra la consola con la ejecución de este procedimiento:

```
o.s.data.mongodb.core.MongoTemplate      : find using query: { } fields: Document{{}} for class: class co.uis.iot.ed
c.u.i.edge.common.model.ProcessVerticle  : [keepAliveListener] KeepAlive received on process id: 1, procesando...
u.v.s.process.SampleProcessVerticle      : Verificando el estado del proceso...
```

Figura 45. Ejecución de la consulta de estado del gateway y los procesos

Como se observa en la anterior figura, el framework emitió una petición hacia el proceso, este la recibió y finalmente respondió después de ejecutar de su lógica.

6.5.8 Prueba de servicio web para gestión de mensajes. En esta prueba se validó el requisito funcional RF03 - Servicio web para gestión de mensajes.

El punto inicial para esta prueba fue justo después de desplegar el proceso, por lo tanto, habían 4 mensajes en la base de datos.

Consulta de mensajes:

Para hacer la prueba se hizo un llamado a través del método *GET* en el endpoint *http://localhost:8080/storage/messages?processId=1*, lo cual indica que se consultaron los mensajes asociados al proceso 1, a continuación, se muestra la respuesta a esta solicitud:

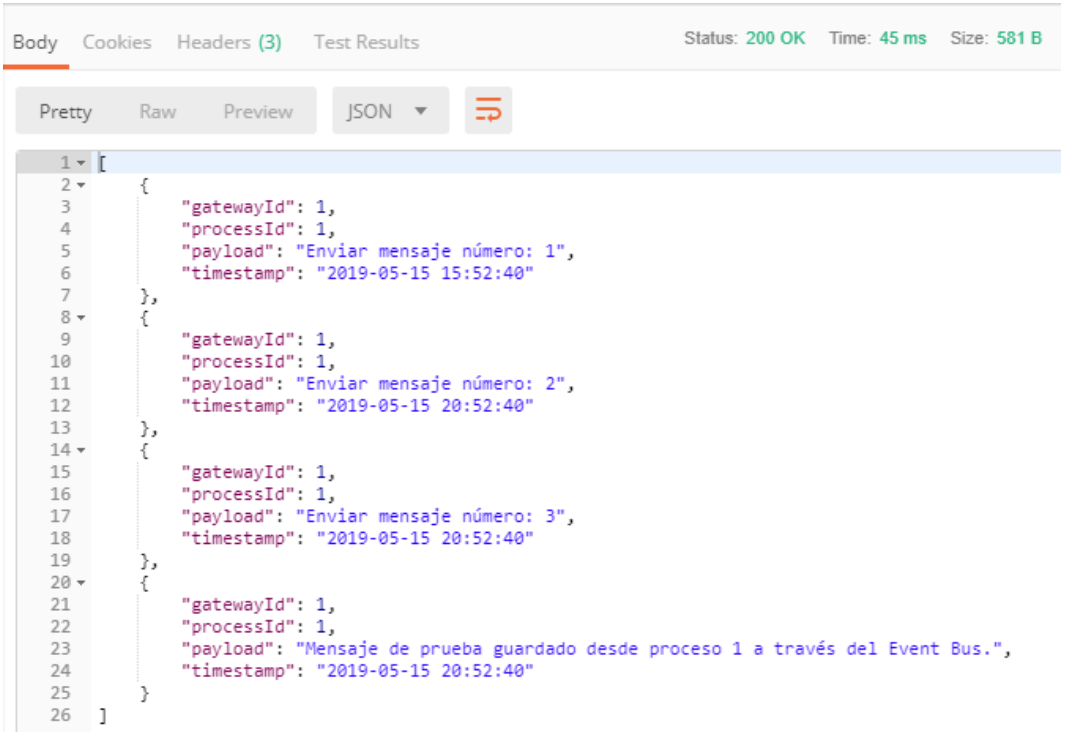

Figura 46. Respuesta de los mensajes de un proceso

La respuesta a esta petición muestra los cuatro mensajes almacenados en la base de datos.

Eliminación de mensajes:

Por último, se probó el servicio para eliminar mensajes a través del método *DELETE* en el endpoint *http://localhost:8080/storage/messages?processId=1&alreadySent=true*, en esta petición se solicitó eliminar los mensajes asociados al proceso 1 que ya fueron enviados *alreadySent=true*. La respuesta a esta solicitud es un *Status 200 ok*. Para verificar la correcta ejecución de esta prueba se muestra a continuación la base de datos:

Figura 47. Colección de mensajes después de eliminar

Como se muestra en la anterior figura el único mensaje que quedó disponible en la base de datos es aquel que no había sido previamente enviado.

6.6 Caso de uso: Implementación e implantación

En esta sección se realizó el despliegue de la plataforma IoT enfocada a Smart Campus, implementado un prototipo para demostrar que todos los componentes se integran correctamente. Para lograr esto, fue necesaria la participación de los autores de los proyectos de grado mencionados en los capítulos anteriores.

En el escenario planteado un supervisor desea activar una red de bombillos ubicados en dos laboratorios de ingeniería eléctrica cuando el voltaje de un generador eléctrico pase un umbral de seguridad para alertar al personal presente en dichos espacios.

Además de esto, el usuario desea monitorear la temperatura generada en otro laboratorio en el cual se almacenan algunas sustancias químicas que deben permanecer constantemente sobre una temperatura mínima. Para esto, instala un sensor de temperatura en el laboratorio, y un led en su oficina, el cual deberá encenderse en caso de que la temperatura en el laboratorio descienda bajo un umbral previamente determinado.

Para simular el escenario anterior se hizo uso de dos dispositivos tipo gateway, una minicomputadora con pines análogos, tres actuadores (LEDs), un sensor de temperatura y un potenciómetro; a su vez, se crearon 4 procesos y 1 aplicación externa en la que se muestran los datos capturados en el caso de uso.

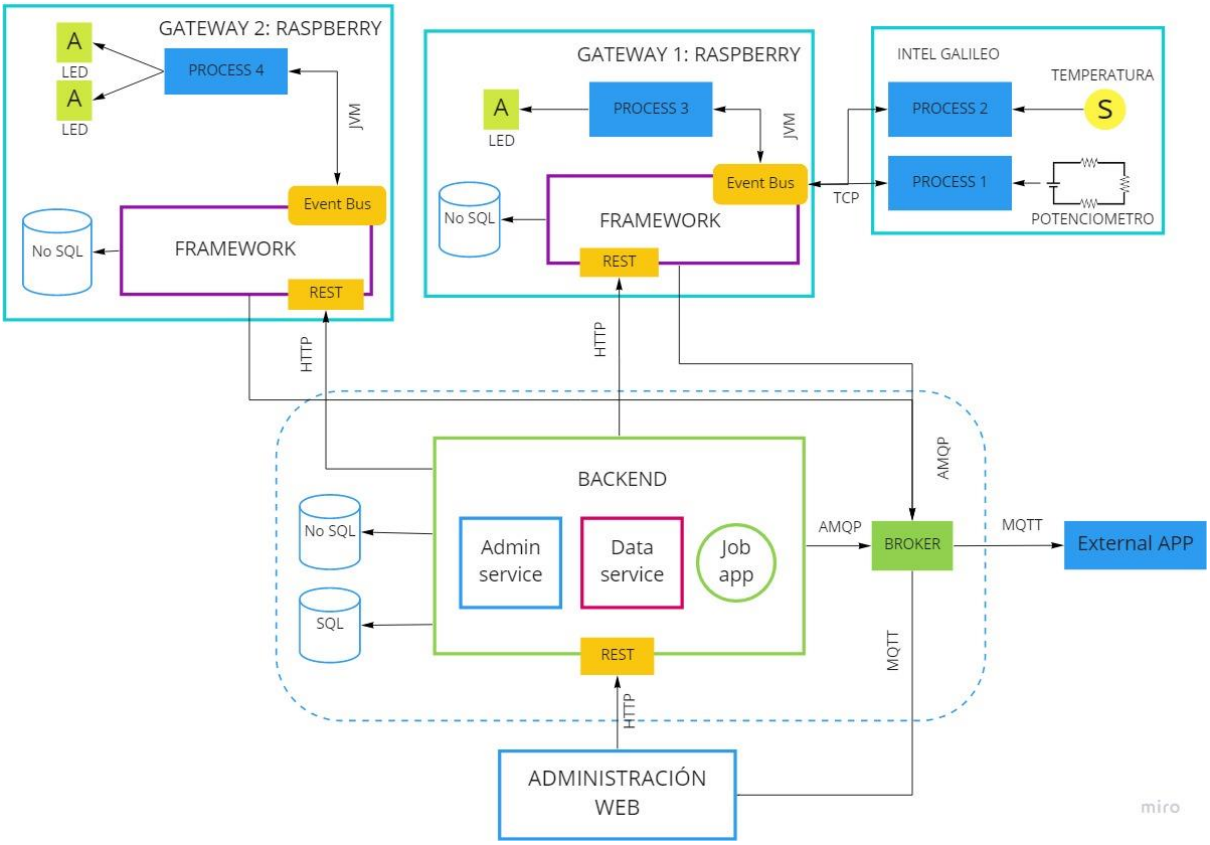

Figura 48. Arquitectura planteada para el caso de uso

Para no hacer extensa la lectura, se explicará lo realizado con respecto al framework gateway en aras de demostrar el valor que aporta este proyecto de grado con respecto al caso de uso y a la plataforma IoT. El requisito no funcional RNF02 - Portabilidad del framework, se validó en este capítulo al desplegar el framework gateway sobre las minicomputadoras de marca Raspberry Pi que tienen especificaciones, arquitectura y software muy diferentes a las del computador usado en el capítulo 6.5.

Para empezar a hablar de lo relacionado con los gateways, a continuación, se describe el hardware utilizado en este caso de uso:

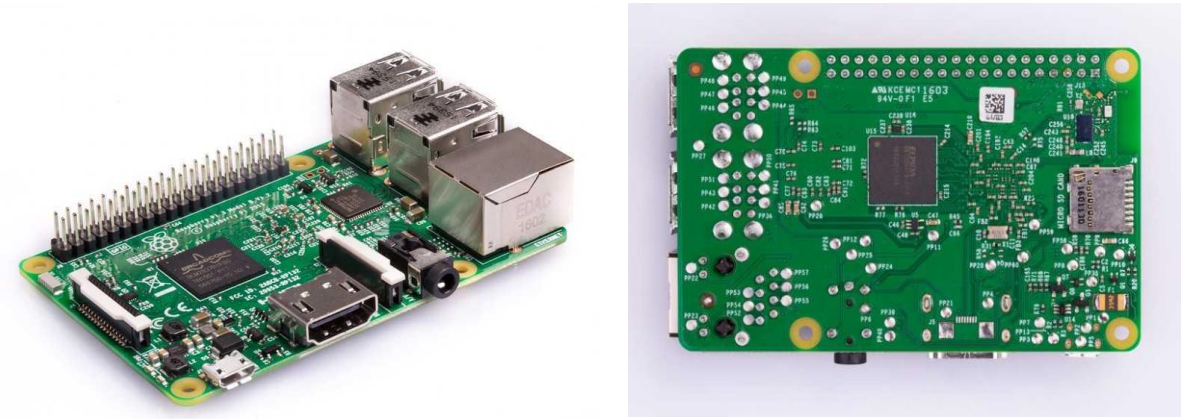

Figura 49. Raspberry Pi 3 Model B. (Raspberry Pi, s.f)

Tabla 6.
Especificaciones Raspberry Pi 3 Model B

Tipo	**Referencia**
Procesador	Quad Core 1.2GHz Broadcom
Arquitectura del procesador	64bit CPU
Memoria RAM	1GB
Almacenamiento	16 GB Micro SD

Figura 50. Placa Intel Galileo (Arduino, s.f)

Tabla 7.
Especificaciones Intel Galileo

Tipo	Referencia
Procesador	Intel® Quark™ SoC X1000 (16K Cache, 400 MHz)
Arquitectura del procesador	32-bit
Memoria RAM	256 MB DDR3
Almacenamiento	16 GB Micro SD

Tabla 8.
Dispositivos finales

Imagen	Tipo	Descripción
 *Figura 51.* Sensor de temperatura	Sensor	LM35: Mide la temperatura del ambiente. Su rango de medición abarca desde -55 °C hasta 150 °C (Ecured, s.f)
 *Figura 52.* Sensor Led	Actuador	LED: Emite luz al recibir una determinada señal eléctrica.

 *Figura 53.* Potenciómetro	Otro	Potenciómetro: Es una resistencia variable que permite modificar el voltaje de un circuito.

A nivel lógico se utilizaron los siguientes cuatro procesos:

- Proceso 1: Este proceso se encargó de manejar los datos generados en el circuito conectado a la placa Intel Galileo. Fue desarrollado en lenguaje Python y se conectó al Event Bus del Gateway 1 a través de TCP. El requisito no funcional RNF03 - Soporte de múltiples lenguajes en el framework fue validado con este proceso.
- Proceso 2: Este proceso se encargó de manejar los datos generados por el sensor de temperatura. Fue desarrollado en JAVA y se conectó al Event Bus del Gateway 1 a través de TCP.
- Proceso 3: Este proceso se encargó de dar instrucciones al LED ubicado en el Gateway 1 para prenderse o apagarse dependiendo de los mensajes que llegaban a través del Event Bus, pero en este caso enviados por el framework. Fue desarrollado en JAVA y desplegado sobre la misma JVM en donde estaba ejecutándose el framework.
    - Proceso 4: Este proceso se encargó de dar instrucciones a los LEDs ubicados en el Gateway 2 para prenderse o apagarse dependiendo de los mensajes que llegaban a través del Event Bus, en este caso enviados por el framework. Fue desarrollado en JAVA y desplegado sobre la misma JVM en donde estaba ejecutándose el framework. Para los

gateways bastó con ejecutar el framework y tomar su ip local puesto que toda la plataforma IoT se ejecutó dentro de la misma red.

En el Apéndice D se puede observar un vídeo explicativo del funcionamiento de este escenario de caso de uso sobre la plataforma IoT.

7. Trabajo a futuro

- Implementar un módulo Rule Engine que permita la toma de decisiones desde el framework gateway con respecto a los datos generados en los procesos mediante los sensores con el objetivo de agrupar funciones comúnmente usadas por los casos de uso de IoT para tomar decisiones en tiempo real.

8. Conclusiones

- El diseño e implementación de este framework demuestra que es viable la idea de convertir universidades tradicionales en universidades inteligentes (Smart campus) pues aquí se ve reflejado que es posible la construcción de casos de uso manipulando datos generados por sensores y actuadores a través de una arquitectura IoT de manera rápida.
- Una propiedad solicitada en la plataforma software a incluir en los gateways es la extensibilidad, que permita de forma adecuada agregar nuevo hardware (sensores y actuadores) y capacidades como, procesamiento, comunicación y persistencia para que el framework no quede obsoleto con el pasar del tiempo. Para soportar esta propiedad en el framework, se hizo uso de Vert.x el cual permite la creación de unidades de despliegues independientes (Verticles).
- Se implementó un API REST para permitir la administración del gateway y sus elementos asociados tales como sensores, actuadores y procesos, lo cual permite que los usuarios creen la representación lógica de los componentes del gateway, con la posibilidad de almacenar información útil (propiedades) para uso posterior.
- Se añadió una característica muy importante al framework como lo es la portabilidad, puesto que al ejecutarse sobre la JVM el desarrollo de este se convierte en multiplataforma, de manera que los usuarios no se ven en la obligación de usar un hardware especifico.
- Cabe destacar que el framework además es multilenguaje, esto quiere decir que si bien el código base está construido con JAVA el usuario puede implementar procesos (casos de

uso) en otros lenguajes sin ningún problema de compatibilidad, abriendo de esta manera la posibilidad de crear soluciones sin las limitaciones de un lenguaje especifico.

- Se proveyó la capacidad de enviar y recibir datos entre el framework y los procesos por medio del EventBus, bien sea a través de la JVM o mediante TCP para aquellos procesos que no se ejecutan sobre el mismo hardware del gateway o fueron construidos en otros lenguajes. Por consiguiente, se logró que procesos incluso desplegados en otra red puedan hacer uso del framework desplegado en cualquier gateway.
- Por último, se incluyó como función del framework la posibilidad de hacer despliegues dinámicos a través de un servicio web REST de procesos desarrollados en JAVA que estén empaquetados con Maven, de manera que estos procesos compartan la misma JVM del framework optimizando el uso de recursos.

Referencias Bibliográficas

Aprendiendoarduino. (2016) Sensores y Actuadores. (2016). Recuperado de https://aprendiendoarduino.wordpress.com/2016/12/18/sensores-y-actuadores/

Arduino. s.f. Arduino - IntelGalileo. Recuperado de https://www.arduino.cc/en/ArduinoCertified/IntelGalileo

Arias, K. y Estupiñan, F. (2019). Diseño del componente software backend orientado a una plataforma IoT diseñada para Smart Campus. Bucaramanga, Colombia.

Butler, B. (2017). What is edge computing and how it's changing the network. Recuperado de https://www.networkworld.com/article/3224893/what-is-edge-computing-and-how-it-s-changing-the-network.html

Camacho, D. (2019). Diseño de una aplicación Web extensible para la administración de una plataforma IoT diseñada para Smart Campus. Bucaramanga, Colombia.

Cuelogic. (2018). Top 3 Programming Languages for IoT Development In 2018 | IoT For All. Recuperado de https://www.iotforall.com/2018-top-3-programming-languages-iot-development/

del Valle Hernández, L. s.f. Aplicaciones del IoT usos prácticos en el mundo real. Recuperado de https://programarfacil.com/podcast/aplicaciones-del-iot-reales/

Eclipse Kura | The Eclipse Foundation. Recuperado de https://www.eclipse.org/kura/

LM35 - EcuRed. Recuperado de https://www.ecured.cu/LM35

Fremantle, P. (2015). A Reference Architecture for the Internet of Things. wso2.com. Recuperado de https://wso2.com/whitepapers/a-reference-architecture-for-the-internet-of-things/

Genbeta. (2014). MongoDB: qué es, cómo funciona y cuándo podemos usarlo (o no). (2014). Recuperado de https://www.genbeta.com/desarrollo/mongodb-que-es-como-funciona-y-cuando-podemos-usarlo-o-no

Malone, T. (2016). The IoT Journey: Connecting to the Cloud. Recuperado de https://blogs.msdn.microsoft.com/tedmalone/2016/05/30/the-iot-journey-connecting-to-the-cloud/

Masadelante. ¿Que es el TCP/IP? - Definición de TCP/IP. Disponible en http://www.masadelante.com/faqs/tcp-ip

McCauley, R. (2016). Using Alexa Skills Kit and AWS IoT to Voice Control Connected Devices. Recuperado de https://developer.amazon.com/de/blogs/post/Tx3828JHC7O9GZ9/Using-Alexa-Skills-Kit-and-AWS-IoT-to-Voice-Control-Connected-Devices

Montañez, F. (2017). Diseño de una plataforma software que apoye los procesos de crianza en el área de la piscicultura. Bucaramanga, Colombia.

Mozilla. HTTP. Recuperado de https://developer.mozilla.org/es/docs/Web/HTTP

NeoAttack ¿Qué es un Framework y para que sirve? - Neo Wiki. Recuperado de https://neoattack.com/neowiki/framework/

Node-RED. Recuperado de https://nodered.org/

Pastor, J. (2018). Edge Computing: qué es y por qué hay gente que piensa que es el futuro. Xataka.com. Recuperado de https://www.xataka.com/internet-of-things/edge-computing-que-es-y-por-que-hay-gente-que-piensa-que-es-el-futuro

Raspberry Pi. Raspberry Pi 3 Model B. Recuperado de https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

Rouse, M. (2018). What is Advanced Message Queuing Protocol (AMQP) ? - Definition from WhatIs.com. Recuperado de https://whatis.techtarget.com/definition/Advanced-Message-Queuing-Protocol-AMQP

Venemedia Comunicaciones C.A. Interfaz - Qué es y Definición. Recuperado de https://conceptodefinicion.de/interfaz/