



Código	FPI-009
Objeto	Guía de elaboración de Informe final de proyecto
Usuario	Director de proyecto de investigación
Autor	Secretaría de Ciencia y Tecnología de la UNLaM
Versión	5
Vigencia	03/9/2019

Departamento: Ingeniería e Investigaciones Tecnológicas

**Programa de acreditación:
CyTMA2**

**Programa de Investigación¹:
Código del Proyecto:
C2-ING-076**

**Título del proyecto
Interconexión de dispositivos IOT empleando MQTT y NodeJs en redes dual stack**

PIDC:

Elija un elemento.

PII:

Elija un elemento.

**Director:
Carlos Alberto Binker**

Director externo:

**Codirector:
Hugo Raúl Tantignone**

Integrantes: Guillermo Buranits, Diego Romero

Investigador Externo, Asesor- Especialista, Graduado UNLaM:

Alumnos de grado: (Aclarar si tiene Beca UNLaM/CIN)

Eliseo Zurdo, Maximiliano Frattini, Lautaro Lasorsa (Beca UNLaM)

Alumnos de posgrado:

Resolución Rectoral de acreditación: N° 456/21

Fecha de inicio: 01/01/2021

Fecha de finalización: 31/12/2022

¹ Los Programas de Investigación de la UNLaM están acreditados con resolución rectoral, según lo indica la Resolución HCS N° 014/15 sobre **Lineamientos generales para el establecimiento, desarrollo y gestión de Programas de Investigación a desarrollarse en la Universidad Nacional de La Matanza**. Consultar en el departamento académico correspondiente la inscripción del proyecto en un Programa acreditado.



Código	FPI-009
Objeto	Guía de elaboración de Informe final de proyecto
Usuario	Director de proyecto de investigación
Autor	Secretaría de Ciencia y Tecnología de la UNLaM
Versión	5
Vigencia	03/9/2019

A. Desarrollo del proyecto (adjuntar el protocolo)

A.1. Grado de ejecución de los objetivos inicialmente planteados, modificaciones o ampliaciones u obstáculos encontrados para su realización (desarrolle en no más de dos (2) páginas)

Descripción general del caso de estudio planteado

Teniendo en cuenta los objetivos planteados en el resumen del proyecto y el caso de estudio propuesto podemos decir que los logros obtenidos han sido altamente satisfactorios. Hemos construido un sistema en donde los dispositivos transmiten datos a través del protocolo mqtt. Estos datos son recepcionados en un cloud server en donde está montado un broker de uso profesional denominado *EMQX*, en donde nosotros nos valemos de una versión denominada *community* que es totalmente gratuita. Los datos recibidos en este broker mqtt son verificados por un motor de reglas para ver si satisfacen o no un determinado criterio. En caso de que la regla coincida (en la jerga se denomina *matcheo*), se envía mediante el método POST a una rutina de código (escrita en JavaScript) que devuelve mediante un tópico usando mqtt un mensaje hacia el dispositivo microcontrolador (recordemos que estamos operando con el ESP8266 y el ESP32) y el comando recepcionado hace que el ESP32 o en su defecto el ESP8266 ahora se comporten como actuadores provocando una determinada acción, como por ejemplo simular la apertura de una barrera, accionar un ventilador porque la temperatura ha superado un determinado umbral previamente configurado o bien apagar dicho ventilador cuando la temperatura desciende de un determinado umbral, etc. A su vez también todo esto está interactuando mediante un bot de Telegram, pudiéndose recibir mensajes a través de dicho bot en un celular por ejemplo.

Descripción específica del prototipo construido como caso de estudio

En primer lugar presentamos el diagrama en bloques de uno de los dispositivos (ver Figura 1 y 2, esta última con el detalle real del prototipo construido).

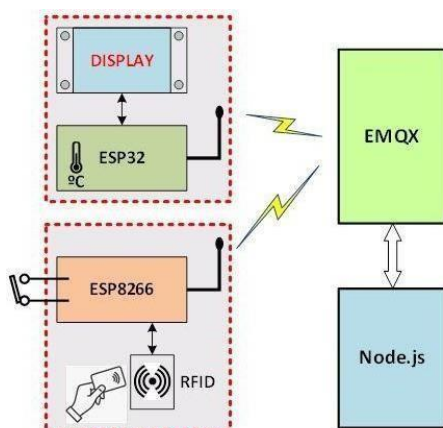


Figura 1 – Diagrama en bloques dispositivo RFID con apertura de barrera e indicación en display

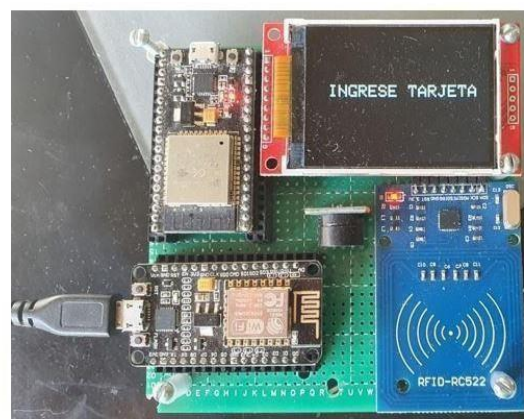


Figura 2 – Diagrama real del prototipo, el ESP32 comanda el display (actuador) y el ESP8266 el lector RFID (elemento sensor)



Código	FPI-009
Objeto	Guía de elaboración de Informe final de proyecto
Usuario	Director de proyecto de investigación
Autor	Secretaría de Ciencia y Tecnología de la UNLAM
Versión	5
Vigencia	03/9/2019

La regla que controla al dispositivo de la **Figura 2** se activa cuando en el payload del tópico se ponga en **1** la llave “enable” enviada en el objeto JSON. El payload que emite el ESP8266 es el siguiente:

```
{“card”: “83475004”, “enable”:1}
```

Por otro lado, los tópicos de emisión para el dispositivo en cuestión son los siguientes: hay un solo tópico para emitir el CardID de la tarjeta RFID en el ESP8266, el tópico es el siguiente: *userId/deviceId/card/sensordata*, (el payload ya fue explicado con anterioridad), mientras que para el ESP32 los tópicos son: *userId/deviceId/temp/sensordata* para emitir la temperatura interna del microcontrolador, mientras que los tópicos de recepción en esta ocasión serán: *userId/deviceId/username/actdata* (para mostrar el nombre del usuario al que se le dio acceso con la tarjeta RFID exitosamente).



Figura 3 - Dispositivo que simula un electroventilador en un rango de temperatura ($85\text{ }^{\circ}\text{C} < \text{temp} < 95\text{ }^{\circ}\text{C}$)

Por otro lado, el dispositivo de la **Figura 3**, corresponde a un ESP32, que obra como un dispositivo que simula un electroventilador en un rango de temperatura ($85\text{ }^{\circ}\text{C} < \text{temp} < 95\text{ }^{\circ}\text{C}$). El tópico de emisión utilizado en este caso y que emplearemos en este segundo dispositivo será: *userId/deviceId/tempESP32/sensordata*; y los tópicos de recepción empleados que permitirán el encendido o el apagado del ventilador serán: *userId/deviceId/command/actdata*. En este caso el payload a pasarle al ESP32 será simplemente el string “fan_on” para activar el ventilador o “fan_off” para apagarlo.

Los servicios a activar fueron dos contenedores, uno con *Mongo* (se empleó la versión 4.4) y el otro con *EMQX* (versión 4.2.2). El *node JS* empleado fue la versión 14.20 LTS y para el gestor de paquetes *NPM* se utilizó la versión 8.1.8. Para levantar los contenedores se empleó *docker-compose* y se configuró un archivo de configuración denominado *docker-compose.yml*. Todos los servicios han sido probados en sistema operativo Linux server versión 20.04. Para dejar activo permanentemente como un servicio al bot de telegram en nuestro cloudserver se ha utilizado *pm2*. Para acceder a mongodb se ha utilizado el cliente *mongodb-compass*. Para la realización de las pruebas de la API se ha utilizado *Postman* en su versión de escritorio.

Todo el detalle de la investigación se aporta más adelante en la publicación que nos fue aceptada en CONAISI 2022.



Código	FPI-009
Objeto	Guía de elaboración de Informe final de proyecto
Usuario	Director de proyecto de investigación
Autor	Secretaría de Ciencia y Tecnología de la UNLaM
Versión	5
Vigencia	03/9/2019

B. Principales resultados de la investigación

B.1. Publicaciones en revistas (informar cada producción por separado)

Artículo 1:	
Autores	
Título del artículo	
N° de fascículo	
N° de Volumen	
Revista	
Año	
Institución editora de la revista	
País de procedencia de institución editora	
Arbitraje	Elija un elemento.
ISSN:	
URL de descarga del artículo	
N° DOI	

B.2. Libros

Libro 1	
Autores	
Título del Libro	
Año	
Editorial	
Lugar de impresión	
Arbitraje	Elija un elemento.
ISBN:	
URL de descarga del libro	
N° DOI	



Código	FPI-009
Objeto	Guía de elaboración de Informe final de proyecto
Usuario	Director de proyecto de investigación
Autor	Secretaría de Ciencia y Tecnología de la UNLaM
Versión	5
Vigencia	03/9/2019

B.3. Capítulos de libros

Autores	
Título del Capítulo	
Título del Libro	
Año	
Editores del libro/Compiladores	
Lugar de impresión	
Arbitraje	Elija un elemento.
ISBN:	
URL de descarga del capítulo	
N° DOI	

B.4. Trabajos presentados a congresos y/o seminarios

CONAIIISI 2022 – UTN Facultad regional Concepción del Uruguay	
Autores	<i>Carlos Binker, Hugo Tantignone, Eliseo Zurdo, Guillermo Buranits, Lautaro Lasorsa</i>
Título	<i>Sistema de notificación de alarmas mediante Telegram y Bots utilizando webhooks con motor de reglas en EMQX empleando tecnología NodeJS y Docker para dispositivos IOT</i>
Año	2022
Evento	CONAIIISI 2022
Lugar de realización	<i>UTN Facultad regional Concepción del Uruguay</i>
Fecha de presentación de la ponencia	03/11/2022
Entidad que organiza	<i>RedUNCI y UTN-FRCU</i>
URL de descarga del trabajo (especificar solo si es la descarga del trabajo; formatos pdf, e-pub, etc.)	https://rtyc.utn.edu.ar/index.php/ajea/article/view/1146/1059 (Ver Pág. 867). Se adjunta el artículo en el ANEXO I.

B.5. Otras publicaciones

Autores	
Año	
Título	
Medio de Publicación	



Código	FPI-009
Objeto	Guía de elaboración de Informe final de proyecto
Usuario	Director de proyecto de investigación
Autor	Secretaría de Ciencia y Tecnología de la UNLaM
Versión	5
Vigencia	03/9/2019

C. Otros resultados. Indicar aquellos resultados pasibles de ser protegidos a través de instrumentos de propiedad intelectual, como patentes, derechos de autor, derechos de obtentor, etc. y desarrollos que no pueden ser protegidos por instrumentos de propiedad intelectual, como las tecnologías organizacionales y otros. Complete un cuadro por cada uno de estos dos tipos de productos.

C.1. Títulos de propiedad intelectual. Indicar: Tipo (marcas, patentes, modelos y diseños, la transferencia tecnológica) de desarrollo o producto, Titular, Fecha de solicitud, Fecha de otorgamiento

Tipo	Titular	Fecha de Solicitud	Fecha de Emisión

C.2. Otros desarrollos no pasibles de ser protegidos por títulos de propiedad intelectual. Indicar: Producto y Descripción.

Producto	Descripción

D. Formación de recursos humanos. Trabajos finales de graduación, tesis de grado y posgrado. Completar un cuadro por cada uno de los trabajos generados en el marco del proyecto.

D.1. Tesis de grado

Director (apellido y nombre)	Autor (apellido y nombre)	Institución	Calificación	Fecha /En curso	Título de la tesis

D.2 Trabajo Final de Especialización

Director (apellido y nombre)	Autor (apellido y nombre)	Institución	Calificación	Fecha /En curso	Título del Trabajo Final

D.2. Tesis de posgrado: Maestría

Director (apellido y nombre)	Tesista (apellido y nombre)	Institución	Calificación	Fecha /En curso	Título de la tesis



Código	FPI-009
Objeto	Guía de elaboración de Informe final de proyecto
Usuario	Director de proyecto de investigación
Autor	Secretaría de Ciencia y Tecnología de la UNLAM
Versión	5
Vigencia	03/9/2019

D.3. Tesis de posgrado: Doctorado

Director (apellido y nombre)	Tesista (apellido y nombre)	Institución	Calificación	Fecha /En curso	Título de la tesis

D.4. Trabajos de Posdoctorado

Director (apellido y nombre)	Posdoctorado (apellido y nombre)	Institución	Calificación	Fecha /En curso	Título del trabajo	Publicación

E. Otros recursos humanos en formación: estudiantes/ investigadores (grado/posgrado/ posdoctorado)

Apellido y nombre del Recurso Humano	Tipo	Institución	Período (desde/hasta)	Actividad asignada ²
Eliseo Zurdo	Alumno	UNLAM	01/01/2021 - 31/12/2022	Ver FPI-013
Maximiliano Frattini	Alumno	UNLAM	01/01/2021 - 31/12/2022	Ver FPI-013
Lautaro Lasorsa	Alumno	UNLAM	01/01/2021 - 31/12/2022	Ver FPI-013

F. Vinculación³: Indicar conformación de redes, intercambio científico, etc. con otros grupos de investigación; con el ámbito productivo o con entidades públicas. Desarrolle en no más de dos (2) páginas.

G. Otra información. Incluir toda otra información que se considere pertinente.

--

² Descripción de la/s actividad/es a cargo (máximo 30 palabras)

³ Entendemos por acciones de “vinculación” aquellas que tienen por objetivo dar respuesta a problemas, generando la creación de productos o servicios innovadores y confeccionados “a medida” de sus contrapartes.



Código	FPI-009
Objeto	Guía de elaboración de Informe final de proyecto
Usuario	Director de proyecto de investigación
Autor	Secretaría de Ciencia y Tecnología de la UNLaM
Versión	5
Vigencia	03/9/2019

H. Cuerpo de anexos:

- Anexo I: Copia de cada uno de los trabajos mencionados en los puntos B, C y D, y certificaciones cuando corresponda.⁴
- Anexo II:
 - FPI-013: Evaluación de alumnos integrantes. (si corresponde)
 - FPI-014: Comprobante de liquidación y rendición de viáticos. (si corresponde)
 - FPI-015: Rendición de gastos del proyecto de investigación acompañado de las hojas foliadas con los comprobantes de gastos.
 - FPI-038: Formulario de reasignación de fondos en Presupuesto.
- Anexo III: Alta patrimonial de los bienes adquiridos con presupuesto del proyecto (FPI 017)
- Nota justificando baja de integrantes del equipo de investigación.

Carlos Alberto Binker

Lugar y fecha: San Justo, 17/2/2023

- Presentar una copia impresa firmada del presente documento junto con los Anexos, y enviar todo en archivo PDF por correo electrónico a la Secretaría de Investigación Departamental. **Límite de entrega: 28 de febrero de 2023**

⁴ En caso de libros, podrá presentarse una fotocopia de la primera hoja significativa o su equivalente y el índice.



Código	FPI-009
Objeto	Guía de elaboración de Informe final de proyecto
Usuario	Director de proyecto de investigación
Autor	Secretaría de Ciencia y Tecnología de la UNLaM
Versión	5
Vigencia	03/9/2019

ANEXO I

Sistema de notificación de alarmas mediante Telegram y Bots utilizando webhooks con motor de reglas en EMQX empleando tecnología NodeJS y Docker para dispositivos IOT

Carlos Alberto Binker^{1,2}, Hugo Tantignone^{1,2}, Eliseo Zurdo^{1,2}, Guillermo Buranits^{1,2} Lautaro Lasorsa^{1,2}

¹Departamento de Ingeniería e Investigaciones Tecnológicas

²Universidad Nacional de La Matanza, Florencio Varela 1903 (B1754JEC) -- San Justo, Provincia de Buenos Aires, Argentina

{cbinker, htantignone, eazurdo gburanits}@unlam.edu.ar; {llasorsa}@alumno.unlam.edu.ar

Resumen

Cada vez que abordamos la necesidad de trabajar con dispositivos IOT se nos presenta un enorme desafío en determinar qué hacer con los variables capturadas por los sensores que se conectan a estos elementos (ARDUINO 1, ESP8266, ESP32, etc.). Normalmente se suele ver un único aspecto (que no es menor), que constituye el mero hardware, la electrónica propiamente dicha, pero queda instalada en otro plano totalmente desvinculada la parte de comunicaciones (concretamente el envío de esos datos a un servidor para su tratamiento), es decir qué hacer con estos datos generados por los sensores que se conectan a dichos dispositivos o entre otras cosas cómo gobernar los actuadores que operan también en estos dispositivos, por ejemplo de manera remota, etc. En este sentido existen numerosas plataformas que se encargan de simplificarnos la vida tales como Ubidot, Blynk, Graffana, Cloudmqtt, etc. Todos estos sistemas nos exigen crearnos una cuenta de usuario para administrar nuestros dispositivos IOT. Por lo tanto, el enfoque que se plantea en este artículo, y aquí viene nuestro VALOR AGREGADO, el cual consiste en diseñar nosotros nuestro sistema de gestión de dispositivos para crear soluciones a medida y no depender de plataformas de terceros. En ese sentido, el núcleo de nuestro proyecto estará basado en el poderoso broker mqtt denominado EMQX, que es una potente solución que mediante su versión community permite gestionar hasta 100K dispositivos.

1. Introducción

Se presentará un caso de estudio en donde se ensayará un sistema de gestión de dispositivos IOT, empleando componentes tales como el ESP32, ESP8266 [1-2], un sistema de lector de tarjetas RFID MC522, display SPI, etc. Este sistema está conformado por una API [3] escrita totalmente en JavaScript que le permitirá a un usuario registrar sus propios dispositivos (los podrá crear, listar o eliminar). A su vez los usuarios podrán autenticarse una única vez y mantener su sesión empleando JWT (JSON Web Token) [4]. Esta API empleará una base de datos no relacional tal como MongoDB [5]. El foco del estudio radica en la generación de reglas de usuario por medio del potente motor de reglas que incorpora EMQX [6]. Estas reglas están manejadas por webhooks [7]. Un webhook es

un handler http (conocido también como un endpoint) y constituye el recurso asociado a una o más reglas aplicadas a un determinado dispositivo. Existen otras variantes de recursos que atiendan reglas, pero cabe destacar que ésta es una de las técnicas más utilizadas y en el caso de este potente broker son de uso gratuito. Las notificaciones generadas cuando se cumpla la condición de la regla, como otro valor agregado más a nuestro estudio, permite el envío de mensajes empleando Telegram a través de bots[8-9]. Todos los servicios están implementados sobre contenedores utilizando Docker [10], lo cual brinda seguridad y escalabilidad al sistema.

2. Descripción de la tecnología de mensajería Telegram y del SOC ESP32

NOTA: si bien en uno de los dispositivos que empleamos utilizamos un ESP8266, sólo brindaremos una breve explicación del SOC ESP32, ya que constituye un dispositivo más evolucionado, de hecho, es el sucesor del ESP8266. Por ende, su explicación en cierta manera cubre aspectos de funcionamiento inherentes también al ESP8266.

2.1 Esquema de la plataforma de mensajería instantánea Telegram

Telegram está creado sobre una serie de servidores distribuidos. Dichos servidores, para comunicarse entre sí, utilizan un protocolo propio llamado MTProto. La razón del uso de este protocolo propietario es la mejora en seguridad como también el envío de mensajes, sobre todo vídeo e imagen. Haciendo un poco de historia podemos ver que hay dos versiones de MTProto. La primera versión se utilizó en 2014. Los mensajes en MTProto eran cifrados con el

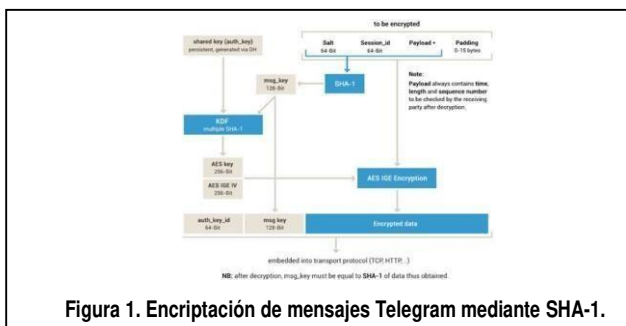


Figura 1. Encriptación de mensajes Telegram mediante SHA-1.

algoritmo SHA-1. Por un reporte en enero de 2015, en donde el investigador Juliano Rizzo reveló un error en el funcionamiento de SHA-1 que originó una vulnerabilidad al interceptar los mensajes, en 2016 se señaló un posible reemplazo del SHA-1 por el SHA-2. Por lo tanto, en 2017, se lanza la segunda versión. El cifrado se reemplazó a SHA-256 con mayor cantidad de bytes de carga útil. Para complementar lo dicho observar la Figura 1 y 1 bis.

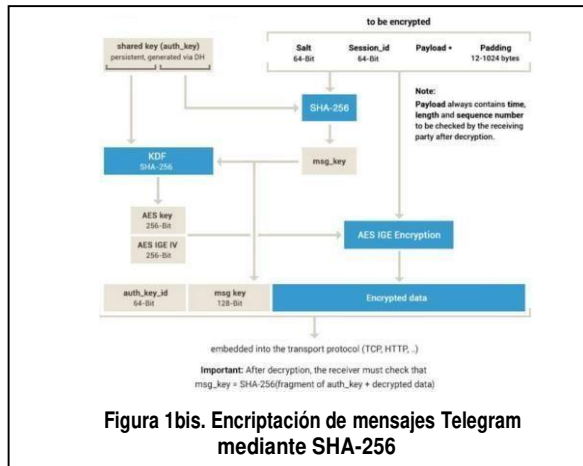


Figura 1 bis. Encriptación de mensajes Telegram mediante SHA-256

2.2 API de Telegram asociada para el manejo de bots

La API de Telegram permite la interacción de los bots con un sistema. En nuestro caso ese sistema será un hardware a controlar remotamente a través de una interfaz de usuario, que es en concreto el bot. Los bots de Telegram son cuentas especiales que no están ligadas a un número de teléfono. Al igual que la cuenta de un usuario, el acceso a la misma es por medio del *Alias*, que se accede anteponiendo @ al nombre del bot, o bien se accede por el propio nombre. Estas cuentas sirven como una interface para albergar código que puede estar ejecutándose en cualquier lugar del mundo en un servidor. Su uso es totalmente transparente, ya que los usuarios no necesitan conocer nada absolutamente sobre como el protocolo de encriptación MTProto funciona. Los servidores de Telegram manejarán todo lo referente a la encriptación de los mensajes y la comunicación con la API de una manera muy sencilla. Esta comunicación con los servidores de Telegram a través de la API se da vía una simple conexión https. Todas las consultas a la API de Telegram Bot deberán realizarse de esta manera:

https://api.telegram.org/bot<token>/METHOD_NAME

Se creará un bot que en esta ocasión sólo recibirá notificaciones provenientes del broker mqtt (EMQX). Dicho bot se denomina Conaiisi_2022. A título de ejemplo con el token suministrado para Conaiisi_2022, la forma de acceder desde la web sería:

<https://api.telegram.org/bot5643275066:AAEEfXKM7vfNoV6ueehjy9FvKWw3mtCPEu0/getme>

La API soporta los métodos http GET y POST. Ante una petición (request) hacia el bot, la respuesta de la API es un objeto JSON (ver Figura 2), la misma fue capturada

empleando Postman con el método GET, cabe aclarar que con POST se obtuvo el mismo resultado. Tal como se observa, la API siempre devuelve un campo de tipo Boolean denominado "ok" y otro campo denominado "result", en donde puede tener un campo opcional String denominado "description" con una descripción del resultado. Si la petición hacia el bot fue satisfactoria, el campo "ok" recibido en el objeto JSON es igual a true y si además el resultado de la consulta fue localizado se nos devolverá el campo "result" con todos los resultados, tal como se indica en el ejemplo de la Figura 2. En cambio, si la respuesta del campo "ok" es igual a false, se devolverá un código de error, tal como el siguiente ejemplo:

```
{ "ok":
false, "error_code":401, "description": "Unauthorized" }
```

En la Figura 3 se puede observar claramente la interacción entre todos los componentes, es decir entre el cliente (el usuario propietario del bot), el bot y la API de Telegram. En nuestro caso la lógica de control del bot estará implementada en un cloudserver (utilizaremos como proveedor a Donweb, siendo nuestra IP pública 168.181.187.81 o bien el dominio c2ing076.ml, obtenido a través de frenom. El evento que controlará al bot será generado en nuestro caso desde la plataforma EMQX. Las capturas de los resultados obtenidos corresponden al development y por lo tanto están realizadas bajo el dominio localhost.



Figura 2. Captura en respuesta a la petición del método GET

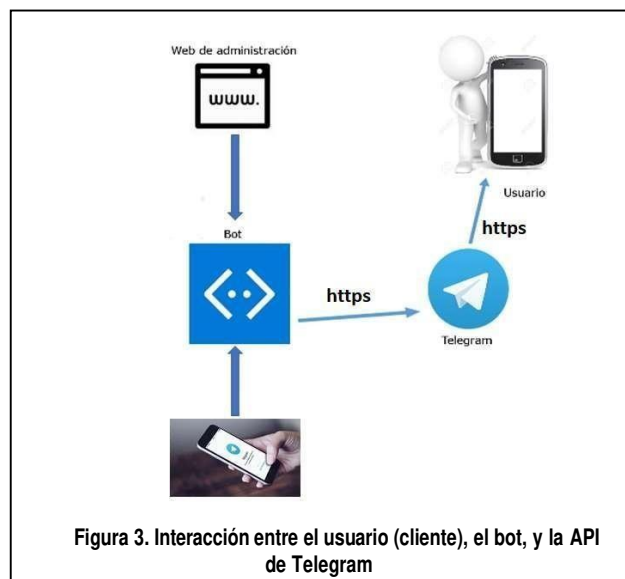
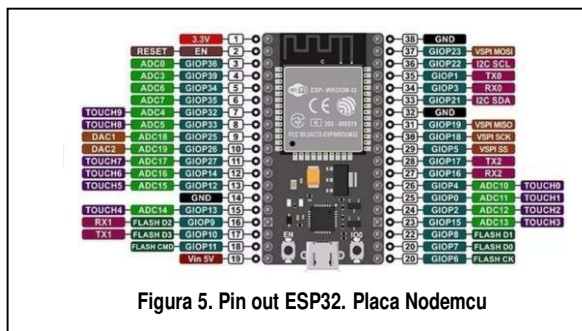
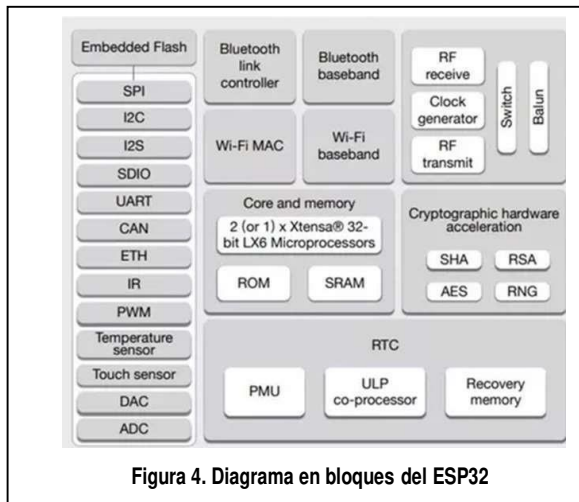


Figura 3. Interacción entre el usuario (cliente), el bot, y la API de Telegram

2.3 Descripción del SOC ESP32

Si bien en uno de los dispositivos del caso de estudio que estamos presentando empleamos un ESP8266, sólo daremos una breve síntesis de su sucesor (el ESP32), ya que es un dispositivo de más jerarquía y con mayores prestaciones. Para más detalle acerca del ESP8266 se puede consultar la página del fabricante: <https://espressif.com/>

Volviendo al ESP32, este dispositivo es un SOC (System on a chip) que está adquiriendo una enorme popularidad debido a su bajo costo y la implementación de diferentes entornos de programación, tales como LUA y Python. Pero su enorme potencial radica en el hecho de que soporta también el IDE de Arduino y trabajar con él es como si realmente estuviéramos trabajando sobre alguna de las placas Arduino, que como ya sabemos goza de una enorme comunidad abocada al desarrollo de código y de librerías. Todo esto puede ser utilizado en forma transparente para este dispositivo y así evitamos la utilización de un doble microcontrolador. Este dispositivo se destaca por sobre todo porque ya resuelve todo lo concerniente a la comunicación

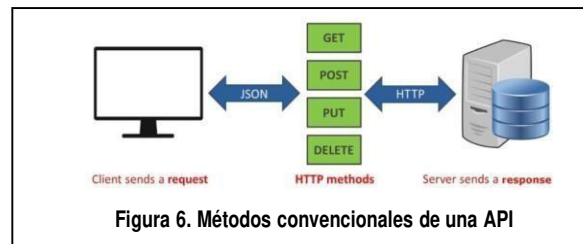


de dispositivos a través de redes, ya que soporta WIFI, Bluetooth, Hardware criptográfico, SPI, I2C, I2S, Ethernet, etc. Además, incorpora un sensor que mide la temperatura interna del dispositivo y sensores Touch capacitivos y de efecto Hall, además de conversores DAC y ADC. Posee un microprocesador con dos núcleos de 32 bits, lo que lo hace ideal para procesamiento paralelo. Cualquier pin del ESP32 puede configurarse para el control de interrupciones. Para

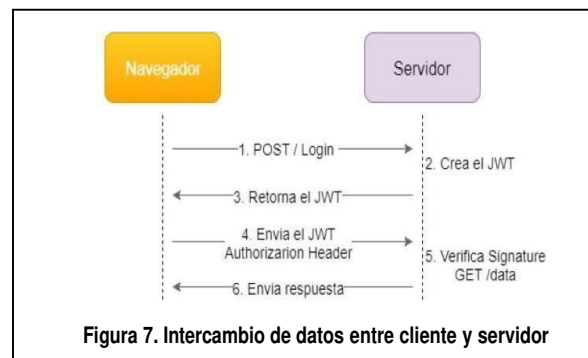
más detalle se expone el diagrama en bloques para dar una idea general de su potencialidad, como así también el correspondiente PIN-OUT, (ver Figura 4 y Figura 5 respectivamente).

3 Concepto de API REST

De acuerdo a las siglas API significa Application Programming Interface, es decir una interface de programación de aplicaciones. Observemos la Figura 6. Según el diagrama de la figura, los clientes envían



peticiones a un servidor mediante el protocolo http. Para ello se utilizan cuatro métodos (GET, POST, PUT y DELETE). Estos métodos permiten cumplir con el estándar CRUD cuyas siglas significan CREATE, READ, UPDATE y DELETE. Es decir (y llevándolo al caso concreto de nuestro estudio) con estos métodos se podrán crear dispositivos (método POST), leer un listado de dispositivos (método GET), actualizar dispositivos (método UPDATE) o bien borrar los mismos mediante el método DELETE. Todas estas peticiones se realizan bajo un modelo de datos denominado JSON, que en concreto son strings que están vinculados a los objetos del tipo JavaScript. Los JSON al ser strings constituyen un método estándar que permiten vincular diferentes aplicaciones entre sí, al igual que por ejemplo XML. Por otro lado como parte del backend la API



también involucra a una base de datos. Suele emplearse MongoDB muchísimo y cada vez más en aplicaciones del tipo IOT. MongoDB tiene la ventaja que, por tratarse de una base de datos no relacional, trata a los datos como objetos JavaScript, permitiendo una mejor experiencia de programación al usar precisamente a JavaScript como lenguaje de alto nivel. Para concluir para simular el servidor web en el desarrollo se utilizará *express* que permite montar una API bajo el entorno de ejecución de *NodeJs*.

5. Descripción de la tecnología mqtt. Plataforma EMQX

5.1 Descripción general

El broker mqtt es el responsable de la administración de los mensajes provenientes tanto de los dispositivos IOT, como de las aplicaciones web (o sea páginas con contenido html) o bien provenientes de un servicio en nodejs . Estos mensajes se organizan bajo estructuras temáticas denominadas tópicos [11-13]. Cuando un dispositivo (por lo general a través de un elemento sensor) o bien una aplicación web, o desde un evento como un webhook (se clarificará más adelante cuando se mencione el motor de reglas), envían información bajo determinado tópico, otros dispositivos u otras aplicaciones web podrán suscribirse a dichos tópicos. De esta manera, sólo aquellos dispositivos o aplicaciones web que se suscriban a determinados tópicos recibirán información de los mismos a través del broker mqtt. Por ende, el mqtt es un protocolo denominado de Publicación/Suscripción y es el encargado de filtrar y administrar la distribución de mensajes entre dispositivos y aplicaciones (ver Figura 12). El broker elegido para esta investigación es el EMQX. Como se ha expresado en la introducción, la elección se debe al alto desempeño profesional que tiene frente a otros brokers (como por ejemplo Mosquitto, CloudMQTT, etc.) y por tener una licencia de uso gratuito capaz de soportar hasta 100K

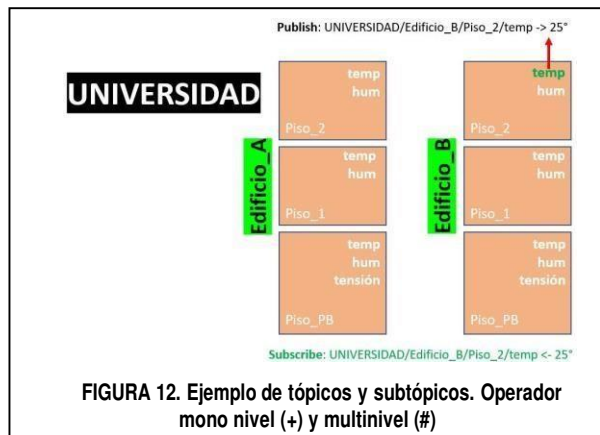


FIGURA 12. Ejemplo de tópicos y subtópicos. Operador mono nivel (+) y multinivel (#)

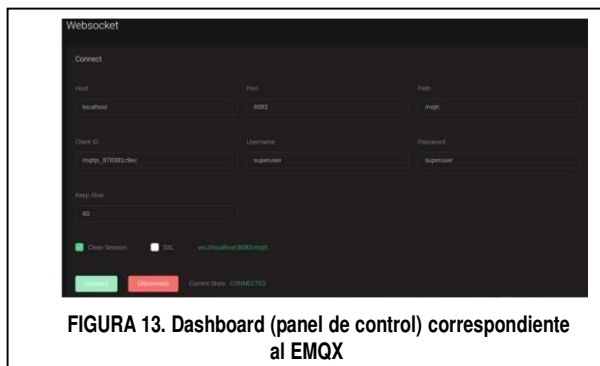


FIGURA 13. Dashboard (panel de control) correspondiente al EMQX

dispositivos. El EMQX se comunica con los dispositivos IOT por medio del puerto TCP 1883, en cambio con las aplicaciones web lo hace por medio de web sockets,

empleando el puerto 8083 para ws, mientras que para wss (Websockets sobre SSL/TLS) [14] se emplea el puerto 8084. Todos estos puertos mencionados, por supuesto son por default y podrán cambiarse configurando los archivos de configuración adecuados en caso de conflictos con otras aplicaciones que por casualidad empleen los mismos puertos. WebSocket es una tecnología que proporciona un canal de comunicación bidireccional y full-duplex sobre un único socket TCP. Está diseñada para ser implementada en navegadores y servidores web, pero puede utilizarse por cualquier aplicación cliente/servidor. Hay otros puertos que también deberán configurarse para el correcto funcionamiento, como el 8883 para SSL y el 8081 para management. Este puerto de management se emplea para comunicarse con la API propia que también suministra EMQX, ya sea para crear, actualizar o borrar recursos, reglas, etc. Por otro lado el puerto 18083 se emplea para conectarse al dashboard del EMQX. El dashboard del EMQX es un cliente web que permite administrar gráficamente el broker de una forma mucho más amigable y es provisto por el propio EMQX y se accede por **http://<IP o DOMINIO>:18083**, (Ver Figura 13). Tomando el ejemplo de tópicos de la Figura 12, el mismo puede interpretarse de la siguiente manera: existe un tópico principal denominado UNIVERSIDAD, la universidad posee dos edificios (A y B), con tres pisos por edificio. A su vez en cada piso se han instalado sensores de humedad y de temperatura y en el Piso_PB se han agregado también módulos de medición de tensión eléctrica por su proximidad a los tableros de energía. Por ende, los edificios, como así también los pisos son subtópicos del tópico principal que como dijimos es UNIVERSIDAD. En el ejemplo, se está emitiendo un mensaje a través del sensor de temperatura bajo el tópico “UNIVERSIDAD/Edificio_B/Piso_2/temp” enviando el valor de 25 °C. Por lo tanto, el dispositivo que se suscriba a dicho tópico recibirá todos los valores de temperatura, pero sólo del Piso_2 del Edificio_B. También puede emplearse el símbolo monovalente “+” o el multivalente “#”. Como ejemplo del operador monovalente podemos indicar que si un dispositivo se suscribiera a “UNIVERSIDAD/+Piso_2/temp” recibiría todos los mensajes de temperatura enviados de ambos edificios, pero sólo del Piso_2, de acuerdo al ejemplo. En cambio como ejemplo de operador multivalente podríamos citar: “UNIVERSIDAD/Edificio_A/#”, en este caso se recibirían todas las variables sensadas (temp, hum) de todos los pisos y la tensión del Piso_PB, pero solamente del Edificio_A. Otra característica muy importante de EMQX (en realidad de mqtt), es lo que se conoce como **QOS** (Quality of Service) [15], el cual puede tomar tres valores posibles: 0, 1 ó 2. Un QOS igual a 0 significa que se confía en TCP para el envío de los mensajes, esto no significa que el mensaje va a llegar necesariamente al usuario, sí llegará al broker, pero éste lo enviará según la disponibilidad de la red, es decir si hubiera un fallo o el servidor se reiniciara, por ejemplo, el mensaje no llegaría al cliente suscripto a un determinado tópico. De todas maneras, es poco probable que suceda si la red es de buena calidad. En cambio si QOS es igual a 1, el broker intentará “al menos una vez” entregar el mensaje al usuario suscripto. Esto da la garantía que el

mensaje será entregado al suscriptor del tópic. Pero en el afán de cumplir con la meta de entrega, podría haber mensajes duplicados, es decir los suscriptores podrían recibir los mensajes en múltiples ocasiones. Esto también va a implicar mayor carga de tráfico y de procesamiento para el broker. Finalmente, una QOS igual a 2 implica que sólo le llegará al suscriptor “una sola vez” el mensaje. Este último caso es importante en la situación en que el suscriptor no pueda recibir mensajes en múltiples ocasiones, pero implica un mayor nivel de procesamiento ya que el broker debe asegurarse que efectivamente el mensaje al suscriptor sólo se entregue en una oportunidad.

5.2 Motor de reglas (Rule engine)

5.2.1 Introducción

EMQX proporciona un esquema de integración de datos en tiempo real, conciso y eficiente mediante la combinación de reglas y bridges de datos. Las reglas se utilizan para procesar mensajes o eventos, y el bridge de datos se utiliza para conectar el sistema de datos. Ver Figura 14.

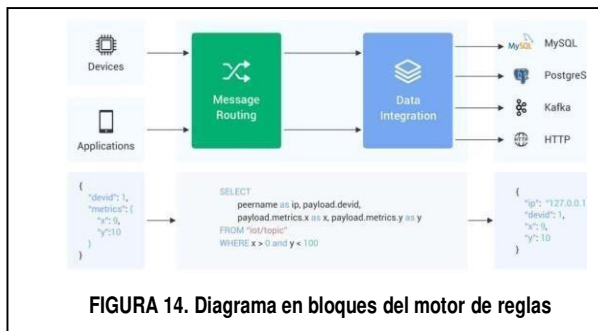


FIGURA 14. Diagrama en bloques del motor de reglas

5.2.2 Explicación de generación de una regla y de un recurso asociado a la regla

EMQX proporciona reglas basadas en la sintaxis SQL para procesar y convertir mensajes o eventos, como convertir tipos de datos, codificar o decodificar mensajes, determinación de una bifurcación condicional, etc. Las reglas están integradas en EMQX y no hay sobrecarga de serialización de mensajes y transmisión de red, por lo que funciona de manera muy eficiente. Esto significa que nosotros podríamos tener una aplicación en NodeJS que realice todo el procesamiento, pero para ello debería estar suscripta a todos los tópicos de todos los clientes del sistema y así determinar las acciones a seguir según los requerimientos de las peticiones, pero esto sin lugar a duda podría generar una verdadera sobrecarga al endpoint (que sería el controlador de los eventos) y afectar seriamente la performance de nuestro servicio. Por eso la idea es dejar todas estas acciones de procesamiento de eventos al EMQX, y nosotros programar los recursos (en este caso los webhooks) que serán los manejadores que atenderán las reglas que programen los usuarios en el broker. Esta notable performance de EMQX se da a la programación bajo Erlang con la que fue construido el broker. La prueba se ha realizado con valores reales de subtópicos extraídos de las colecciones de la base de datos de MongoDB (colecciones users y devices, se detallará más adelante). Ver figuras 15,

16, 17 y 18 respectivamente (esta última la correspondiente al recurso que atiende la regla, o sea el webhook).



FIGURA 15. Escritura de la Regla en lenguaje SQL



FIGURA 16. Comprobación del matcheo de la regla



FIGURA 17. Edición del handler (Webhook)

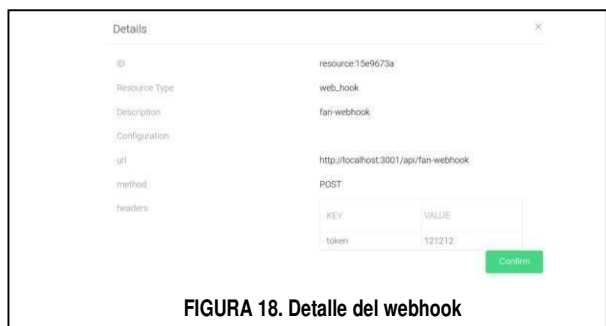


FIGURA 18. Detalle del webhook

6. Planteo del caso de estudio

Se plantea el siguiente escenario:

Se disponen de dos dispositivos (ver Figuras 19 y 20). El dispositivo de la Figura 19 simula el encendido o el apagado de un electroventilador, en donde por debajo de 85 °C se apaga y por encima de 95 °C se enciende. La explicación de la regla y del recurso que la atiende, ya se expuso en las gráficas de las Figuras 15 a 18 respectivamente. Para ello se dispuso de un ESP32, un display I2C, un ventilador de 12 V con la fuente respectiva y el integrado ULN2003 que es el driver para manejar adecuadamente la corriente que requiere el ventilador, ya que la misma no puede ser suministrada por el pin de control del ESP32. Las alarmas, cuando la temperatura supere los 95° C o bien cuando se perfore el umbral de los 85 °C se notificarán al respectivo bot de telegram, al que lo hemos denominado **Conaiisi_2022** (ver luego resultados obtenidos en la Figura 28). Por otro lado, el dispositivo de la Figura 20, está constituido por un ESP8266 y un ESP32. Este hardware podría representar también por ejemplo un control de acceso, ya sea desde abrir una barrera, abrir una puerta con cierre electromagnético, o brindar un café, o cualquier producto (por ejemplo una máquina expendedora de alimentos, bebidas, etc). Para lograr brindar este acceso a un determinado usuario, este dispositivo incorpora un lector de tarjetas RFID que está controlado por el ESP 8266. El lector lee el CardID de la tarjeta RFID (cada usuario cuando se registra incorpora este número en la base de datos users) y la emite bajo un cierto tópico hacia el broker EMQX. EL ESP32 tiene por otro lado dos funciones: por un lado emite bajo un tópico la temperatura interna del microcontrolador y por otro lado recibe dos tópicos provenientes del broker (y por ende también del webhook que es el recurso asociado a la regla que controla el acceso de los usuarios registrados). Es decir, por un lado se recibe el nombre del usuario que se conecta al dispositivo (pasando su tarjeta por el lector RFID) para que sea visualizado en el display SPI y por otro lado se recibe una orden de comando para indicarle al ESP32 que muestre en el display una leyenda que indique que el usuario tiene acceso o no al dispositivo. La regla que controla a este

dispositivo se activa cuando en el payload del tópico se ponga en **1** la llave “enable” enviada en el objeto JSON. El payload que emite el ESP8266 es el siguiente:

```
{“card”: “83475004”, “enable”:1}
```

Acá se muestra el ejemplo del usuario **Carlos** que posee una tarjeta RFID con un CardID como el indicado. A continuación se exponen los tópicos de emisión y de recepción por parte de los dos dispositivos:

Dispositivo 1 (ver Figura 19)

El tópico de emisión que emplearemos será: *userId/deviceId/temp/sensordata*; y los tópicos de recepción empleados que permitirán tanto el encendido como el apagado del ventilador serán: *userId/deviceId/command/actdata*. En este caso el payload a pasarle al ESP32 será simplemente el string “fan_on” para activar el ventilador o “fan_off” para apagarlo.

Dispositivo 2 (ver Figura 20)

Los tópicos de emisión son los siguientes:

Hay un solo tópico para emitir el CardID de la tarjeta RFID en el ESP8266, el tópico es el siguiente: *userId/deviceId/card/sensordata*, (el payload ya fue explicado con anterioridad), mientras que para el ESP32 los tópicos son: *userId/deviceId/tempESP32/sensordata* para la temperatura interna del microcontrolador, mientras que los tópicos de recepción en esta ocasión serán: *userId/deviceId/username/actdata* (para mostrar el nombre del usuario al que se le dio acceso con la tarjeta RFID exitosamente) y



FIGURA 19. Dispositivo que simula un electroventilador en un rango de temperatura (85 °C <temp <95 °C)

userId/deviceId/command/actdata, siendo los valores de los payloads *open*, *close* y *refused*, los cuales actuarán en el ESP32 escribiendo en el display ABRIENDO, CERRANDO y ACCESO DENEGADO respectivamente.

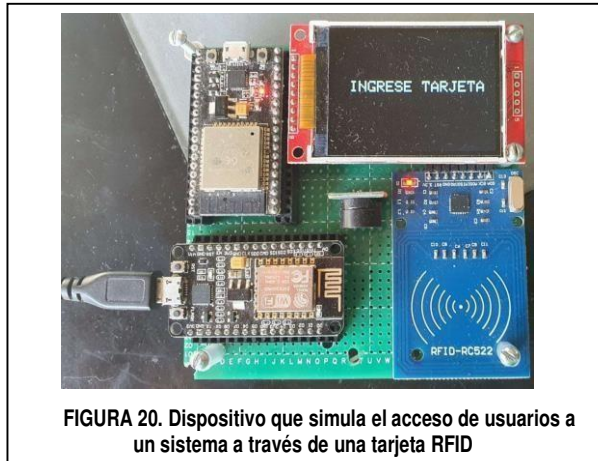


FIGURA 20. Dispositivo que simula el acceso de usuarios a un sistema a través de una tarjeta RFID

7. Desarrollo de la experiencia

Los servicios a activar serán dos contenedores, uno con Mongo (se empleó la versión 4.4) y el otro con EMQX (versión 4.2.2). El node JS empleado fue la versión 14.20 LTS y para el gestor de paquetes NPM se utilizó la versión 8.1.8. En la etapa de desarrollo NodeJS no se empleó en contenedor porque si no se reducía mucho la performance de la máquina. Para levantar los contenedores se empleó docker-compose y se configuró un archivo de configuración denominado *docker-compose.yml*. Todos los servicios han sido probados en sistema operativo Linux server versión 20.04. Para dejar activo permanentemente como un servicio al bot de telegram en nuestro cloudserver se ha utilizado *pm2*. Para acceder a mongodb se ha utilizado el cliente *mongodb-compass*. Para la realización de las pruebas de la API se ha utilizado Postman en su versión de escritorio. Cabe destacar que Postman sustituye en cierta medida al front end (el mismo se encuentra en realización como un trabajo futuro empleando un framework como *nuxt-js*).

8. Resultados obtenidos

8.1 Resultados de la API por medio de Postman

A continuación se indican los resultados obtenidos de un registro de un usuario y de un login (ambos exitosos). En

ambas situaciones, tal como lo indica la captura, los datos se reciben mediante el método POST. También se indica la creación de un componente (método POST) y la verificación del listado de componentes de un usuario dado (método GET). Ver a continuación las figuras 21 a 24. En dicho ejemplo, para que el tamaño de las capturas sea razonable, se ha agregado un único dispositivo. También se indica la captura de las colecciones *users* y *devices* de la base de datos en MongoDB (ver Figura 25, punto 8.2). En cuanto a un resultado obtenido de login (en donde el usuario obtiene el token) ya fue explicado en el punto 4 (Json web token, figuras 9, 10 y 11 respectivamente).



FIGURA 21. Registro exitoso del usuario Carlos - POST



Figura 22. creando con éxito device usuario Carlos - POST



Figura 23. Error por mail o card existente usuario Carlos

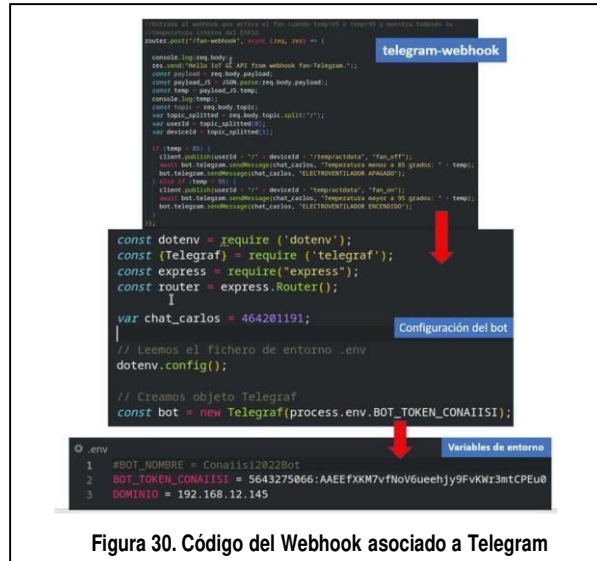


Figura 24. Listando dispositivos usuario Carlos - GET

8.2 Correspondencia con la base de datos (colecciones users y devices)



8.3 Resultados obtenidos con el dispositivo del electroventilador



8.4 Resultados obtenidos con el dispositivo lector de tarjetas RFID

Hacemos notar que el dispositivo pertenece al usuario Carlos (Observar userId y deviceId). Las pruebas se han realizado para tres usuarios denominados Carlos, Eliseo y Guillermo, cuyos respectivos CardID son los siguientes:

Carlos --> 83475004, Eliseo --> bae9911a y Guillermo --> 5a278324

En este caso como puede observarse en la figura 31, la regla se habilitará cuando se reciba el payload con la tarjeta con el campo "enable" en 1, caso contrario no se leerán las tarjetas y el dispositivo quedará deshabilitado. El mensaje recibido por el broker proviene del webhook que se dispara cuando matchea la regla correspondiente.

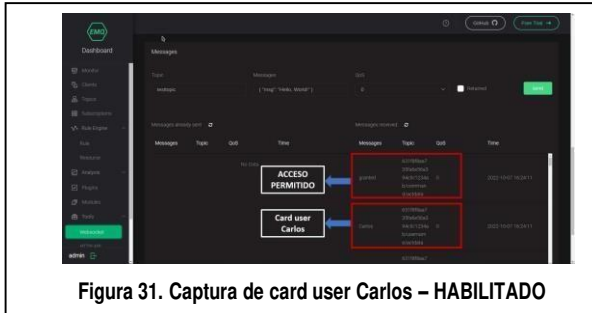


Figura 31. Captura de card user Carlos – HABILITADO



Figura 32. Verificación de accesos permitidos



Figura 33. Acceso denegado para cualquier otra CARD

A continuación observamos la transmisión en el tópic `deviceId/card/sensordata` desde el ESP8266 y el impacto por POST en el webhook asociado con la regla.

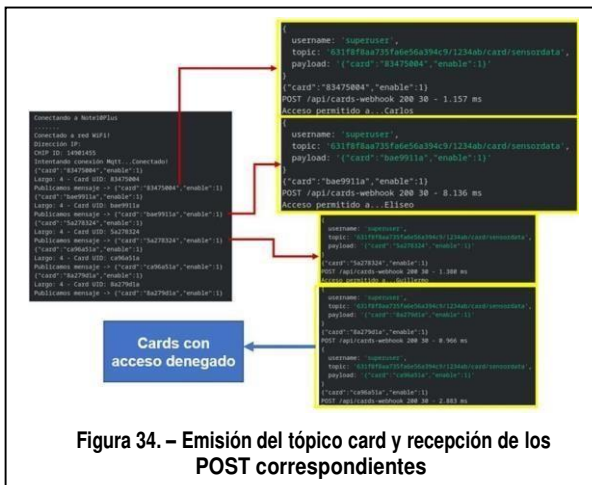


Figura 34. – Emisión del tópic card y recepción de los POST correspondientes

10 Referencias

[1] NodeMCU Connect Things EASY, http://www.nodemcu.com/index_en.html

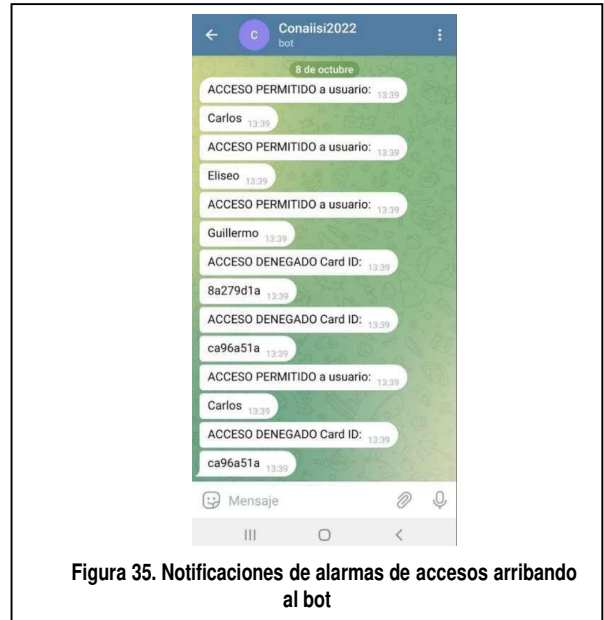


Figura 35. Notificaciones de alarmas de accesos arribando al bot

9 Conclusiones y trabajo futuro

1. En cuanto al número de serie del dispositivo (lo que hemos denominado `deviceId`) a nivel más profesional se podría emplear un chip de la serie Dallas quedando el dispositivo con un valor grabado a nivel de hardware. En este link se encuentra la información correspondiente:

https://www.cryptomuseum.com/df/kolibrie/files/ds2401_pdf

2. Se pueden generar las reglas y los recursos asociados de manera automática empleando una API suministrada por EMQX a través de programación adicional.

3. El empleo de servicios mediante Docker permite que el sistema escale muy fácilmente. Prácticamente se reducen drásticamente los tiempos de desarrollo y es más fácil pasar a la fase de producción, ya que se desarrolla en un entorno local muy fácilmente (en el sentido de que si se rompe un contenedor, el mismo dada su persistencia de datos se recupera rápidamente), y de aquí el paso a la producción resulta casi inmediato.

4. Los webhooks pueden implementarse a futuro a través de un servicio denominado serverless (computación sin servidor). Se trata de un servicio en donde el cliente paga según la cantidad de peticiones recibidas y no debe mantener una máquina virtual para este propósito. Todos los grandes proveedores de servicios en la nube lo brindan actualmente.

Trabajo futuro:

1. Para una segunda actualización del sistema se prevé que los dispositivos adquieran las credenciales de acceso a mqtt a través de un webhook que atienda las peticiones enviadas por POST del ESP32 o ESP8266.

2. Se prevé la realización de un front end empleando un framework tal como Nuxt-JS o REACT.

[2]] <https://www.espressif.com/>

[3] <https://aws.amazon.com/es/what-is/restful-api/>

[4] <https://jwt.io/>

[5] <https://www.mongodb.com/>

[6] EMQX. <https://www.emqx.io/>

[7] Beginning API Development with Node.js: Build highly scalable, developer-friendly APIs for the modern web with JavaScript and Node.js - ISBN-13: 978-1789539660

[8]] <https://telegram.org/>

[9] <https://core.telegram.org/bots#6-botfather>

[10] <https://hub.docker.com/>

[11] Naik, N. (2017, October). Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. In 2017 IEEE international systems engineering symposium (ISSE) (pp. 1-7). IEEE.

[12] Implementación de middleware publicador/ subscriber para aplicaciones web de monitoreo. In XIX Workshop de Investigadores en Ciencias de la Computación (WICC 2017, ITBA, Buenos Aires).

[13] Hands-On Internet of Things with MQTT: Build connected IoT devices with Arduino and MQ.

[14] Arquitectura de software con websocket para aplicaciones web multiplataforma. VI Workshop Innovación en Sistemas de Software (WISS). CACIC 2014.

[15] <https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels>

Certificado



10 mo
CONGRESO
NACIONAL

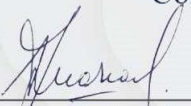
CoNaiISI
2022
modalidad híbrida



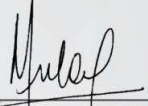
**Carlos Alberto Binker, Hugo Tantignone, Eliseo Zurdo, Guillermo Buranits,
Lautaro Lasorsa**

han participado como autores del trabajo titulado “Sistema de notificación de alarmas mediante Telegram y Bots utilizando webhooks con motor de reglas en EMQX empleando tecnología NodeJS y Docker para dispositivos IOT” y que el mismo ha sido aceptado para ser presentado en el 10° Congreso Nacional de Ingeniería Informática / Sistemas de Información -CONAIISI 2022- organizado por la Red de Carreras de Ingeniería Informática / Sistemas de Información -RIISIC- perteneciente al CONFEDI, realizado de forma híbrida por la Facultad Regional Concepción del Uruguay de la Universidad Tecnológica Nacional, los días 3 y 4 de noviembre de 2022.

Concepción del Uruguay, Entre Ríos, 4 de noviembre de 2022.-


Lic. Augusto Nasrallah
COORDINADOR RIISIC 2022


Mg. Patricia Cristaldo
COORDINADORA CONAIISI 2022


Esp. Ing. Martín E. Herlax
DECANO UTN FRCU



ANEXO II

FPI-013

**FORMULARIO DE EVALUACIÓN DE ALUMNOS INTEGRANTES DE EQUIPOS DE INVESTIGACIÓN**

Unidad Académica: Departamento de Ingeniería e Investigaciones Tecnológicas

Código: C2-ING-076

Título del Proyecto: Interconexión de dispositivos IOT empleando MQTT y NodeJs en redes dual stack

Director del Proyecto: Carlos Alberto Binker

Programa de acreditación: PROINCE CyTMA2: **X**

Fecha de inicio: 01/01/2021

Fecha de finalización: 31/12/2022

1. Datos del alumno

Apellido y Nombre: Zurdo Eliseo Alfredo

DNI: 26353759

Unidad Académica: DIIT

Carrera que cursa: Ingeniería Informática

Período evaluado: 2022

2. Dictamen de evaluación de desempeño del alumno:*Colocar una cruz donde corresponda*2.1 Satisfactorio: **X**

2.1 No satisfactorio:

Fundamentos del dictamen:

Eliseo Zurdo ha cumplido con todas las etapas del proyecto tal lo planificado durante 2022. Ha tenido una participación muy activa y comprometida con el grupo de investigación. Dada su enorme experiencia en programación, constituye una pieza clave para el desarrollo del equipo de investigación.

3. Propuesta de continuidad en el proyecto (si corresponde según duración estimada)*Colocar una cruz donde corresponda*3.1 Continuar en el presente proyecto: **X**

3.2 No continuar en el presente proyecto:

Fundamentos del dictamen:

Dado el gran nivel de actuación y compromiso que ha tenido el alumno Eliseo Alfredo Zurdo es que propongo su continuidad en futuros proyectos de investigación. Eliseo Zurdo ha culminado sus estudios de grado durante 2021 y ha adquirido el título de ingeniero en Informática, por lo que a partir del 2023 su actuación en investigación será ya como *integrante de grupo de investigación*. Además se destaca que éste es el cuarto proyecto consecutivo del cual participó y siempre lo ha hecho con total profesionalismo y compromiso para con el director del proyecto, sus compañeros de grupo y para las autoridades del DIIT.

San Justo, 17/02/2023

Lugar y fecha

Carlos Alberto Binker

Firma del Director

Aclaración de firma



FORMULARIO DE EVALUACIÓN DE ALUMNOS INTEGRANTES DE EQUIPOS DE INVESTIGACIÓN

Unidad Académica: Departamento de Ingeniería e Investigaciones Tecnológicas

Código: C2-ING-076

Título del Proyecto: Interconexión de dispositivos IOT empleando MQTT y NodeJs en redes dual stack

Director del Proyecto: Carlos Alberto Binker

Programa de acreditación: PROINCE CyTMA2: **X**

Fecha de inicio: 01/01/2021

Fecha de finalización: 31/12/2022

1. Datos del alumno

Apellido y Nombre: Frattini, Maximiliano Gabriel

DNI: 26.849.323

Unidad Académica: DIIT

Carrera que cursa: Ingeniería en Informática

Período evaluado: 2022

2. Dictamen de evaluación de desempeño del alumno:

Colocar una cruz donde corresponda

2.1 Satisfactorio: **x**

2.1 No satisfactorio:

Fundamentos del dictamen:

Maximiliano Frattini ha cumplido con todas las etapas del proyecto, tal lo planificado durante 2022. Ha tenido una participación activa y comprometida con el grupo de investigación.

3. Propuesta de continuidad en el proyecto (si corresponde según duración estimada)

Colocar una cruz donde corresponda

3.1 Continuar en el presente proyecto: **X**

3.2 No continuar en el presente proyecto:

Fundamentos del dictamen:

Dado el gran nivel de actuación y compromiso que ha tenido el alumno Maximiliano Gabriel Frattini es que determino su continuidad en un futuro proyecto de investigación.

San Justo, 17/02/2023

Lugar y fecha

Firma del Director

Carlos Alberto Binker

Aclaración de firma



FORMULARIO DE EVALUACIÓN DE ALUMNOS INTEGRANTES DE EQUIPOS DE INVESTIGACIÓN

Unidad Académica: Departamento de Ingeniería e Investigaciones Tecnológicas

Código: C2-ING-076

Título del Proyecto: Interconexión de dispositivos IOT empleando MQTT y NodeJs en redes dual stack

Director del Proyecto: Carlos Alberto Binker

Programa de acreditación: PROINCE CyTMA2: **X**

Fecha de inicio: 01/01/2021

Fecha de finalización: 31/12/2022

1. Datos del alumno

Apellido y Nombre: Lasorsa, Lautaro

DNI: 42.394.430

Unidad Académica: DIIT

Carrera que cursa: Ingeniería en Informática

Período evaluado: 2022

2. Dictamen de evaluación de desempeño del alumno:

Colocar una cruz donde corresponda

2.1 Satisfactorio: **x**

2.1 No satisfactorio:

Fundamentos del dictamen:

Lautaro Lasorsa ha cumplido con todas las etapas del proyecto de acuerdo a lo planificado durante 2022. Ha tenido una participación muy activa y comprometida con el grupo de investigación. Dada su enorme experiencia en programación, constituye una pieza clave para el desarrollo del equipo de investigación. Ha completado de manera satisfactoria también su trayectoria en el plan de "Becas de Investigación Científica y Becas de Desarrollo Tecnológico y Social de la Universidad Nacional de La Matanza".

3. Propuesta de continuidad en el proyecto (si corresponde según duración estimada)

Colocar una cruz donde corresponda

3.1 Continuar en el presente proyecto: **X**

3.2 No continuar en el presente proyecto:

Fundamentos del dictamen:

Dado el gran nivel de actuación y compromiso que ha tenido el alumno Lautaro Lasorsa es que determino su continuidad en un futuro proyecto de investigación.

San Justo, 17/02/2023
Binker

Lugar y fecha

Firma del Director

Carlos Alberto

Aclaración de firma