# Desarrollo de herramientas y aplicaciones para Internet de las cosas y aprendizaje automático utilizando la computadora *Raspberry Pi*

Carlos René Gil Monterroso

UVG UNIVERSIDAD DEL VALLE DE GUATEMALA

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería


# Desarrollo de herramientas y aplicaciones para Internet de las cosas y aprendizaje automático utilizando la computadora *Raspberry Pi*

Trabajo de graduación presentado por Carlos René Gil Monterroso para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

2024

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería


# Desarrollo de herramientas y aplicaciones para Internet de las cosas y aprendizaje automático utilizando la computadora *Raspberry Pi*

Trabajo de graduación presentado por Carlos René Gil Monterroso para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

2024

Vo.Bo.:

(f) _____

MSc. Jonathan de los Santos

Tribunal Examinador:

(f) _____

MSc. Jonathan de los Santos

(f) _____

MBA. Pablo Mazariegos

(f) _____

Ing. Yosemite Meléndez

Fecha de aprobación: Guatemala, 6 de enero de 2024.

# Prefacio

Agradezco a mi papá por darme la oportunidad de estudiar en la universidad y apoyarme en todo lo que he necesitado lo largo de todo este tiempo. Agradezco a mis tías, abuelita y hermano por su ayuda de diferentes maneras, que representaron motivación para la culminación de este proceso. Agradezco a mis compañeros de electrónica por compartir conmigo cada momento, prestarme su ayuda, y hacer que cada momento a lo largo de la carrera haya sido memorable. Finalmente, de manera especial, agradezco a mi mamá por todo lo que me ha dado a lo largo de mi vida, por enseñarme y guiarme por el camino correcto, siendo estas las principales razones por las que logré culminar este proceso de manera exitosa. Pero sobre todo, por siempre creer en mí y ser mi soporte en cada uno de mis momentos de debilidad durante todo el proceso que tuve que llevar a cabo para lograr esta meta.

# Índice

# Lista de figuras

# Resumen

Este trabajo de graduación presenta varias propuestas de herramientas y aplicaciones para Internet de las Cosas y aprendizaje automático en la computadora *Raspberry Pi*. Inicialmente se realizó una evaluación de distintos tipos de herramientas de código abierto como, sistemas operativos existentes diseñados específicamente para funcionar en la *Raspberry Pi*, plataformas de Internet de las Cosas, sistemas de bases de datos y librerías de aprendizaje automático. Con los resultados obtenidos, se diseñaron herramientas para utilizar a la *Raspberry Pi* como *Smart Gateway* para dispositivos IoT de distintos protocolos de comunicación, para almacenar en base de datos los valores de cada uno de estos sensores y luego utilizar todos estos datos para entrenar diferentes modelos de aprendizaje automático. Finalmente, utilizando todas las herramientas realizadas, se desarrolló una aplicación de una casa inteligente que tiene sensores de 5 protocolos diferentes, con diferentes métodos de administración para los sensores y almacenamiento de cambios para los estados de cada sensor. Añadido a esto, se utilizaron los datos almacenados para entrenar diferentes modelos de aprendizaje automático, cuyos resultados se le presentaron al usuario como recomendaciones para rutinas de automatización que él podría integrar al sistema.

# Abstract

This thesis presents several proposals for tools and applications for the Internet of Things and machine learning on the Raspberry Pi computer. Initially, an evaluation of different types of open source tools, such as existing operating systems specifically designed to run on the Raspberry Pi, Internet of Things platforms, database systems and machine learning libraries. The results have been used to develop tools that use the Raspberry Pi as a smart gateway for IoT devices with different communication protocols, to store the values of each of these sensors in a database and then use all this data to train different algorithms. Then use all this data to train different machine learning models. Finally, using all the tools developed, a smart home application was developed that has sensors from 5 different protocols, with different management methods for the sensors and storage of sensor changes. In addition, the stored data was used to train different machine learning machine learning models, the results of which were presented to the user as recommendations for automation routines that he could use.

# CAPÍTULO 1

## Introducción

Internet de las Cosas (IoT por sus siglas en inglés) se define como una red mundial de objetos interconectados direccionables, basada en protocolos de comunicación estándar [1]. Sin embargo, la importancia del concepto nace de su capacidad de conectar una variedad de dispositivos de distintos tipos, que se comunican a través de diferentes protocolos, que poseen distintos tipos de inteligencia y son proporcionados por diferentes proveedores. [2]

Dentro del concepto de IoT, es sencillo visualizar que existe una tendencia hacia el control y recopilación de datos. De modo que, en un sistema de IoT se pueden integrar sistemas de bases de datos para aportar permanencia a los datos de los sensores que ya se tenían conectados, dándole robustez al sistema y un mayor control al usuario final. Así mismo, para una mejor administración de los datos, es usual que se integren herramientas gráficas para visualizar los datos. Las herramientas de visualización, además de aportar vistosidad al sistema, facilitan la comprensión de los datos y le dan al usuario final la oportunidad de realizar análisis rápidos y confiables.

Teniendo el poder de los datos, además de poder obtener resultados estadísticos convencionales, se pueden extraer conjeturas mucho más profundas al implementar conceptos como el aprendizaje automático. El aprendizaje automático consiste en la creación de sistemas informáticos con la capacidad de mejorar automáticamente con base en la experiencia e implementar procesos de aprendizaje [3]. De este modo, el resultado de integrar un sistema de IoT con almacenamiento de datos y algoritmos de aprendizaje automático abre las posibilidades a realizar aplicaciones de alta complejidad, que no se hubieran podido lograr al realizarlas mediante la aplicación de cada uno de estos conceptos de manera aislada.

# CAPÍTULO 2

# Antecedentes

## 2.1. Curso de Electrónica digital 3

En este curso impartido por la Universidad del Valle de Guatemala, se exploran diversas alternativas para el desarrollo de sistemas embebidos con la computadora *Raspberry Pi*. Los temas cubiertos por el curso son el manejo de puertos digitales, programas con hilos y procesos múltiples, sincronización entre procesos, interrupciones, protocolos de comunicación, sensores, *drivers*, comunicación de red e Internet de las cosas. Los programas desarrollados en el curso son realizados en lenguaje C, con ayuda principal de una librería llamada *wiringPi* [4]. Esta librería presenta una serie de herramientas que permiten interactuar de manera sencilla con las funciones de los puertos de la computadora. A lo largo del curso se realizan laboratorios dedicados a la validación de cada uno de los temas desarrollados y un proyecto final. Este proyecto consiste en explorar el concepto de un sistema de Supervisión, Control y Adquisición de datos SCADA. Para su elaboración es necesario integrar los temas desarrollados en el curso, dispositivos de IoT y desarrollo de interfaces gráficas para el usuario final.

## 2.2. Herramientas de IoT para *Raspberry Pi*

El Internet de las Cosas (IoT) generalmente se refiere a los escenarios en los que la capacidad computacional y la conectividad a la red se extiende a dispositivos que normalmente no son considerados como computadoras. La *Raspberry Pi* es una computadora con una serie de pines de propósito general (GPIO), que el usuario puede administrar a conveniencia para interactuar con sensores y actuadores de diferentes tipos, lo que la hace una herramienta idónea para la implementación de proyectos de IoT. Dentro de la gran variedad de herramientas existentes para el desarrollo de este tipo de proyectos, se tienen dos que son específicamente diseñadas para usarse con la *Raspberry Pi*. *WebThings* [5] es una implementación de código abierto de *Web of Things*, incluyendo el *WebThings gateway*, *WebThings framework* y

*WebThings Cloud*. Específicamente para el *gateway*, *WebThings* ofrece una imagen de sistema operativo para la *Raspberry Pi*. Ofreciéndole al usuario una interfaz basada en web para supervisar y controlar dispositivos inteligentes, un motor de reglas para automatizarlos y un sistema complementario para ampliar la compatibilidad con una gran variedad de dispositivos inteligentes domésticos existentes. *Picroft* [6] es una manera de ejecutar *Mycroft* en una *Raspberry Pi 3, 3B+ o 4*, mediante la instalación de una imagen de sistema operativo en la microSD de la *Raspberry Pi*. Con esto se logra convertir la *Raspberry Pi* en una asistente virtual como *Amazon Alexa*, *Google Assistant*, *Microsoft Cortana* y *Apple's Siri*. La ventaja que presenta *Mycroft* a diferencia de los asistentes mencionados es que es de código abierto, lo que le permite a los usuarios inspeccionar, copiar, modificar y contribuir a la comunidad para que todos se beneficien con eso.

## 2.3. Uso de Raspberry en aplicaciones de IoT

En el artículo citado en [7] se muestra un sistema inteligente de monitorización sanitaria usando la *Raspberry Pi* en una plataforma de IoT. El sistema desarrollado consta de una parte de potencia para la alimentación del sistema, sensores de respiración, sensores EGC, sensores de aceleración, sensores de presión sanguínea y sensores de temperatura. Cada uno de estos esta enviando información a la *Raspberry Pi*, la cual procesa y almacena los valores en el servidor IoT, además de mostrar localmente los valores más recientes. Con esto se logra que el doctor pueda ver los datos del paciente desde que se registró en la plataforma, permitiéndole tener una visión completa del estado del paciente para sugerir o cambiar cualquier tipo de medicina.

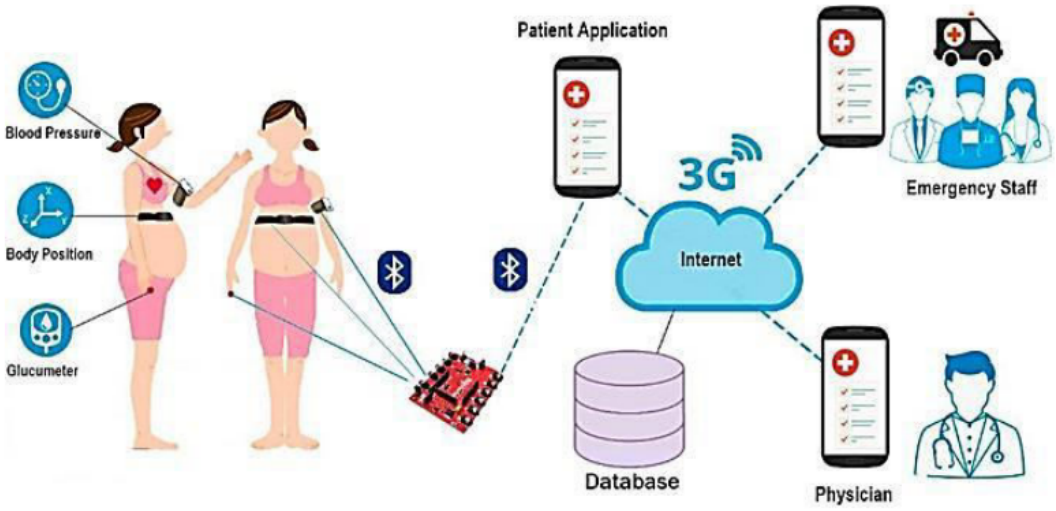


Figura 1: Arquitectura virtual de la aplicación realizada.

## 2.4. Uso del arduino nano 33 IoT para aplicaciones de IoT

En el artículo citado en [8] se describe un proyecto que consiste en el diseño y elaboración de una araña robótica con funciones de localización y mapeo simultáneo (SLAM por sus siglas en inglés) y reconocimiento de objetos, cuyo objetivo principal es mejorar el aprendizaje de conceptos de robótica de manera remota. El proyecto combina técnicas como la localización y cartográfica simultáneas basadas en la visión y detección de objetos. Los dispositivos periféricos utilizados en este proyecto son el ratón, el teclado, el monitor, un control de PS4 y el iPad. El ratón, el teclado y el monitor se utilizan para administrar los programas, el control PS4 se utiliza para controlar la araña de forma remota y la *Raspberry Pi* y el *iPad* trabajan en una red local para garantizar que puedan compartir los mismos resultados de procesamiento de datos. El *iPad* sólo se usa para visualización, de modo que el resultado del SLAM se mostrará en el iPad a través del bus Bluetooth. El arduino nano 33 IoT es usado para comunicarse entre los miembros del grupo. El prototipo lo tiene el líder del grupo y los demás miembros pueden controlarlo de forma remota a través de la aplicación IoT.

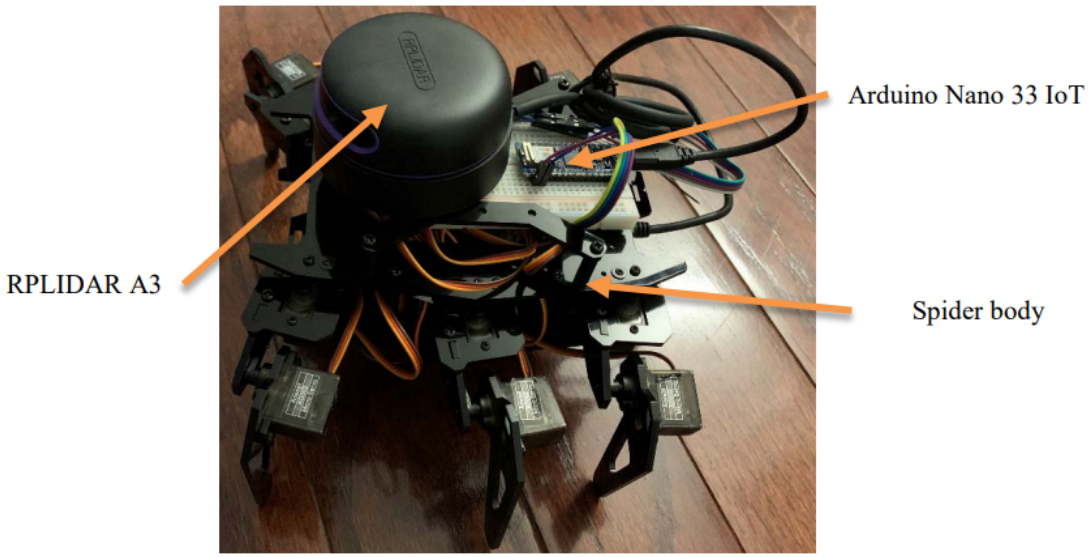


Figura 2: Prototipo final de la araña robótica.

## 2.5. Uso del microcontrolador ESP32 para aplicaciones de IoT

En el artículo citado en [9] se presenta la implementación de un sistema de control, supervisión y adquisición de datos (SCADA) utilizando el microcontrolador ESP32 como unidad terminal remota (RTU) y la plataforma de IoT *Thinger.io* como unidad terminal maestra (MTU). El sistema completo consiste en sensores de corriente y voltaje, el microcontrolador ESP32, una *Raspberry Pi* y un cablerouter wifi. Para su implementación los sensores de corriente y voltaje envían los datos al ESP32, este los procesa y envía a la plataforma de IoT *Thinger.io* para el almacenamiento de los datos, la visualización en tiempo real y el control remoto. El servidor de *Thinger.io* esta localmente en la *Raspberry Pi*, mientras que

se crea el canal de comunicación usando el cablerouter wifi.

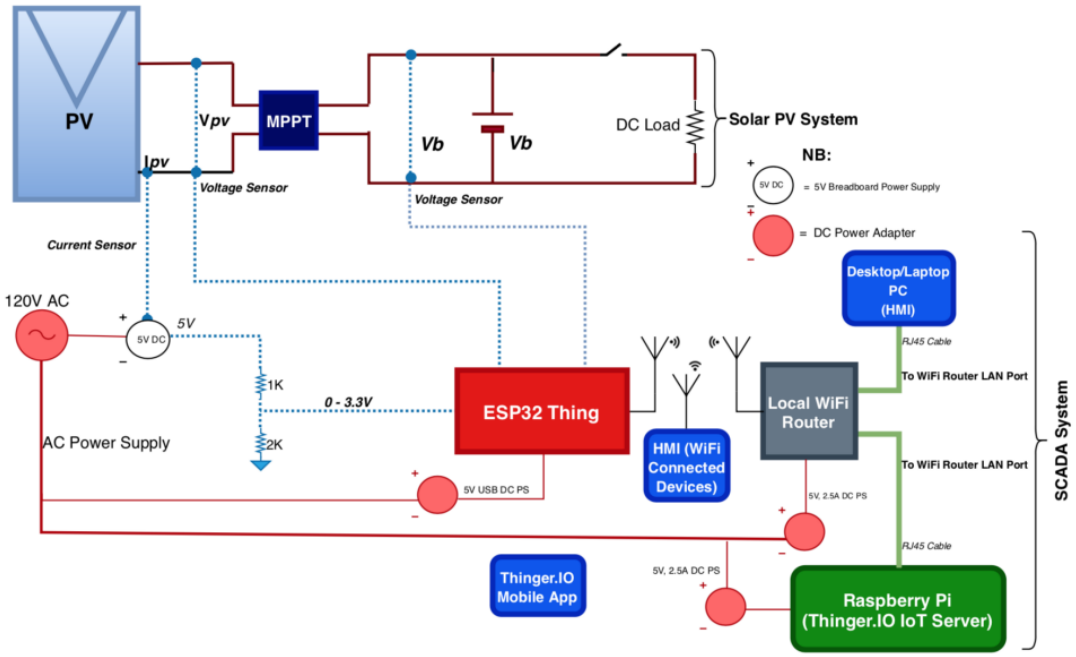


Figura 3: Configuración del sistema SCADA planteado.

## 2.6. Uso de *RaspBerry Pi* para aplicaciones de aprendizaje automático

En el artículo en [10] presenta la implementación de un sistema de predicción de lluvias, utilizando un algoritmo de aprendizaje automático. En el sistema propuesto, la aplicación fue desarrollada en la *Raspberry Pi 3*. Esta aplicación toma los valores de sensores de presión, humedad y temperatura en tiempo real, los procesa y los utiliza para establecer la posibilidad de que llueva ese día. Todos los sensores están conectados a los pines de propósito general (GPIO) de la *Raspberry Pi*. La aplicación realizada consta de 3 diferentes módulos programados en lenguaje Python. El primer módulo adquiere los valores de los sensores utilizados, el segundo módulo implementa el algoritmo de aprendizaje automático, el cual es entrenado de acuerdo al modelo de clasificación. Para ello, toma los datos obtenidos por el primer módulo como un arreglo de 3x3, con ellos el sistema produce un resultado entre 0 y 1, donde 0 es que no lloverá y 1 que sí. Finalmente, el tercer módulo es una interfaz gráfica para el usuario (GUI), donde el usuario es capaz de visualizar los datos obtenidos del sensor, comenzar la captura de datos de los sensores, predecir el clima y mostrar la conclusión final al usuario.

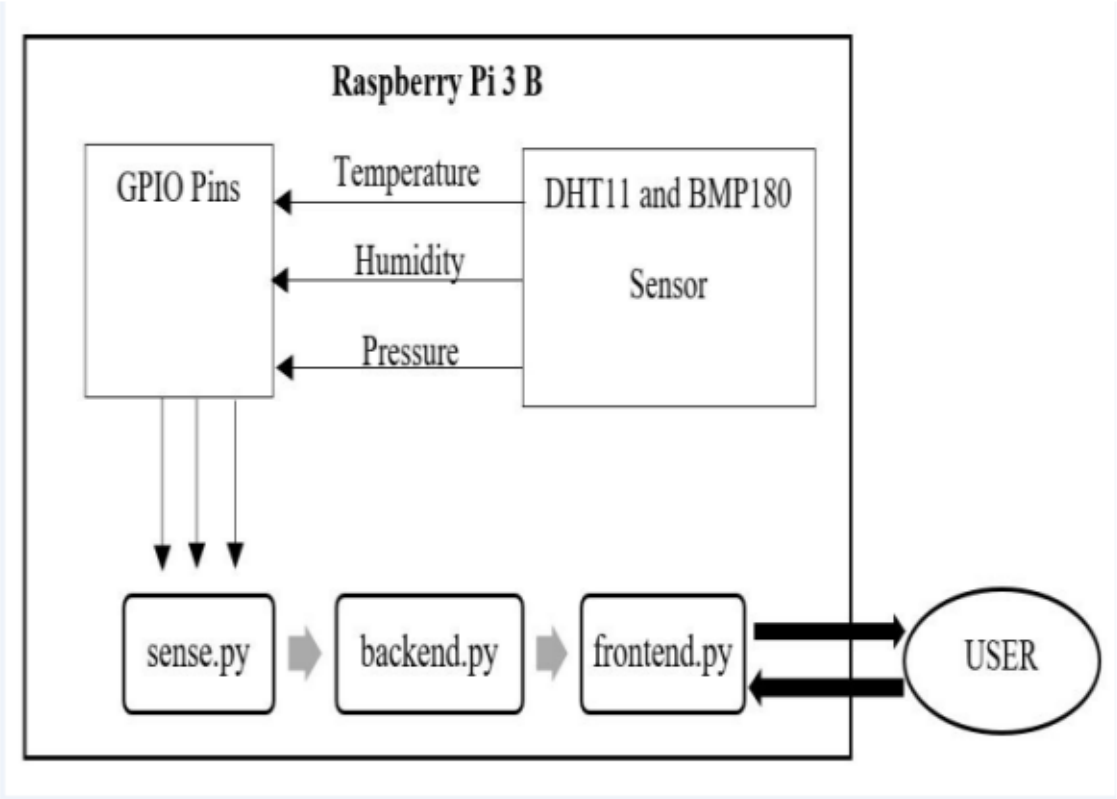


Figura 4: Configuración del sistema de aprendizaje automático para predecir el clima.

# CAPÍTULO 3

## Justificación

El Internet de las Cosas (IoT por sus siglas en inglés) es un concepto que ha tomado relevancia en los últimos años, junto con el crecimiento de los dispositivos móviles, la comunicación omnipresente, la informática en la nube y el análisis de datos. Lo que nos ha llevado a evolucionar a un mundo en el que los objetos pueden registrar, comunicar y transmitir información conectados sobre redes IP públicas o privadas. Estos objetos interconectados poseen datos recopilados y analizados, los cuales utilizan para iniciar cualquier tipo de acción, lo que les da una noción de inteligencia para la planificación, administración y toma de decisiones [2]. La versatilidad del IoT es tal que se tienen aplicaciones en todos los tipos industrias. Mediante la implementación de este concepto se pueden crear desde redes con objetos cotidianos, como focos, enchufes, interruptores y actuadores domésticos, hasta redes con sensores de precisión para medicina, como en el ejemplo mencionado en [7]. Todas estas iniciativas de implementación generan la necesidad de desarrollar nuevos productos que tengan embebidas las capacidades computacionales y de conectividad, así como de desarrollar metodologías para darle estas capacidades a todos aquellos dispositivos que no las poseen.

En conjunto con todos los avances enfocados en la recopilación y almacenamiento de datos, surgen herramientas para el análisis de datos capaces de resolver problemas que quedan fuera del área de aplicación de las herramientas estadísticas tradicionales. Las herramientas analíticas con enfoque en una menor dependencia de conocimientos previos, como el aprendizaje automático, tienen potentes capacidades de ajuste, lo que ha demostrado su capacidad para resolver patrones y formatos de datos complejos. A consecuencia de esto, en la última década el aprendizaje automático ha tenido un crecimiento avanzado, especialmente en el aprendizaje profundo, para aplicaciones de de diferentes tipos, como la clasificación, motores de búsqueda y predicción, como el ejemplo mostrado en [10]. Además, ha revolucionado todos los campos científicos [11].

El Internet de las Cosas y el aprendizaje automático, presentan una amplia variedad de ventajas en su implementación individual, sin embargo, si se implementan de manera conjunta, se obtienen como resultado sistemas capaces de desempeñar funciones bastante complejas. La microcomputadora *Raspberry Pi* tiene capacidades de procesamiento, conecti-

vidad, portabilidad y versatilidad que la hacen un dispositivo ideal para ser utilizada. como unidad central, en la implementación de este tipo de sistemas. Además de ello, es un dispositivo con alta disponibilidad en el mercado, precio accesible y un alto número de usuarios a nivel mundial, por lo que todo lo que desarrolle para trabajar con esa plataforma, es práctico de implementar para nuevos usuarios y un aporte significativo para los usuarios existentes [12].

# CAPÍTULO 4

# Objetivos

## 4.1. Objetivo general

Desarrollar herramientas y aplicaciones de Internet de las Cosas (IoT) y aprendizaje automático, donde se utilice a la *Raspberry Pi* como *Smart Gateway* para dispositivos que tengan o carezcan de funcionalidades de IoT.

## 4.2. Objetivos específicos

- Evaluar herramientas de código abierto para usar la *Raspberry Pi* como *Smart Gateway* en aplicaciones de IoT.
- Diseñar herramientas para usar la *Raspberry Pi* como *Smart Gateway* en aplicaciones de IoT.
- Diseñar herramientas de código abierto para usar la *Raspberry Pi* en aplicaciones de aprendizaje automático.
- Agregar la funcionalidad IoT a dispositivos no inteligentes, utilizando los microcontroladores ESP32 y arduino nano 33 IoT.
- Desarrollar una aplicación gráfica demostrativa donde se integren las herramientas de aprendizaje automático e IoT seleccionadas.

# CAPÍTULO 5

## Alcance

En este trabajo se busca presentar una serie de herramientas que permitan utilizar a la *Raspberry Pi* como un *Smart Gateway* dentro de un sistema de IoT. Esto quiere decir que las herramientas diseñadas le dan a la *Raspberry Pi* la capacidad de administrar sensores y actuadores de protocolos Bluetooth, wifi, Zigbee, Z-wave y MQTT, ya sean nativos del respectivo protocolo o utilicen un dispositivo intermediario para lograr la comunicación. La posibilidad de almacenar el valor de cada uno de estos sensores en bases de datos.

En conjunto con lo que se mencionó anteriormente, se integraron herramientas de aprendizaje automático al sistema, con lo que se pretende dejar de un lado los márgenes establecidos por una plataforma de IoT común, en la que las posibilidades se limitan a un control remoto y almacenamiento de estados, con las herramientas presentadas para elevar el nivel de complejidad de las aplicaciones IoT a otro nivel. Mostrando las posibilidades que se tienen al mezclar los conceptos de IoT con aprendizaje automático en un mismo sistema.

Por la naturaleza de las herramientas que se desarrollaron, la cantidad de aplicaciones existentes es inmensurable. Sin embargo, con la aplicación demostrativa diseñada se procura utilizar cada una de las herramientas elaboradas, no solo para validar su funcionamiento individual sino para ejemplificar la manera en la que se potencia el valor de cada elemento por separado, al momento en el que se integra en una sola aplicación. Buscando establecer un precedente en relación de los conceptos de aprendizaje automático e IoT, para abrir el campo de nuevas lineas de investigación y desarrollo para aplicar los conceptos tratados en este trabajo, en cualquier aplicación de cualquier industria.

# CAPÍTULO 6

# Marco teórico

## 6.1. Conceptos

### 6.1.1. Internet de las Cosas

En [1] se define al Internet de las Cosas (IoT por sus siglas en inglés) como una red mundial de objetos interconectados direccionables, basada en protocolos de comunicación estándar. Sin embargo, la importancia del concepto nace de su capacidad de conectar una variedad de dispositivos de distintos tipos, como objetos cotidianos, sensores inteligentes, sistemas informáticos, redes informáticas y objetos inteligentes, los cuales tienen distintos tamaños, diseños y sistemas, se comunican a través de diferentes protocolos, poseen distintos tipos de inteligencia y son proporcionados por diferentes proveedores. [2]

### 6.1.2. Aprendizaje automático

El aprendizaje automático consiste en la creación de sistemas informáticos con la capacidad de mejorar automáticamente con base en la experiencia e implementar procesos de aprendizaje. De otra manera, puede definirse como una forma de aprender la teoría de manera automática, a partir de los datos. Mediante procesos de inferencia, ajuste de modelos y aprendizaje a partir de ejemplos. [3]

### 6.1.3. Base de datos

Una base de datos es una recopilación organizada de información estructurada que se almacena de manera electrónica en un sistema informático. Normalmente son controladas por un sistema de gestión de bases de datos, que en conjunto con los datos y las aplicaciones asociadas a ellos se les conoce como sistemas de bases de datos. Para interactuar con este

sistema se utiliza un lenguaje de consulta estructurada (SQL). Este es un lenguaje de programación que se utiliza para consultar, manipular y definir los datos, además de proporcionar control de acceso. [13]

### 6.1.4. Interfaz de Programación de Aplicaciones

Una Interfaz de Programación de Aplicaciones (API por sus siglas en inglés) es un mecanismo que le permite a dos componentes de software comunicarse entre sí, utilizando un conjunto de definiciones y protocolos. La arquitectura de las APIs es de cliente - servidor. En el contexto de las APIs, la palabra aplicación se refiere a las distintas aplicaciones de cualquier software, la interfaz es un contrato de servicio entre dos aplicaciones. Ahí se definen la manera en la que va a tener lugar la comunicación. Dentro de la documentación de las APIs se encuentra la manera en que deben estructurase las solicitudes y respuestas. [14]

## 6.2. Dispositivos

### 6.2.1. *Raspberry Pi* 3B

La *Raspberry Pi* 3B es una microcomputadora que posee un Sistema de Chip (SoC) Broadcom BCM2837, con 4 núcleos de procesamiento ARM Cortex-A53 de alto rendimiento a 1.2 GHz con 32KB de memoria caché de nivel 1 y 512KB de nivel 2, un procesador de gráfico VideoCore IV y un módulo de memoria LPDDR2 de 1GB en la parte trasera de la placa. Posee 4 puertos USB integrados y un puerto micro USB para alimentación. Así también, posee 40 pines de propósito general (GPIO) junto con la capacidad de comunicación mediante distintos protocolos como UART, SPS e I2C. Para la comunicación inalámbrica, esta microcomputadora maneja Bluetooth de bajo consumo (BLE) y wifi, sin necesidad de utilizar una antena externa, ya que posee una soldada directamente a la placa. Por el lado del software, hay un sistema operativo optimizado especialmente para utilizarse con la *Raspberry Pi*. Este sistema operativo se conoce como Raspbian y esta basado en la distribución de Linux, Debian. Raspbian ofrece aproximadamente 35,000 paquetes adicionales, que son software precompilado agrupado en un formato que permite una fácil instalación en la *Raspberry Pi*. [15]

### 6.2.2. Arduino nano 33 IoT

El arduino nano 33 IoT es una placa con un microcontrolador de baja potencia ARM Cortex-M0 de 32 Bits SAMD21. Posee conectividad por wifi y Bluetooth brindada por un módulo de u-blox, el NINA-W10. Este es un chipset de baja potencia que opera en el rango de 2.4 GHZ. En la parte superior de estos, el crypto chip Microchip ECC608 garantiza una comunicación segura a través de él. Por otro lado, esta placa posee 14 pines digitales, 11 pines de modulación por ancho de pulso (PWM), 1 módulo UART, 1 módulo SPI, 1 módulo I2C y 8 pines analógicos capaces de dar conversiones con una precisión de 8, 10 y 12 bits. [16]

### 6.2.3. ESP32-WROOM-32

El ESP32-WROOM-32 es una placa de desarrollo que forma parte de los modelos ESP32. Esta placa tiene un amplio conjunto de periféricos, optimizados para realizar prototipos sin mayores inconvenientes. Posee un microcontrolador ESP32-D0EDQ6, con dos CPUs que pueden ser controlados de manera independiente con una frecuencia de reloj ajustable entre 80MHz y 240MHz. Puede manejar protocolos de comunicación inalámbrica como wifi, Bluetooth y Bluetooth LE. Además, posee interfaces para memoria SD, UART, SPI, SDIO, I2C, LED PWM, Motor PWM, I2S, IR, Contador de pulsos, pines de propósito general, sensor capacitivo de contacto, módulos ADC, DAC y TIWAI. [17]

## 6.3. Protocolos de comunicación

### 6.3.1. ZigBee

ZigBee es un protocolo de red respaldado únicamente por la alianza ZigBee, que utiliza los servicios de transporte que forman parte de la especificación de red IEEE 802.15.4. La alianza ZigBee define las capas de red, seguridad y aplicación. Mientras que la IEEE 802.15.4, define las capaz físicas de control y acceso al medio para LR-WPAN. La potencia que requiere es considerablemente pequeña, consumiendo a lo sumo 1mW, en la mayoría de los casos. Posee un alcance de hasta 150m en exteriores, utilizando la técnica de espectro ensanchado de secuencia directa (DSSS).

La frecuencia en la que funciona es de 868 MHz en Europa, 915 MHz en Norteamérica y Australia y 2.4 GHz en todo el mundo, con velocidades de datos de hasta 20kbps, 40 kbps y 250 kbps respectivamente. Estas bandas son diferentes a las de las demás redes inalámbricas, como Bluetooth, wifi, USB inalámbrico etc, lo que implica que al usar este protocolo no se tienen interferencias con otras redes inalámbricas. El estándar IEEE 802.15.4 utiliza direcciones de 64 bits y 16 bits, para lograr direccionar más de 65,000 nodos por red. Una red ZigBee puede tener hasta 65535 dispositivos, los cuales pueden estar separados hasta 50 metros y cada nodo puede retransmitir datos a otros nodos, lo que le da la capacidad de hacer una red considerablemente grande, capaz de cubrir una gran distancia.

La arquitectura del protocolo esta basada en el modelo de interconexión abierta de sistemas (OSI). ZigBee se basa en el estándar IEEE 802.15.4, que define la capa física y la capa de control de acceso al medio (MAC). La alianza ZigBee define la capa de red y de aplicación.

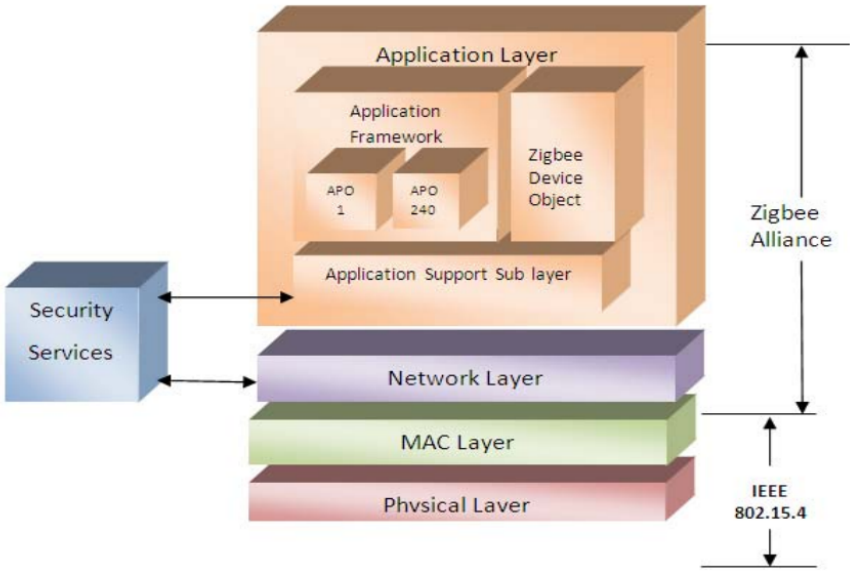


Figura 5: Capas del protocolo ZigBee.

La capa física de la norma IEEE 802.15.4 es la capa más cercana al hardware, que puede controlar y comunicar directamente con el transmisor de radio. Esta capa se encarga de todas las tareas relacionadas con el acceso al hardware ZigBee, incluyendo la inicialización del hardware, selección del canal, estimación de la calidad del enlace, medición de energía y evaluación del canal para ayudar con la selección del canal. Admite tres bandas de frecuencia, la de 2.45GHz que usa 16 canales, 915 MHz con 10 canales y 868 MHz con 1 canal y las tres utilizan el modo DSSS.

La capa de control de acceso al medio proporciona una interfaz entre la capa física y capa de red. Proporciona dos servicios principales, servicio de datos y de gestión, que interactúa con el punto de acceso de la entidad de gestión de sub capa MAC (MLME-SAP). El servicio de datos MAC habilita la transmisión y recepción de las unidades de datos del protocolo MAC (MPDUs) a través del servicio de datos PHY. La capa MAC es la responsable de generar balizas y sincronizar los dispositivos con la señal de baliza en un servicio habilitado para balizas. También, realiza funciones de asociación y disociación. Además, define cuatro estructuras de trama, trama de baliza, trama de datos, trama de confirmación y trama de comandos MAC.

La capa de red establece una interfaz entre la capa de aplicación y capa MAC. Esta capa es la responsable de la formación de redes y el enrutamiento. El enrutamiento es el proceso de selección de la ruta para la retransmisión de los mensajes al nodo de destino. Esto forma la red, que incluye la entrada y salida de nodos, las tablas de enrutamiento, el enrutamiento real y la asignación de direcciones El encargado de realizar el descubrimiento de direcciones, es el enrutador ZigBee. Esta capa proporciona seguridad en toda la red y permite a los dispositivos de bajo consumo maximizar la duración de la batería. Con base en las topologías básicas, hay tres topologías de red que se consideran en el IEEE802.15.4, estrella, árbol y malla.

La capa de aplicación es la capa de protocolo más alta y almacena los objetos de aplicación. La especificación ZigBee separa la capa de aplicación en tres subcapas diferentes, la subcapa de soporte de aplicaciones, los objetos de dispositivo ZigBee y el marco de aplicación con objetos de aplicación. Los objetos de aplicación (APO) controlan y administran las capas del protocolo en un dispositivo ZigBee. La definición clave de ZigBee es el objeto de dispositivo ZigBee, que aborda tres operaciones principales, descubrimiento de servicios, seguridad y vinculación. La función de descubrimiento es encontrar nodos y preguntar por la dirección MAC del enrutador mediante mensajes unicast. El descubrimiento de servicios es facilitar el procedimiento para localizar algunos servicios usando sus identificadores de perfil. Los servicios de seguridad en un objeto de dispositivo ZigBee tiene la función de autenticar y derivar las claves necesarias para la codificación de datos. La subcapa de soporte de aplicación proporciona una interfaz entre las capas de red y aplicación a través de varios servicios generales brindados por datos APS y entidades de administración. [18]

### 6.3.2. Z-Wave

Z-Wave es un protocolo de red diseñado como medio de comunicación inalámbrico para tecnología utilizada en casas residenciales. La banda que utiliza esta por debajo de 1GHz, evitando las frecuencias 2.4 GHz y 5GHz extremadamente congestionadas por tecnologías como wifi y ZigBee. Permite comunicación segura y confiable, en dos vías, utilizando confirmación de recepción en los mensajes y redes malladas. Ofrece un precio razonable, el cual es más alto que las tecnologías analógicas de gama baja, pero considerablemente más bajo que tecnologías de gama alta como EnOcean, diseñadas para el mercado profesional de la construcción.

Este protocolo ofrece completa interoperabilidad, como característica principal. Todos los dispositivos que se comuniquen por medio de Z-Wave pueden funcionar juntos en una sola red y pueden ser manejados por cualquier controlador que también utilice Z-Wave. Como una mejora al protocolo, la alianza Z-Wave desarrollo Z-Wave plus. La cual ofrece, una definición precisa de los roles de cada dispositivo en una red Z-Wave, una definición precisa de los posibles dispositivos Z-Wave, extensión del programa de certificación para cubrir manuales, empaquetado y el uso de logos y que sus dispositivos ahora tendrán una comunicación mas rápida, una confiabilidad inalámbrica más alta y utilizarán baterías con bajo consumo de potencia. [19]

Z-Wave opera en la banda de radio frecuencia industrial, científica y médica (ISM). En Europa transmite en las frecuencias de 868.42 MHz y en Estados Unidos en las frecuencias de 908.42 MHz, diseñadas para comunicaciones de datos con bajo ancho de banda para dispositivos de domótica. El protocolo se compone de cuatro capas, la capa física, la capa de transporte, la capa de red y la capa de aplicación.

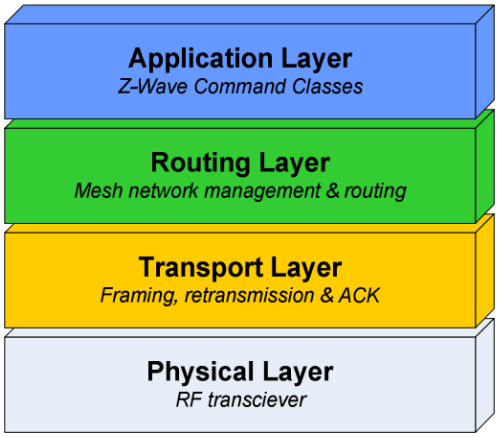


Figura 6: Capas del protocolo Z-Wave.

La capa física es la que interactúa directamente con el hardware del transmisor RF de Z-Wave. Para implementar la capa física Z-Wave es necesario analizar las recomendaciones G.9959 del ITU-T, que contienen las especificaciones de las capaz físicas y de control de acceso al medio de comunicaciones de radio sub GHz, como el protocolo Z-Wave. Un ejemplo de dispositivo para esta implementación puede ser el transmisor CC11100, capaz de recibir y transmitir paquetes Z-Wave con velocidades de transmisión de 9.6 Kbps y 40 Kbps.

La capa de transporte es la principal responsable de la retransmisión, confirmación de recepción de paquetes, despertar nodos de red y la autenticación del origen de los paquetes. Cada trama Z-Wave en esta capa contiene el identificador de origen de 32 bits que tiene asociada una red Z-Wave, 8 bits de origen, 8 bits de encabezado de trama que define el tipo de trama y la bandera de control para indicar aspectos como una transmisión de baja potencia, 8 bits de longitud de carga útil seguido de la carga útil y el valor de suma de comprobación de la trama de 8 bits. La capa de transporte se basa en esta suma de comprobación para detectar y descartar paquetes erróneos.La retransmisión de tramas sucede cuando no se ha recibido una trama de confirmación del receptor antes de que se acabe el tiempo de expiración. Esta capa también controla las tramas de haces que son usadas para despertar los nodos Z-Wave alimentados por batería. Para preservar la batería, el dispositivo entra en reposo y periódicamente enciende su radio en busca de tramas de haces. El nodo transmisor envía tramas de este tipo cada 100 ms para garantizar que el dispositivo que esta dormido reciba alguna de estas tramas para despertar y así mantendrá su radio encendida para recibir las siguientes transmisiones.

En la capa de red del protocolo Z-Wave se crea una red en forma de malla con un controlador principal y hasta 232 nodos que pueden actuar como repetidores de paquetes, con la excepción de los nodos de alimentación. Esto permite enrutar paquetes aunque las dos partes que se están comunicando no puedan establecer un radio de enlace directo entre ellas. Para determinar la mejor ruta hacia un nodo de destino, cada dispositivo en la red Z-Wave almacena la topología de la red que le indica los dispositivos cercanos. Cuando un dispositivo cambia de ubicación o es removido de la red, esta topología puede ser errónea y provocar problemas de enrutamiento. Para evitar esto, el protocolo es capaz de descubrir automáticamente la nueva topología, para calcular las nuevas tablas de ruta y actualizar

todas las tablas de enrutamiento.

La capa de aplicación es la responsable de analizar la carga útil de la trama y decodificar los parámetros y comandos soportados por el protocolo. Si el nodo es un dispositivo Z-Wave de control, el comando decodificado y el parámetro asociado son reenviados al controlador de software que esta corriendo en la computadora de control o aplicación. La carga útil comienza con un encabezado de comando de un byte que especifica si el comando es unicast, multicast o broadcast seguido de la clase del comando. Las clases de los comandos Z-Wave definen la funcionalidad del dispositivo, cada clase de comando puede estar compuesta de varios comandos. La carga útil esta encriptada y se encuentra seguida de 8 bits de autenticación. [20]

### 6.3.3. Bluetooth

La tecnología inalámbrica Bluetooth fue diseñada como una solución de conductividad de bajo rango para dispositivos electrónicos personales, portátiles y de mano. Un radio Bluetooth opera en la frecuencia sin licencia de 2.4 GHz. El protocolo está agrupado en dos categorías, los protocolos de transporte y los protocolos de software intermedio. Los protocolos de transporte comprenden protocolos desarrollados exclusivamente para la tecnología inalámbrica Bluetooth. Estos protocolos están involucrados en todas las comunicaciones entre dos dispositivos Bluetooth. Los protocolos de software intermedio comprenden los protocolos exclusivos y otros protocolos añadidos. Estos protocolos son usados selectivamente para habilitar diferentes aplicaciones, incluyendo las aplicaciones antiguas y nuevas, para intercambiar datos utilizando Bluetooth. Cuando es necesario, los protocolos de software intermedio protegen estas aplicaciones de las especificaciones de los protocolos de transporte Bluetooth.

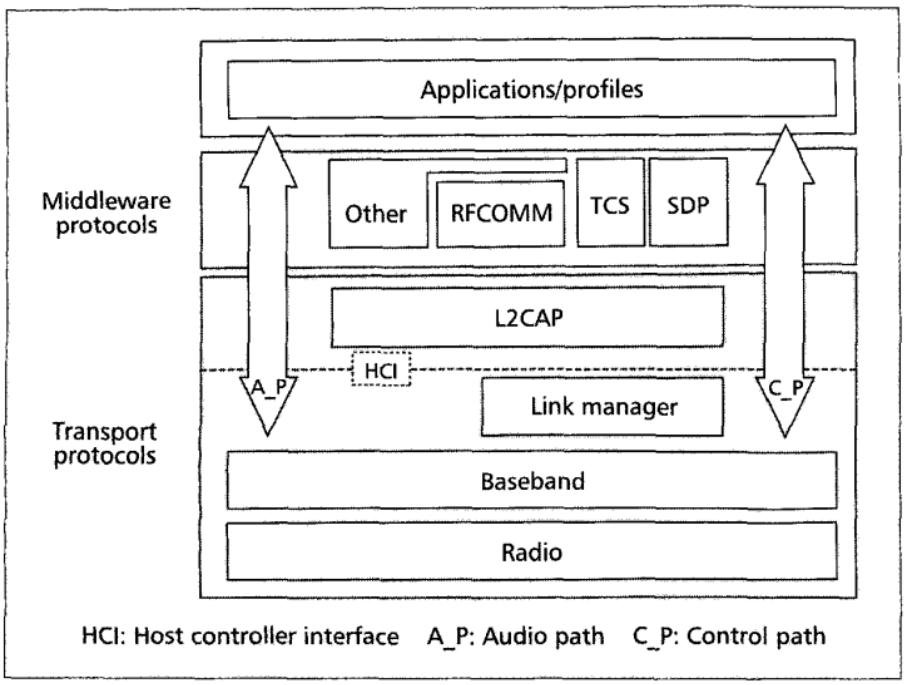


Figura 7: Capas del protocol Bluetooth.

La capa de radio define las características técnicas de los radios Bluetooth. Estos radios operan en la banda sin licencia ISM de 2.4 GHz y cumplen con la parte 15 de la normativa FCC. Utiliza técnica de salto de frecuencia y espectro ensanchado (FHSS), para lograr una velocidad de 1600 saltos por segundo. La técnica de modulación que utiliza es salto binario de frecuencia Gaussiano (GFSK), con una velocidad de transmisión de 1 Msymbol por segundo. Los radios Bluetooth se dividen en tres clases según la potencia, los de clase 1 capaces de transmitir con una potencia de 100 mW, los de clase 2 capaces de transmitir con una potencia de 2.5 mW y los de clase 3 capaces de transmitir a una potencia de 1 mW. Acorde con las restricciones de potencia y de costo, para los dispositivos personales que utilizan radios Bluetooth, las clases 2 y 3 son las más utilizadas.

Cada dispositivo Bluetooth tiene dos parámetros que intervienen en todos los aspectos de este tipo de comunicación. El primero es la dirección única de tipo IEEE de 48 bits, asignada cada relación Bluetooth en el momento de fabricación. La dirección de dispositivo Bluetooth esta grabada en el hardware del dispositivo y no puede ser modificada. El segundo parámetro es el reloj de 28 bits con pulsos cada 312.5 us, que corresponde a la mitad del tiempo de permanencia en una frecuencia cuando el radio salta a la velocidad nominal de 1.600 saltos por segundo. Los dispositivos Bluetooth se pueden comunicar entre sí, al momento de adquirir las direcciones y el relojes de los demás dispositivos.

El piconet Bluetooth es un conjunto de dispositivos Bluetooth que se pueden comunicar entre sí. Un piconet se forma de manera espontánea, sin ayuda de infraestructuras, y dura el tiempo que su creador necesite y esté disponible para comunicarse con otros dispositivos. Contiene al menos un dispositivo maestro y siete dispositivo esclavos como máximo, con los que el maestro está involucrado activamente en sus comunicaciones. Las definiciones de maestro y esclavo están asociadas a un piconet en particular, estos términos no están asigna-

dos a las unidades del radio en el momento de manufactura, por lo que un radio Bluetooth puede actuar como maestro y como esclavo en momentos diferentes. Para identificar cada esclavo, el maestro del piconet le asigna localmente una dirección de mimbro activo única. El maestro regula y controla quien transmite y cuando. Mientras tanto, hasta siete dispositivos pueden estar comunicándose activamente con dispositivos adicionales, llamados dispositivos estacionados, que pueden registrarse con el maestro y volverse activos cuando sea necesario. Cuando un dispositivo no esta asociado con ningún piconet, está en espera. Los piconets pueden coexistir en tiempo y espacio de manera independiente, además, un dispositivo puede ser miembro de varios piconets, si se refiere a una red dispersa en el protocolo Bluetooth.

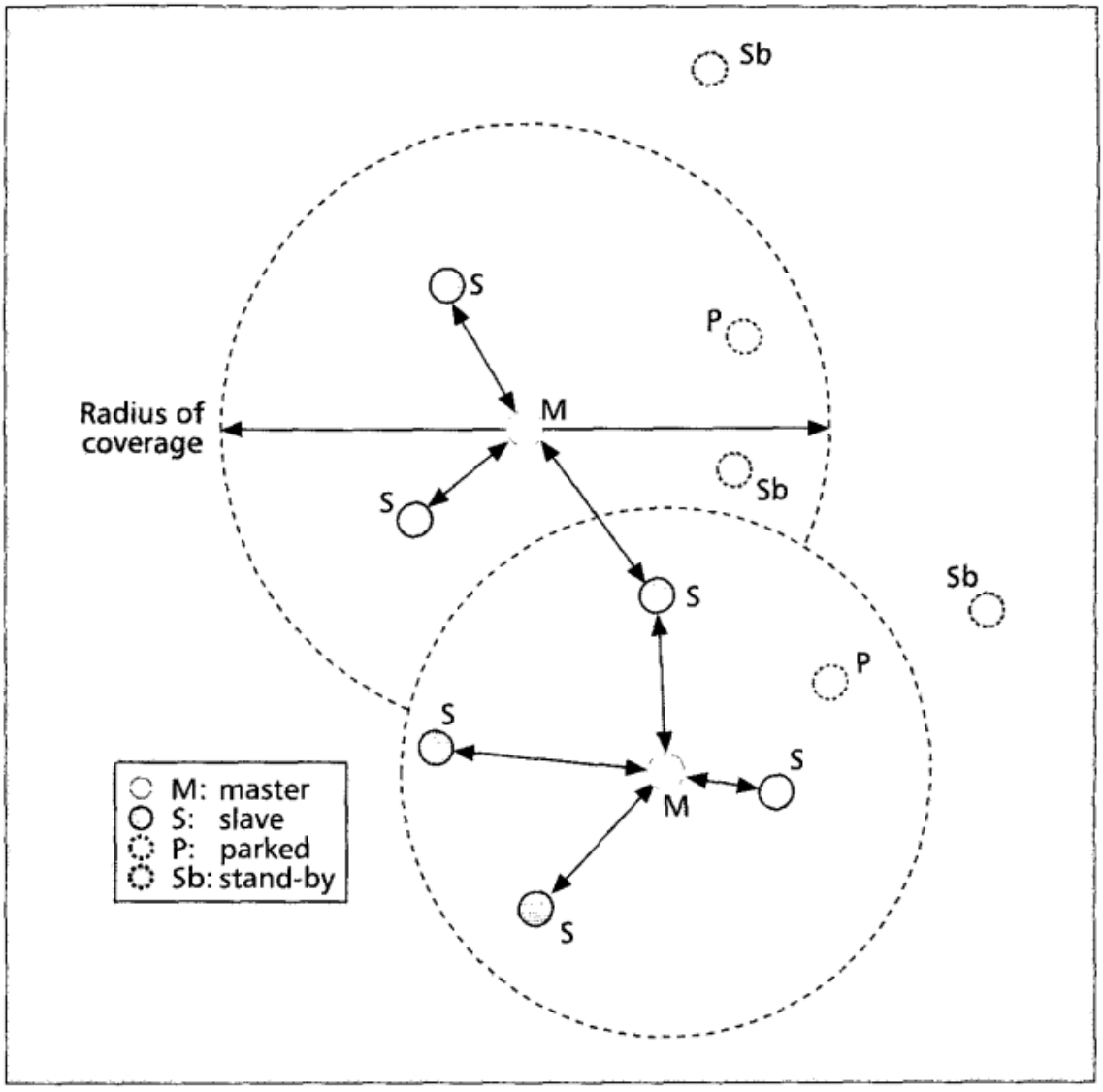


Figura 8: Piconets Bluetooth.

Existen dos tipos de enlaces soportados en el piconet Bluetooth. Se tienen un único enlace asíncrono soportado (ACL) y opcionalmente un piconet puede soportar como máximo tres enlaces síncronos orientados a la conexión (SCO). El ACL es un enlace de mejor esfuerzo

apropiado para transmisiones de datos asíncronas. Este mantiene la integridad utilizando retransmisión y secuencia de miembros, también reenvía la corrección del error (FEC) si es necesario. El enlace SCO soporta transmisiones periódicas de audio de 64 Kb/s en cada dirección. El tráfico de SCO no es retransmitido, pero puede usar el mecanismo FEC para cubrir errores en la transmisión cuando sucedan.

El protocolo de administración de enlaces (LMP) es un protocolo transaccional entre dos entidades de administración en dispositivos Bluetooth en comunicación. Su responsabilidad es configurar las propiedades del enlace Bluetooth. A través de las transacciones LMP un dispositivo puede autenticar otro dispositivo a través de un mecanismo de respuesta. Para dispositivos autenticados, el enlace puede estar encriptado. Dos administradores de enlaces pueden aprender de las características del otro. Las conexiones SCO son establecidas utilizando transacciones LMP, así también se establecen los intervalos y los tamaños de los paquetes. [21]

### 6.3.4. MQTT

El protocolo de transporte de telemetría de colas de mensajes (MQTT) es un protocolo utilizado en un entorno de IoT, que funciona sobre el protocolo de control de transporte [22]. El protocolo fue creado por IBM como un método de comunicación ligero entre maquinas. MQTT fue estandarizado por ISO/IEC 20922 y fue aceptado como parte de la Organización para el Avance de las Normas de Información Estructurada (OASIS). En esencia, MQTT es un protocolo de mensajería que utiliza el modelo de comunicación *publish-subscribe*, donde los clientes no requieren actualizaciones, generando una reducción considerable en el uso de recursos y siendo ideal para implementarse en un entorno con bajo ancho de banda.

La arquitectura del protocolo consta de dos partes, el cliente y el *broker*. Cada mensaje MQTT contiene un tema, organizado en una estructura en forma de árbol, al que los clientes pueden suscribirse o publicar. El *broker* recibe los mensajes publicados de los clientes que contienen un determinado valor o comando y transmite la información a todos los clientes que se han suscrito a ese tema específico. [23]

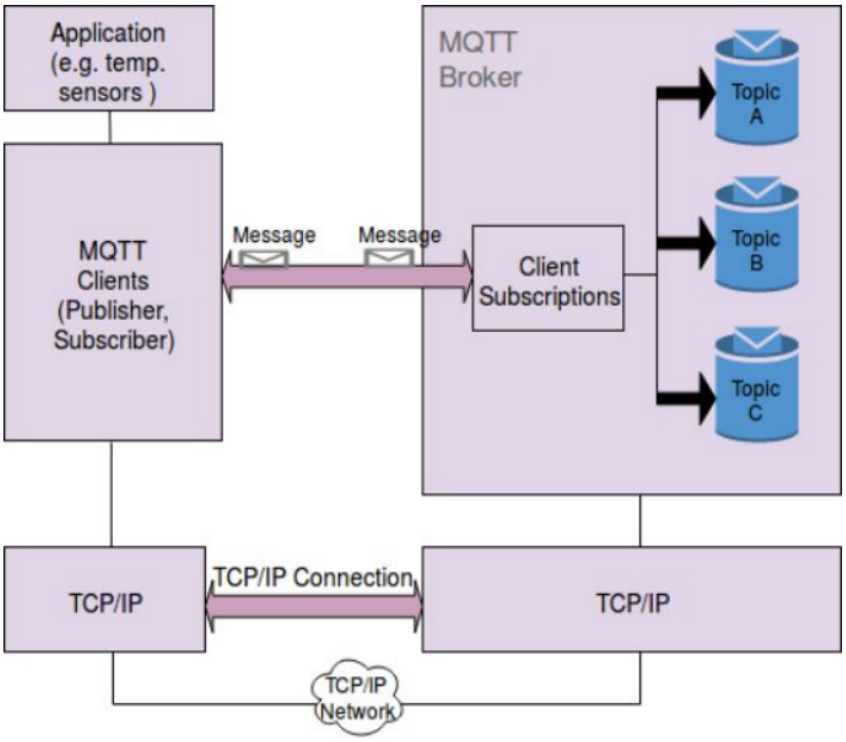


Figura 9: Arquitectura del protocolo MQTT.

El cliente puede funcionar como publicador o como suscriptor y siempre es el que establece la conexión con el*broker*. El cliente puede publicar mensajes para usuarios interesados, suscribirse a un tema específico para recibir mensajes, darse de baja en alguna suscripción para dejar de recibir mensajes de ese tema y desvincularse del *broker*.

El *broker* controla la distribución de la información y es el principal responsable de la recepción de todos los mensajes de los publicadores. Así mismo, los filtra y decide que cliente esta interesado en cada mensaje y lo envía a todos los clientes suscritos en ese tema. El *broker* puede aceptar requerimientos de clientes, recibir mensajes publicados por los usuarios, procesar diferentes requerimientos como requerimientos de suscripción y requerimientos para darse de baja de los clientes y enviar a cada cliente interesado los mensajes que correspondan al tema al que está suscrito. [24]

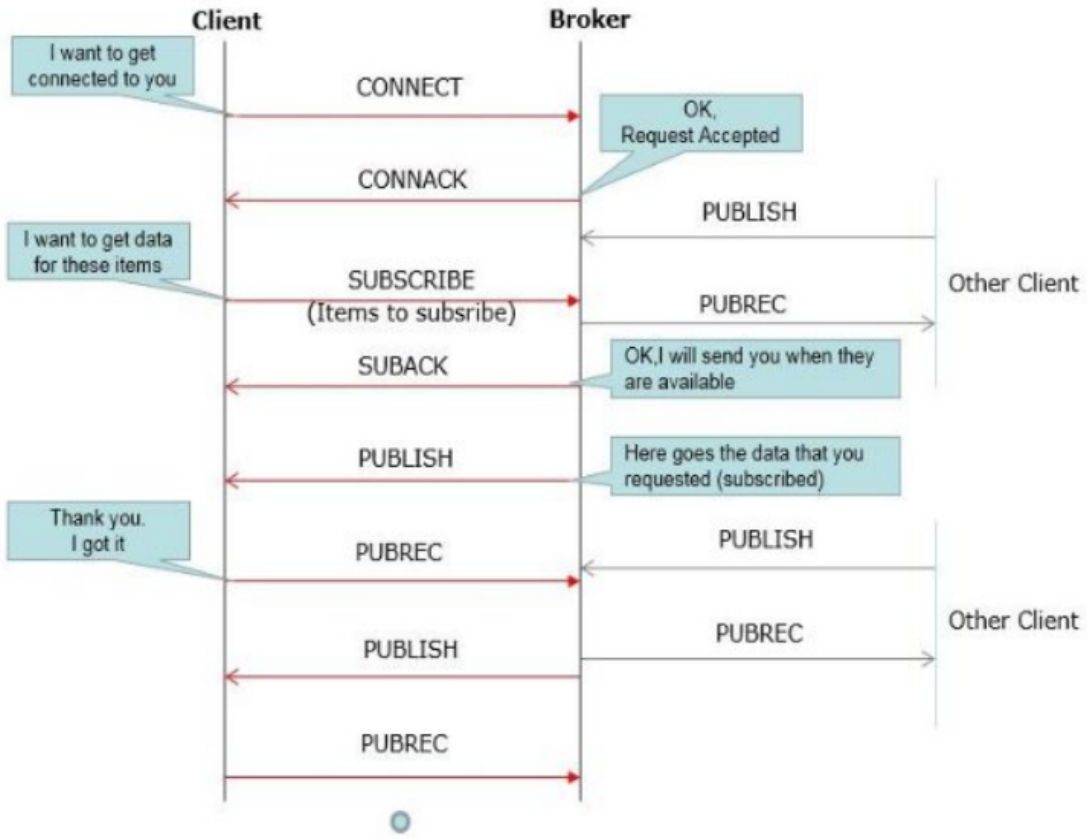


Figura 10: Comuniación utilizando MQTT.

### 6.3.5. wifi

El protocolo *Wireless Fidelity* (wifi) es una tecnología de área local diseñada para casas y espacios de implementación pequeños. Wifi es un sistema de transmisión de datos diseñado para permitirle el acceso a la red, independientemente de su ubicación, a dispositivos informáticos, mediante ondas de radio o un medio guiado. Su objetivo principal es proveer cobertura de banda ancha dentro de edificios, basada en la norma IEEE 802.11 para comunicaciones inalámbricas de corto alcance. Un *hotspot* se utiliza para proporcionar la conectividad entre diferentes dispositivos para la transferencia de datos, siendo este la región cubierta por uno o varios puntos de acceso (AP).

Un punto de acceso inalámbrico proporciona conectividad a un grupo de dispositivos inalámbricos con una red de área local (LAN) cableada, para comunicación entre dispositivos inalámbricos con un único dispositivo conectado por cable. Su arquitectura consiste en una serie de APs y varios clientes. Un cliente esta directamente conectado a un AP. El AP se comunica con el cliente transmitiendo su identificador de conjunto de servicios (SSID) o nombre de red a través de paquetes, lo que se conoce como balizas. El AP transmite sus balizas cada 100 ms con una velocidad de datos de 1 Mbps. La conectividad del cliente depende estrictamente de la configuración del SSID, si esta no es buena, puede que no se de la conexión. Si hay varios APs con el mismo SSID, el firmware del cliente utiliza la intensidad

de la señal como criterio para determinar a cual AP conectarse.

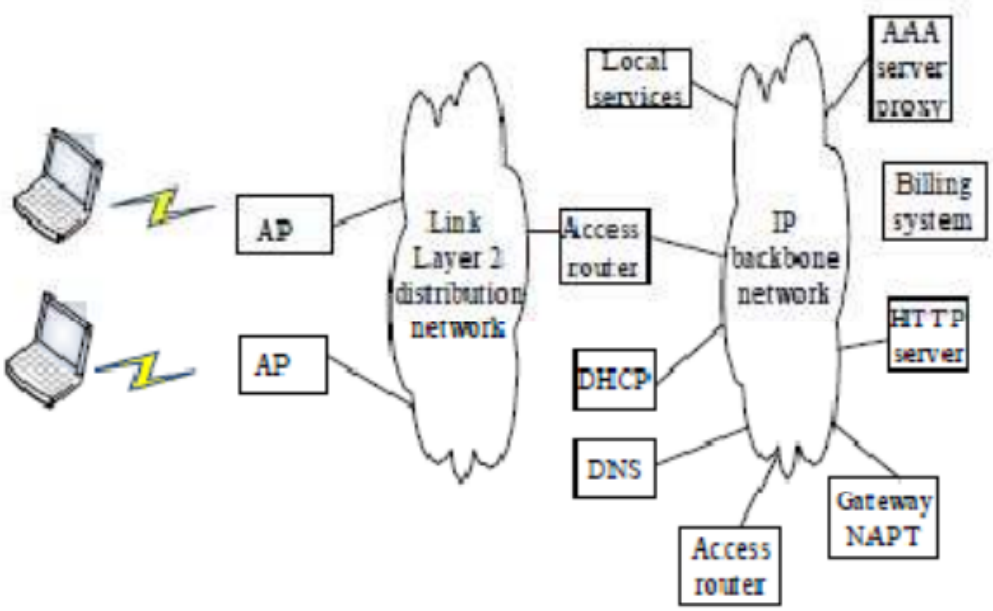


Figura 11: Aqruitectura del protocolo wifi.

Las redes wifi usan señales de radio para dar conectividad a Internet, a la red del operador móvil o para uso de servicios de internet. Esta provee servicio únicamente para el nivel de capa de enlace y depende de la infraestructura de capa 3 para la conectividad de extremo a extremo. En la Figura 11 se observa que la puerta de enlace NAPT proporciona conectividad a otras redes de capa 2, mientras que el servidor proxy AAA se encarga del control de acceso y de la autenticación de las terminales portátiles. wifi utiliza el protocolo RADIUS (*Remote Authentication Dial In User Service*) junto con el protocolo EAP (*Extensible Authentication Protocol*) para autenticar una terminal que intenta acceder a la red. Cada AP en una red wifi tiene un alcance limitado para que el cliente se conecte. La distancia real depende del entorno, es decir, si el cliente se encuentra en interiores o exteriores. El alcance usuario para interiores es de 45 - 90 metros y el alcance en exteriores es de 300 metros. Por naturaleza, el wifi no fue diseñado para tener soporte para QoS (calidad de servicio). Se diseñó para los servicios de datos de mejor esfuerzo. También existe un estándar IEEE 802.11e ampliado para aplicaciones multimedia sensibles al retardo, como voz sobre IP.[25]

### 6.3.6. HTTP

El protocolo HTTP surge en 1990 con su primera versión, la 0.9. En ese momento el protocolo únicamente consistía en una simple conexión TCP con un método *GET* para adquirir documentos con hipertextos, como hiperenlaces, referencias y otros textos accesibles directamente de un servidor HTTP dada su respectiva dirección. El protocolo es sin estado, lo que significa que cada requerimiento se ejecuta de manera independiente a los comandos que se ejecutaron antes. Esta versión del protocolo no tenía encabezados ni metadatos. Además,

era un protocolo ASCII, de modo que los requerimientos eran en texto plano, entendible por humanos.

La versión 1.1 surge como una mejora al protocolo HTTP incorporando algunos cambios significativos. Esta actualización de caracterizó por limitar a una sola solicitud por conexión TCP, lo que provoco un problema un problema conocido como "*head of line blocking*". Esto consiste en que cuando se envía un archivo pesado como primer documento, los siguientes no podrían enviarse hasta que el primer documentos terminara de cargarse. La versión 2, desarrollada el grupo de trabajo de la IETF, se baso en el protocolo de Google SPDY. Esta nueva mejora no viene como remplazo de la versión 1.1, sino que es una capa de protocolo que funciona debajo del HTTP/1.1 y no encima de la capa de transporte. Además, este protocolo es binario, lo que implica que las solicitudes y respuestas ya no son entendibles por humanos. Sin embargo, esta mejora le permite a las maquinas analizar los mensajes de manera más eficiente. [26]

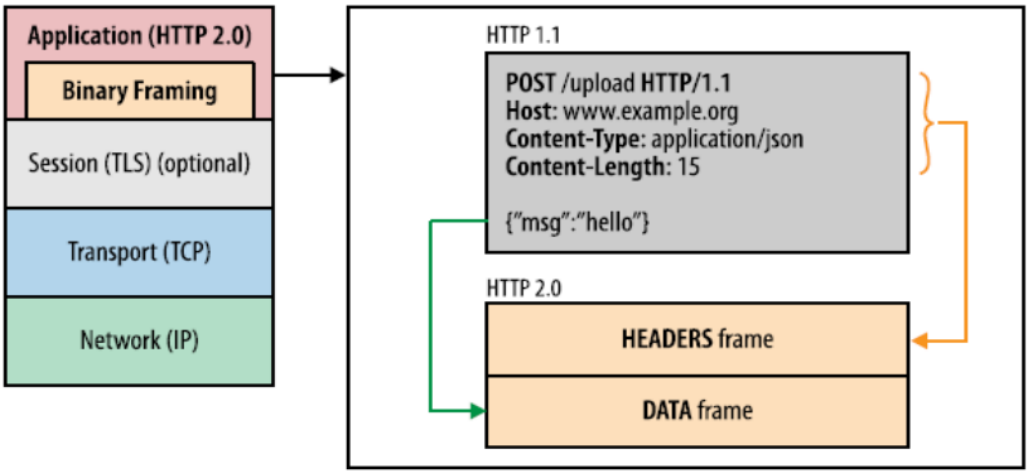


Figura 12: Capa de enmarcado binario del HTTP/2.

El protocolo HTTP es un protocolo de tipo *request/response*. Un cliente envía un requerimiento al servidor en la forma de un método de solicitud, URI, versión del protocolo, seguido del mensaje MIME-like que contiene los modificadores de la solicitud, la información del cliente e información adicional referente a la conexión con el servidor. El servidor responde con una linea de estatus, incluyendo la versión del protocolo del mensaje y el código de éxito o error, seguido del mensaje MIME-like con la información del servidor, la metainformación de la identidad y posiblemente contenido adicional sobre la identidad. La mayoría de comunicaciones HTTP son iniciadas por el usuario y consisten en una solicitud para ser aplicada a un recurso en algún servidor de origen.

El mensaje de solicitud de un cliente a un servidor incluye, en la primera línea del mensaje, el método que se aplicará al recurso, el identificador del recurso, y la versión del protocolo en uso. La linea de solicitud comienza con el *token* del método, seguido del *Request-URI* y la versión del protocolo, terminando con CLRF. Los elementos están separados por caracteres SP (Espacio Simple). No se permiten CR ni LF excepto en la secuencia final

CRLF. El *token* del método indica el método que va a ser implementado en el recurso identificado por el *Request-URI*. Los métodos disponibles son OPTIONS, GET, HEAD, POST, PUT, DELETE y TRACE, exactamente como están escritos ya que el método es sensible a mayúsculas y minúsculas. El *Request-URI* es un identificador de recurso uniforme e identifica el recurso sobre el que se va a aplicar la solicitud. Para este se tienen tres tipos "*", absoluteURI y abs_path. El tipo que se utiliza depende de la naturaleza de la solicitud, el asterisco significa que el requerimiento no aplica a un recurso particular, el de tipo absoluteURI es necesario cuando la solicitud esta siendo hecha a un proxy y el abs_path se usa para identificar el recurso en un servidor de origen o una puerta de enlace.

Los servidores de origen HTTP/1.1 deberían ser conscientes de que el recurso exacto identificado por una solicitud de Internet se determina examinando tanto el campo Request-URI y el campo de cabecera *Host*. Un servidor de origen que no permite que los recursos se diferencien por el *Host* solicitado puede ignorar el valor del campo *Host*. Los campos de *request-header* permiten a los clientes pasar información adicional sobre la solicitud o sobre el mismo cliente para el servidor. Estos campos funcionan como modificadores de requerimiento con semánticas equivalentes a los parámetros en la instanciación de un método en un lenguaje de programación. [27]

## 6.4. Herramientas de IoT diseñadas para la *Raspberry Pi*

### 6.4.1. *WebThings*

*WebThings* [5] es una implementación de código abierto de *Web of Things*, incluyendo el *WebThings gateway*, *WebThings framework* y *WebThings Cloud*. Específicamente para el *gateway*, *WebThings* ofrece una imagen de sistema operativo para la *Raspberry Pi*. Dándole al usuario una interfaz basada en web para supervisar y controlar dispositivos inteligentes, un motor de reglas para automatizarlos y un sistema complementario para ampliar la compatibilidad con una gran variedad de dispositivos inteligentes domésticos existentes, como dispositivos ZigBee, Z-Wave, Bluetooth, HomeKit y WEAVE. También, permite colocar todos los dispositivos en un plano interactivo de la casa del usuario para tener el estado y el control en un vistazo.

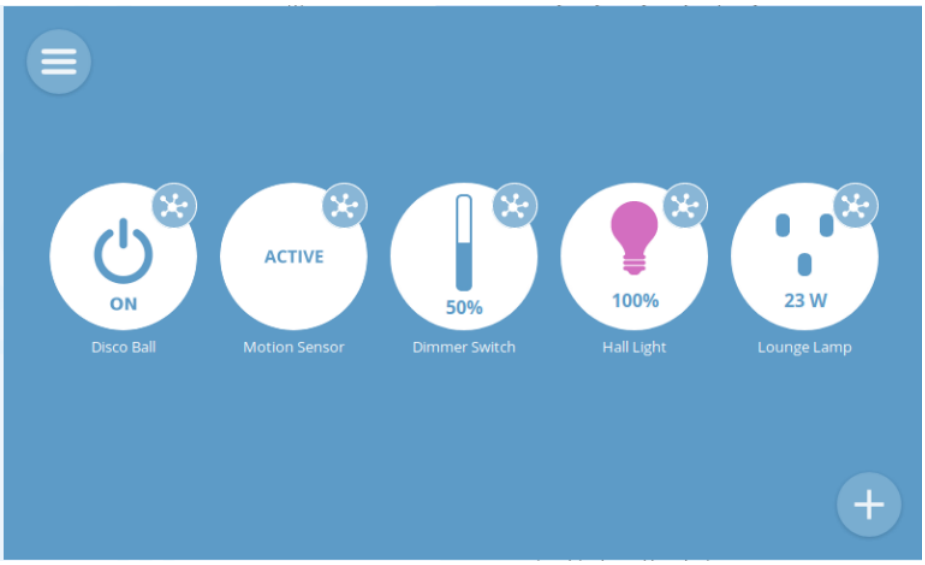


Figura 13: Ejemplo de la interfaz web.

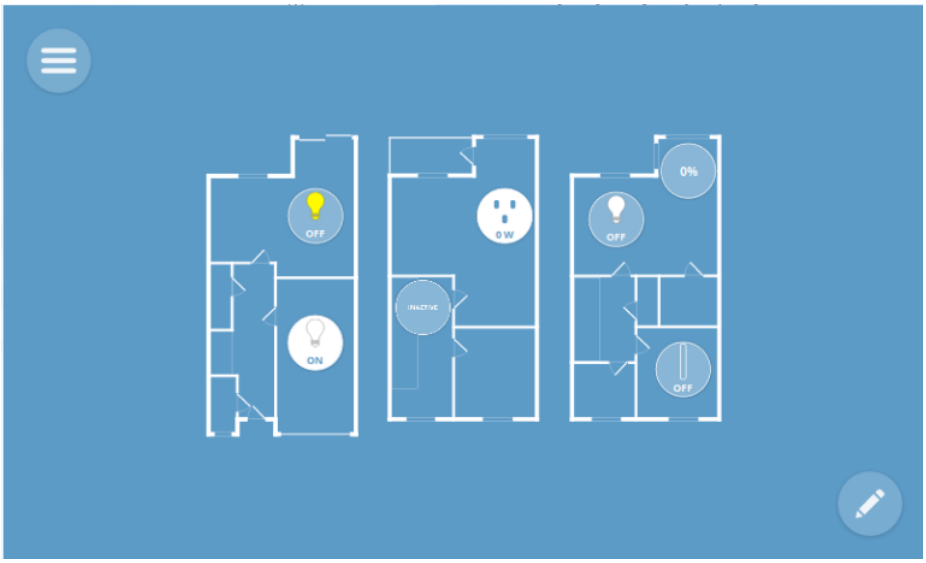


Figura 14: Ejemplo del plano interactivo.

### 6.4.2. *Pycroft*

*Picroft* [6] es una manera de ejecutar *Mycroft* en una *Raspberry Pi 3, 3B+ o 4*, mediante la instalación de una imagen de sistema operativo en la microSD de la *Raspberry Pi*. Con esto se logra convertir la *Raspberry Pi* en una asistente virtual como *Amazon Alexa*, *Google Assistant*, *Microsoft Cortana* y *Apple's Siri*. La ventaja que presenta *Mycroft* a diferencia de los asistentes mencionados es que es de código abierto, lo que le permite a los usuarios inspeccionar, copiar, modificar y contribuir a la comunidad para que todos se beneficien con eso.

### 6.4.3. *HomeAssistant*

*HomeAssistant* [28] es un software de código abierto para implementaciones de domótica. Este prioriza el control local y la privacidad, siendo impulsado por una comunidad mundial de desarrolladores y entusiastas del *DIY*. Este sofware puede cargarse en diferentes dispositivos, sin embargo, recomiendan su implementación mediante la instalación de su sistema operativo en una *Raspberry Pi* 3 o 4 en cualquiera de sus versiones. Este sistema ofrece varias integraciones como, amazon alexa, ecobee, ESPhome, Google Assistant, Google Cast, MQTT, SmartThings, dispositvos Z-Wave , dispositivos ZigBee entre otros. Cada una de estas integraciones puede configurarse de manera sencilla desde la interfaz web, permitiéndole al usuario implementar sistemas completos, sin la necesidad de aplicar conocimientos profundos del área de domótica. Para interactuar con los dispositivos del sistema, *HomeAssistant* permite la creación de diferentes *Dashboards*, en los que se pueden colocar diferentes tipos de *widgets* para la visualización y control de los dispositivos del sistema.

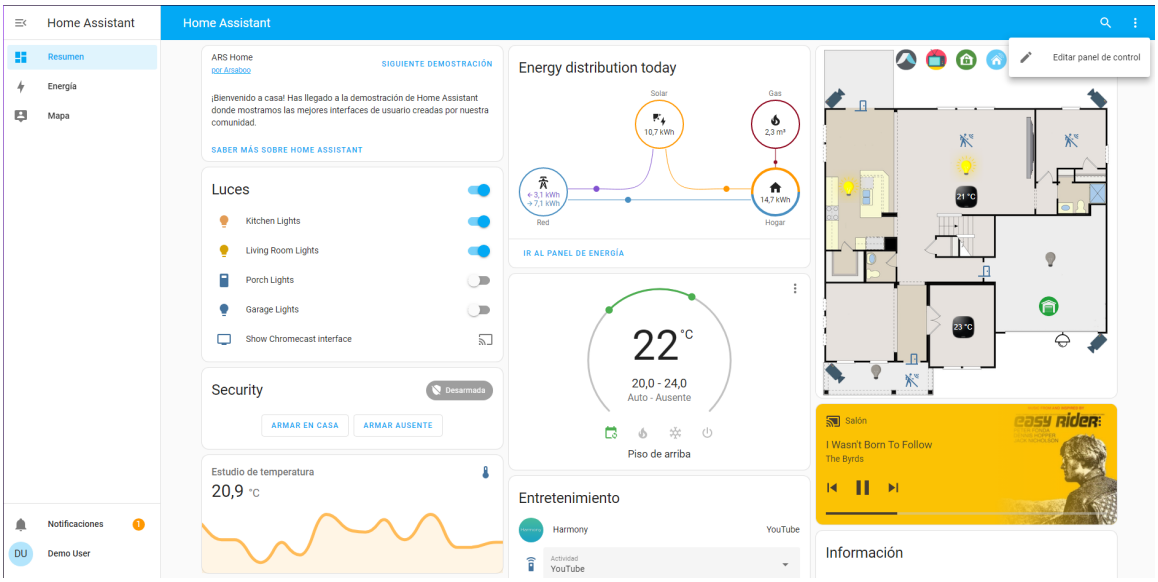


Figura 15: Ejemplo de un *Dashboard* realizado con *HomeAssistant*.

## 6.5. Plataformas de código abierto de IoT compatibles con la *Raspberry Pi*

### 6.5.1. *ThingsBoards*

*ThingsBoard* [29] es una plataforma de IoT de código abierto, que permite el rápido desarrollo, administración y escalamiento de proyectos de IoT. Su objetivo es es proporcionar una solución lista para usar en la nube o en las instalaciones que habilitará la infraestructura del lado del servidor para aplicaciones de IoT. Con esta plataforma se pueden disponer de dispositivos, activos y consumidores y definir relaciones entre ellos, se puede colectar y visualizar los datos obtenidos de los activos y dispositivos, se pueden analizar los datos entrantes y accionar alarmas con procesamiento de eventos complejos, se pueden controlar

los dispositivos utilizando llamadas a procedimientos a distancia (RPC), se pueden crear flujos de trabajo basados en un evento *life-cycle* del dispositivo, un evento de la API REST, una solicitud RPC, etc, se pueden diseñar *dashboards* dinámicos con capacidad de respuesta para presentar la telemetría de los dispositivos o activos, se pueden habilitar características especiales utilizando una cadena de reglas configurable, enviar datos de un dispositivo a otros sistemas, entre otras.

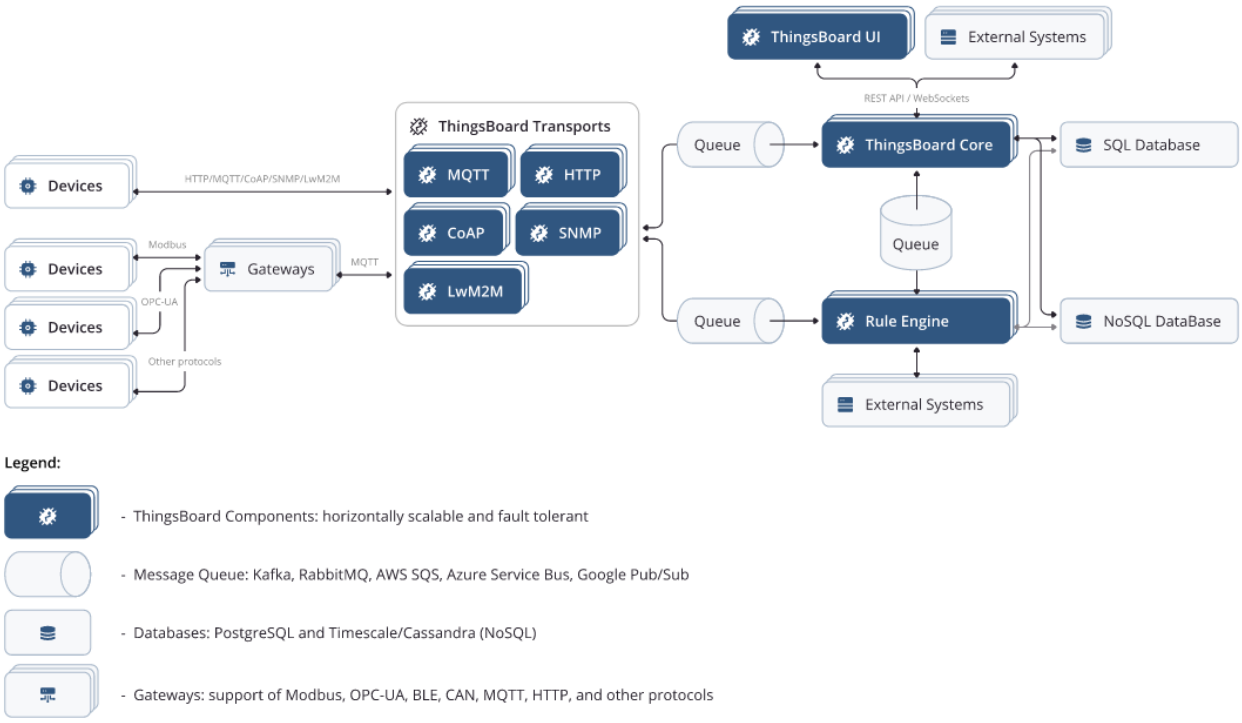


Figura 16: Esquema de un sistema con *ThingsBoard*.

### 6.5.2. *Thinger.io*

*Thinger.io* [30] es una plataforma de IoT en la nube que brinda cualquier herramienta necesaria para prototipar, escalar y administrar productos conectados de una manera simple. Su objetivo es democratizar el uso de IoT haciéndolo accesible para para todo el mundo, y agilizar el desarrollo de grandes proyectos de IoT. Esta plataforma ofrece una licencia *free-miun* para comenzar a diseñar y prototipar, que se puede escalar a *premiun* con capacidades completas de manera rápida, una manera simple de conectar dispositivos para comenzar a recibir datos de el o controlarlo, permitiendo administrar cientos de dispositivos de manera simple, ofrece compatibilidad completa con cualquier dispositivo de cualquier fabricante que tenga capacidades de conectividad y una estructura eficiente y extremadamente escalable, que se basa en un servidor de IoT que suscribe recursos de los dispositivos para recopilar datos sólo cuando es necesario, de modo que una única instancia de la plataforma es capaz de gestionar miles de dispositivos de IoT con baja carga computacional, ancho de banda y latencia.

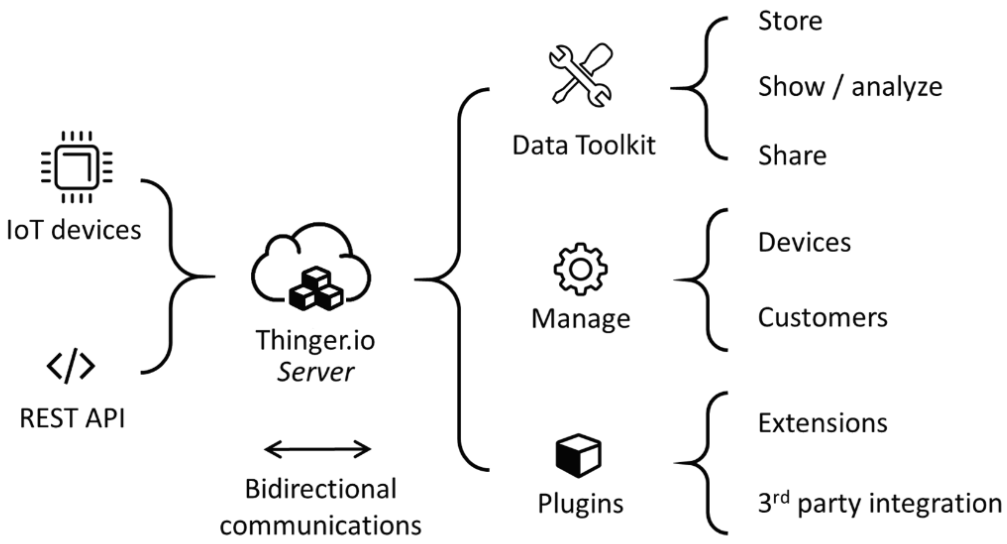


Figura 17: Esquema de características que ofrece *Thinger.io*.

### 6.5.3. *Mainflux*

*Mainflux* [31] es una plataforma moderna, escalable, segura, de código abierto y sin patente de IoT en la nube escrita en *Go*. Acepta conexiones entre usuarios y dispositivos a través de varios protocolos de red como HTTP, MQTT, WebSocket y CoAP. Esta plataforma es utilizada como *middleware* para aplicaciones de IoT complejas. Ofrece la creación de puentes de comunicación a través de protocolos como HTTP, MQTT, WebSocket y CoAP, permite la administración y aprovisionamiento de dispositivos, ofrece control de acceso detallado, ofrece inicio de sesión y soporte para la plataforma y su despliegue es basado en contenedores mediante Docker. La plataforma fue desarrollada al rededor de tres elementos, usuarios, objetos y canales. Los usuarios representan a los usuarios humanos, se identifican con el correo electrónico y contraseña que utilizan como credenciales de acceso en la plataforma para obtener fichas de acceso. Los objetos representan los dispositivos o aplicaciones conectados a la plataforma, que la usan para intercambiar mensajes con otros objetos. Y los canales, que representan los canales de comunicación. Este elemento se compone de mensajes sobre algún tema que puede ser consumido por todos los objetos que estén conectados con el.

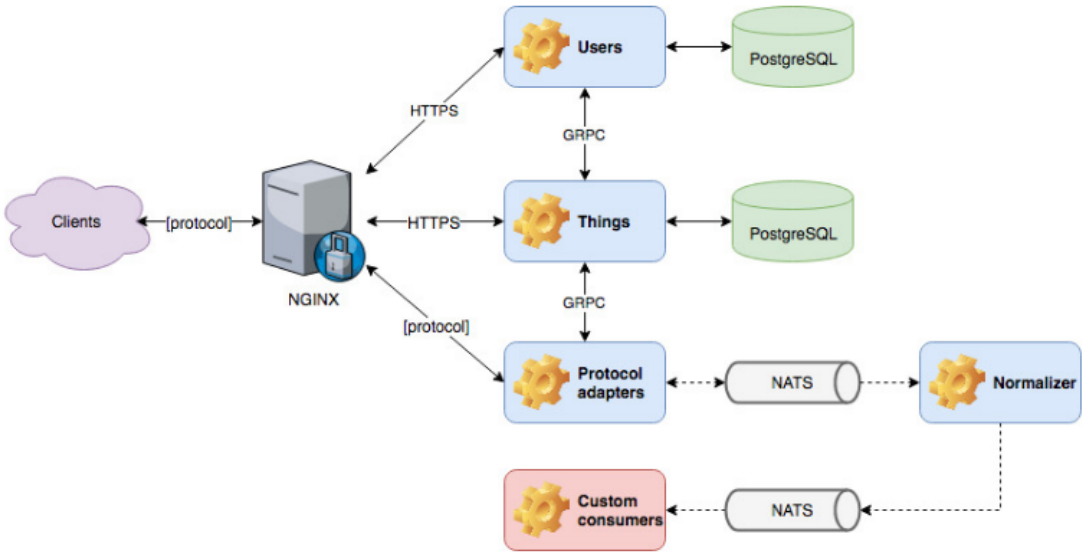


Figura 18: Esquema de un sistema con *Mainflux*.

### 6.5.4. *OpenRemote*

*OpenRemote* [32] es una plataforma de IoT intuitiva, amigable con el usuario y de código abierto para creadores de dispositivos, integradores y gobiernos. Incluye todas las funciones y características para interconectar dispositivos para construir aplicaciones simples y aplicaciones inteligentes de dominio específico. Las principales características de la plataforma son las siguientes. El Administrador de activos, es la interfaz mediante la que se crean, conectan y administran activos. Los tipos de activos pueden ser totalmente personalizados, visualizados en mapas guardados o utilizados para crear reglas y *dashborads*.

Servicios localizados, se pueden visualizar de manera geográfica cada uno de los activos junto con sus valores de actuales. Así también, se puede modificar el mapa para visualizar únicamente el área de interés y utilizar vallas geográficas para accionar alarmas o enviar notificaciones. Agentes de protocolos, los agentes de protocolo conectan los dispositivos y servicios externos. Los protocolos genéricos como HTTP, SNMP, MQTT, Bluetooth, Serial, TCP, UDP, Z-Wave estan incluidos, asi como también, los protocolos específicos de algún proveedor, como KNX y Velbus. Administrador de APIs, el administrador de APIs permite que dispositivos externos se conecten a la plataforma como clientes. *OpenRemote* incluye un *broker* MQTT, HTTP y un *Websocket* API.

### 6.5.5. *PlatyPush*

*PlatyPush* [33] es una plataforma modular de código abierto diseñada para funcionar en cualquier dispositivo que pueda correr un interprete del lenguaje Python. Incluye varias integraciones como ZigBee, Z-Wave, arduino, Bluetooth, MQTT, IFTT, YouTube, alexa, GitHub, Dropbox, entre otras. Cada una de estas esta envuelta con una API, haciendo que

crear rutinas de automatización cuando algún evento suceda, programar acciones periódicas en los *scripts* y controlar cada uno de estos desde una interfaz web, sean tareas realmente sencillas. Esta plataforma viene con un núcleo que envía mensajes y eventos sobre Redis, y su poder proviene del alto numero de integraciones que proporciona, las cuales pueden ser seleccionadas por el usuario dependiendo de la aplicación que desee desarrollar.

Para configurar cada una de las integraciones únicamente hay que definir sus atributos en el archivo de configuración, los cuales son los mismos que los atributos de los constructores de las clases mostradas en el repositorio. Se pueden crear piezas de lógica que corran acciones configuradas cuando algún evento se accione, estas pueden ser escritas en un *script* de Python o utilizando un archivo YAML. Cada una de las integraciones se puede controlar de manera sencilla en una interfaz web. Además, se expone una API que permite controlar las integraciones desde cualquier lugar y desde cualquier *script*. Esta interfaz web funciona en cualquier dispositivo, y se puede utilizar en conjunto con la aplicación móvil para dispositivos Android.

## 6.6. Bases de datos de código abierto compatibles con la *Raspberry Pi*

### 6.6.1. *MariaDB*

*MariaDB* [34] es uno de los servidores de bases de datos más populares en el mundo. Fue desarrollada por los desarrolladores originales de MySQL y se garantizo que se mantuviera de código abierto. Posee usuarios significativos como Wikipedia, WordPress.com y Google. Este servidor convierte los datos en información estructurada en una amplia gama de aplicaciones, que van desde la banca hasta los sitios web. Fue diseñado como un sustituto mejorado de MySQL. Se utiliza por sus capacidades de escalabilidad y robustez, además de poseer un ecosistema de motores de almacenamiento, *plugins* y otras herramientas que lo hacen muy versátil para utilizarse en una variedad de aplicaciones. *MariaDB* se desarrollo como base de datos relacional que proporciona una interfaz SQL para acceder a los datos.

Este servidor mantiene altos niveles de compatibilidad con MySQL, tiene un fuerte énfasis en no romper la compatibilidad hacia atrás para sus usuario. De modo que, las actualizaciones de versiones antiguas de MySQL y las versiones más nuevas de *mariaDB* son compatibles. Ofrece un modo de compatibilidad con la sintaxis de Oracle para ejecutar aplicaciones de bases de datos de Oracle sin cambios. Para los usuarios de mongoDB ofrece varias funciones JSON para manejar datos no estructurados, tiene un tipo de datos JSON con una restricción para asegurar que el JSON es válido y el motor de almacenamiento *CONNECT* tiene un tipo de tabla JSON que incluye una potente funcionalidad para manejar datos JSON.

### 6.6.2. *PostgreeSQL*

*PostgreeSQL* [35] es un sistema de bases de datos relacional de código abierto que utiliza y amplía el lenguaje SQL, combinado con muchas características que almacenan y escalan

con seguridad las cargas de trabajo de los datos más complicadas. Tiene una estructura fiable que mantiene la integridad de los dados, gracias a la dedicación de la comunidad para ofrecer constantemente soluciones innovadores y de alto rendimiento. Este sistema de bases de datos se ejecuta en los principales sistemas operativos, es compatible con ACID desde 2001 y cuenta con potentes complementos como el extensor de bases de datos geoespaciales PostGIS.

*PostgreeSQL* tiene una lista de características con mucha relevancia, las cuales son. La capacidad para manejar datos de tipo JSON/JSONB, XML y Hstore, Garantizar la integridad de los datos, ofreciendo concurrencia y rendimiento con tipos de indexado como, *B-tree*, indexado avanzado con GiST, un sistema de administrador de requerimientos, control de concurrencia multiversión (MVCC), paralelización para leer requerimientos y elaborar los índices *B-tree* y particionamiento de las tablas. Y un alto nivel de seguridad, con distintos tipos de autenticación, un robusto sistema de control de acceso, seguridad para las filas y columnas y un mutlifactor de autenticación con certificado adicional.

### 6.6.3. *Redis*

*Redis* [36] es un *data structure store* en memoria de código abierto, que se utiliza como base de datos, cahé, agente de mensajes y motor de *streaming*. *Redis* proporciona estructuras de datos como cadenas, *hashes*, listas, *sets*, *sorted sets* con rango de requerimiento, *bitmaps*, hiperlogos, índices geoespaciales y *streams*. Incorpora replicación, *Lua scripting*, *LRU eviction*, diferentes niveles de persistencia en disco y proporciona una alta disponibilidad a través de *Redis Sentinel* y un particionamiento automático con *Redis Cluster*. Puede ejecutar operaciones atómicas como, anexar a una cadena, incrementar el valor en un *hash*, empujar un elemento a una lista, calcular la intersección, unión y diferencia de conjuntos y obtener el miembro con la clasificación más alta en un conjunto ordenado.

Para lograr el máximo rendimiento, *Redis* trabaja con un conjunto de datos en memoria. Dependiendo de su caso de uso, puede guardar los datos colocando periódicamente el conjunto de datos en el disco o anexando cada comando a un registro basado en disco. También, puede desactivar la persistencia si sólo se necesita una caché en memoria, conectada a red con muchas funciones. Admite la replicación asíncrona, con sincronización rápida sin bloqueo y reconexión automática con resincronización parcial en la división de la red. También incluye, Transacciones, *Pub/sub*, Secuencias de comandos Lua, Claves con tiempo de vida limitado, desalojo LRU para las claves, recuperación automática y puede utilizarse desde la gran mayoría de lenguajes de programación. Por último, tiene una lista de usuarios significativos donde se encuentran, Twitter GitHub, Snapchat, Craigslist y StackOverflow.

## 6.7. Herramientas de aprendizaje automático de código abierto compatibles con la *Raspberry Pi*

### 6.7.1. *TensorFlow*

*TensorFlow* [37] es una plataforma de código abierto de extremo a extremo, para aprendizaje automático. Facilita la creación de modelos de aprendizaje automático, sin tener que ser un experto en el tema. Para principiantes ofrece una API secuencial, que permite crear modelos conectando bloques. Para avanzados, ofrece una API de subclases que proporciona una interfaz *define-by-run* para la investigación avanzada. Para un nuevo modelo, se debe crear una clase y luego colocar de manera imperativa el pase de avance, siendo una manera fácil de crear capas, activaciones y bucles de entrenamiento personalizados por el usuario.

*TensorFlow* ofrece un camino directo a la producción, permitiendo el entrenamiento y la implementación de un modelo, en servidores, en dispositivos finales o en la web, independientemente de la plataforma que se utilice. Con esta plataforma se pueden crear y entrenar modelos complejos sin sacrificar la velocidad ni el rendimiento. *TensorFlow* proporciona la flexibilidad y el control con funciones como, la API funcional de keras y la API de subclases de modelos, para crear topologías complejas con un porotipado fácil, depuración rápida y una ejecusión casi inmediata. Tambien, es compatible con un ecosistema de bibliotecas de complementos y modelos para experimentar, entre los que se encuentran *Raged Tensors*, *TensorFlow Probability*, *Tensor2Tensor* y BERT. Además de eso, su implementación a gran escala presenta casos de éxito en empresas como airbnb, CocaCola, Google y twitter.

### 6.7.2. *scikit-learn*

*scikit-learn* [38] es un conjunto de herramientas para realizar aplicaciones o proyectos de aprendizaje automático en lenguaje de programación Python. Ofrece herramientas simples y eficientes para el análisis de datos predictivo, es accesible para todos y reusable en varios contextos distintos, esta desarrollado sobre librerías como NumPy, SciPy y matplotlib y es totalmente de código abierto, para uso personal o comercial, con licencia BSD. Con estas herramientas pueden realizarse modelos de aprendizaje automático de clasificación, regresión, *clustering*, reducción de dimensionalidad, selección de modelos y preprocesamiento de datos. Además, para cada una de las funciones disponibles, *scikit-learn* presenta documentación y ejemplos que facilitan la implementación de cualquiera de sus modelos.

### 6.7.3. *PyTorch*

*PyTorch* [39] es un marco de trabajo *end-to-end* para aplicaciones de aprendizaje automático. Habilita una experimentación rápida y flexible con una producción eficiente, a través de un ecosistema de herramientas y librerías amigables con el usuario. Con *TorchScript*, *PyTorch* proporciona facilidad de uso y flexibilidad en el modo *eager*, mientras que la transición al modo gráfico para la velocidad, optimización y funcionalidad sucede sin problemas en entornos de tiempo de C++. *TorchServe* es una manera fácul de usar modelos de *Pytorch* a escala. Es agnóstica a la nube y al entorno, admitiendo funciones como servicio

multimodelo, registro, métricas y creación de puntos finales *RESTfil* para la integración de aplicaciones.

Con esta herramienta se puede optimizar el rendimiento en la producción y en la investigación, aprovechando su soporte nativo para la ejecución asíncrona de operaciones colectivas y la comunicación *peer-to-peer* accesible desde Pyton y C++. Ofrece una versión experimental para dispositivos móviles, que soporta un entorno de trabajo desde Python para desplegarse en IOS o Android. Esta versión extiende la API de *PyTorch* para cubrir las tareas de integración y preprocesamiento necesarias para incorporar el aprendizaje automático en aplicaciones móviles. Tiene la funcionalidad de poder exportar modelos en formato del estándar ONIX, para tener acceso directo en plataformas, *runtimes* y visualizadores compatibles con ONIX. Además de utilizarse con lenguaje Python, puede utilizarse en C++, ya que ofrece una interfaz en C++ que sigue el diseño y la arquitectura de la interfaz en Python. Con esto se pretende habilitar la investigación en aplicaciones de C++ con alto rendimiento y baja latencia.

# CAPÍTULO 7

# Resultados de la evaluación de las herramientas

## 7.1. Resultados de la evaluación de las herramientas de IoT diseñadas para la Raspberry Pi

Con base en la información obtenida para cada una de las siguientes herramientas [5], [6], [28], se determinó que ninguna era útil para cumplir con los objetivos planteados en este trabajo. Aunque todas las herramientas son de código abierto, con integraciones de todos los protocolos utilizados y documentación clara, son herramientas que buscan minimizar la intervención del usuario para funcionar. Para evitar que el usuario final tenga que hacer mucho trabajo para poder interactuar con sus diferentes sensores y actuadores, estas herramientas implementan metodologías que le evitan esa tarea, sin embargo, limitan la flexibilidad y escalabilidad al momento de querer realizar sistemas más complejos, como el que se se elaboró en este trabajo. Además, para poder utilizar cada uno de estos softwares es necesario cargar su imagen de sistema operativo a la *Raspberry Pi*, lo que elimina por completo la capacidad de utilizar a la *Raspberry Pi* para desempeñar otras funciones, lo que limita la utilización del hardware de este dispositivo y reduce la cantidad de posibles implementaciones con él.

## 7.2. Resultados de la evaluación de las plataformas de IoT compatibles con la Raspberry Pi

En el proceso de selección de las plataformas para aplicaciones de IoT se tomaron en consideración la siguientes opciones, [29]-[33]. La que se seleccionó fue [33], esta ofrecía las herramientas de poder administrar, con funciones de Python, dispositivos de todos los protocolos que se utilizaron. Al ser una plataforma nativa del lenguaje, su instalación es muy similar a cualquier librería Python. La configuración de cada una de las integraciones

es extremadamente sencilla y a pesar de tener en un mismo sistema diferentes dispositivos y protocolos, las funciones asociadas a cada uno de estos son similares, facilitando el proceso de aprendizaje para poder realizar cada implementación. Finalmente, como factor decisivo se tomó en cuenta la versatilidad de esta herramienta, ya que al ser una librería de Python, se puede utilizar en conjunto con otras librerías del lenguaje, lo que permite que se puedan construir una infinidad de aplicaciones de distinto grado de complejidad, con capacidad de escalarse sin limitación.

## 7.3. Resultados de la evaluación de las bases de datos compatibles con la Raspberry Pi

Para delimitar el sistema de base de datos que se iba a utilizar, se investigaron las cualidades de 3 opciones de código abierto [34]-[36]. Cada uno de estos sistemas cumplían con mantener la estructura de los datos y con ser de código abierto. Sin embargo, se seleccionó [35] para integrarse al sistema por su popularidad, documentación y especial enfoque en la integridad y fiabilidad de los datos. Además, se tiene una librería disponible en el lenguaje Python, lo que facilitó en gran manera su integración con la plataforma en la que se encontraban los sensores y actuadores.

## 7.4. Resultados de la evaluación de las herramientas para aprendizaje automático compatibles con la Raspberry Pi

La herramienta de aprendizaje automático que se seleccionó fue [38]. A diferencia de [37], [39], es una librería desarrollada para implementarse en archivos que utilicen el lenguaje Python, lo que facilita la integración con la plataforma donde se encontraban los sensores y el sistema de bases de datos utilizados.

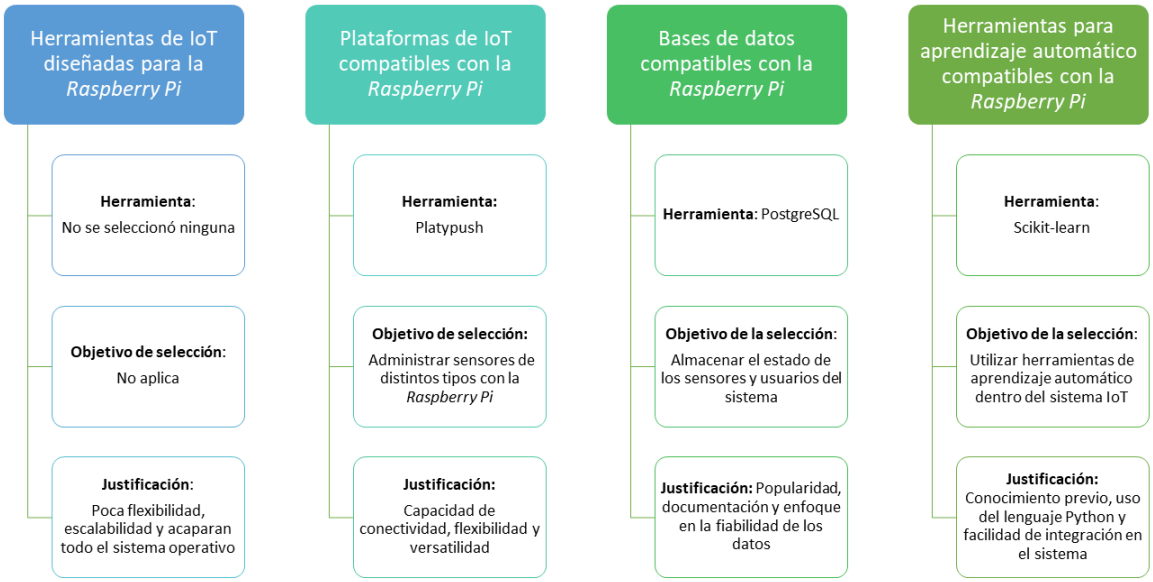


Figura 19: Resumen de los resultados de la comparación de herramientas.

# CAPÍTULO 8

## Instalación de software utilizado en la *Raspberry Pi*

## 8.1. Platypush

Platypush utiliza un servidor de redis para el almacenamiento y administración de los datos de cada integración disponible. Por esta razón, inicialmente se debe instalar un servidor de redis en la *Raspberry Pi*.

```
apt-get install redis-server
```

Una vez instalado, se debe iniciar el servicio de redis y habilitarlo para que se ejecute al reiniciar.

```
systemctl start redis.service
systemctl enable redis.service
```

Para validar que todo este correctamente instalado, con siguiente comando se puede validar que el servicio de redis esté funcionando correctamente

```
systemctl status redis.service
```

Seguido de esto, procedemos con la instalación de Platypush ejecutando el siguiente comando. Es importante mencionar Platypush esta diseñado para usarse con versiones de Python mayores a 3.5

```
pip install platypush
```

En caso de tener problemas de permisos, puede ejecutarse de la siguiente manera.

```
sudo pip install platypush
```

Si todo se instaló correctamente, al ejecutar el siguiente comando, se debería de mostrar la versión de Platypush instalada.

```
platypush --version
```

### 8.1.1. TP-Link switches

Para poder utilizar switches TP-Link desde Platypush se debe ejecutar el siguiente comando

```
pip install pyHS100
```

### 8.1.2. zigbee2mqtt

Para poder utilizar dispositivos zigbee desde Platypush se debe instalar zigbee2mqtt. Antes de comenzar con su instalación, se debe instalar node o npm. Para ello se debe copiar el link de la versión adecuada para la arquitectura del sistema operativo donde se desee descargar, que disponible en el siguiente link `https://nodejs.org/en/download/`. Con el link correcto se debe ejecutar el siguiente comando para realizar la descarga.

```
wget "link"
```

Luego de esto, hay que descomprimir el archivo con el siguiente comando, remplazando las X con los valores de la versión descargada.

```
tar -xzf node-vXX.XX.X-linux-armvXl.tar.gz
```

Luego de esto es necesario mover los directorios a alguna carpeta que este dentro del PATH de linux o colocar la dirección a la carpeta descargada en el PATH. Para este caso se moverán los archivos hacia */usr/local* un directorio que usualmente se encuentra en el PATH. Pero para validar este dato se puede usar el siguiente comando.

```
echo $PATH
```

Una vez se haya validado que el directorio */usr/local* este en el PATH, se deben ejecutar los siguientes comandos remplazando la X por los valores correspondientes.

```
cd node-vXX.XX.X-linux-armvXl/
sudo cp -R * /usr/local/
```

Finalmente, se pueden ejecutar los siguientes comandos desde cualquier directorio, para validar que el proceso de instalación este correcto.

```
node -v
npm -v
```

Continuando con el proceso de instalación, se debe ejecutar el siguiente comando para instalar el repositorio de zigbee2mqtt. Para que el comando funcione se debe tener git instalado

```
sudo git clone https://github.com/Koenkk/zigbee2mqtt.git /opt/zigbee2mqtt
```

Luego de esto, hay que darle permisos de lectura al nuestro usuario, que en este caso es pi, pero puede ser distinto.

```
sudo chown -R pi:pi /opt/zigbee2mqtt
```

Finalmente, ejecutamos los siguientes comandos para ir a la carpeta donde se encuentra zigbee2mqtt y realizar el proceso de instalación

```
cd /opt/zigbee2mqtt
npm install
```

Esta herramienta utiliza el protocolo MQTT para funcionar, de modo que es necesario tener un *broker* para mediar las comunicaciones. Con el siguiente comando se instalará mosquitto en la *Raspberry Pi* para utilizarse como *broker*.

```
sudo apt install mosquitto
```

Una vez finalizada la instalación se debe editar el archivo de configuración de zigbee2mqtt, que se encuentra en **/opt/zigbee2mqtt/data/configuration.yaml**. El archivo debe verse como el que se mostrará a continuación. Para poder utilizar dispositivos zigbee se necesita un dispositivo de traducción. En este trabajo se utilizó el siguiente `https://sonoff.tech/product/gateway-and-sensors/sonoff-zigbee-3-0-usb-dongle-plus-p/`, no es obligatorio utilizar el mismo, puede ser cualquiera dentro de la siguiente lista `https://www.zigbee2mqtt.io/guide/adapters/`. En el apartado **serial -> port** se debe colocar la ruta del dispositivo al conectarlo, se recomienza colocar la que se encuentra en **/dev/serial/by-path/** dado que esta no cambia. La opción **permit_join** debe colocarse en *true* únicamente cuando se desee integrar un nuevo dispositivo a la red.

```
homeassistant: false
permit_join: false
mqtt:
  base_topic: zigbee2mqtt
  server: mqtt://localhost:1883
serial:
  port: >-
    /dev/serial/by-path/platform-fd500000.pcie-pci-0000:01:00.0-usb-0:1.3:1.0-port0
advanced:
  homeassistant_legacy_entity_attributes: false
  legacy_api: true
  legacy_availability_payload: true
device_options:
  legacy: false
```

Por último, es útil realizar un servicio de linux para interactuar más fácilmente con zigbee2mqtt, además de configurarlo para que inicie automáticamente, al reiniciar la *Raspberry Pi*. Para ello se debe ejecutar el siguiente comando para crear el archivo donde de creará el servicio.

```
sudo touch /etc/systemd/system/zigbee2mqtt.service
```

Dentro de ese archivo se debe colocar la siguiente información.

```
[Unit]
Description=zigbee2mqtt
After=network.target

[Service]
Environment=NODE_ENV=production
ExecStart=/usr/bin/npm start
WorkingDirectory=/opt/zigbee2mqtt
StandardOutput=inherit
# Or use StandardOutput=null if you don't want Zigbee2MQTT
# messages filling syslog, for more options see systemd.exec(5)
StandardError=inherit
Restart=always
RestartSec=10s
User=pi

[Install]
WantedBy=multi-user.target
```

Una vez con el servicio creado, se deben ejecutar los siguientes comandos para iniciar el servicio y para habilitar que se inicie automáticamente luego de un reinicio.

```
sudo systemctl start zigbee2mqtt.service
sudo systemctl enable zigbee2mqtt.service
```

### 8.1.3. zwave2mqtt

Para instalar zwave2mqtt se deben ejecutar los siguientes comandos. Al igual que con zigbee, es necesario un dispositivo de traducción para poder interactuar con dispositivos zwave. Para este trabajo se utilizó el siguiente `http://www.gocontrol.com/detail.php?productId=6`.

```
cd ~
mkdir zwave-js-ui
cd zwave-js-ui
# download latest version
curl -s https://api.github.com/repos/zwave-js/zwave-js-ui/releases/latest  \
| grep "browser_download_url.*zip" \
| cut -d : -f 2,3 \
| tr -d \" \
| wget -i -
unzip zwave-js-ui-v*.zip
```

Al ejecutar zwave-js con el siguiente comando **./zwave-js-ui** se iniciará un servidor web local en el siguiente enlace `http://localhost:8091`. En **Settings -> Z-Wave -> Serial Port** se debe colocar la dirección hacia el dispositivo usb, en **Settings -> Z-Wave -> Security Keys (S0 Legacy, S2 Unauthenticated, S2 Authenticated, and S2 Access Control** se deben colocar las diferentes claves de la red, también se pueden generar automáticamente al presionar los botones que se encuentran al a par de cada espacio para escribir la clave, en **Settings -> Z-Wave -> Log Enabled, then elect a Log Leve** se debe habilitar la opción, en **Settings -> General -> Log Enabled, then elect a Log Level** se debe habilitar la opción y en **Settings -> Z-Wave -> Enable Statistics** se debe habilitar la opción.

### 8.1.4. SmartThings

Para poder utilizar dispositivos asociados a una cuenta de SmartThings inicialmente se debe de obtener el token de la respectiva cuenta que se desea operar. Para ello se debe ingresar al siguiente enlace `https://account.smartthings.com/tokens`, iniciar sesión y generar el token. Seguido de esto, se debe de ejecutar el siguiente comando para descargar la librería de Python necesaria para interactuar con los dispositivos asociados a la cuenta desde Platypush.

```
pip install pysmartthings
```

### 8.1.5. MQTT

Para utilizar MQTT basta con instalar el *broker*, como se describió anteriormente

## 8.2. Bluetooth

Para administrar dispositivos Bluetooth, se deben ejecutar los siguientes comandos, para descargar las librerías de Python necesarias.

```
sudo apt install python3-pip libglib2.0-dev
sudo pip3 install bluepy
```

## 8.3. PostgreSQL

Para descargar el sistema de bases de datos PostgreSQL, se debe ejecutar el siguiente comando.

```
sudo apt install postgresql postgresql-contrib
```

Para validar que todo este funcionando correctamente, se puede ejecutar el siguiente comando.

```
sudo systemctl status PostgreSQL
```

Luego de esto, se debe crear un usuario para la base de datos. Al ejecutar el siguiente comando, se creará un usuario y se le solicitará ingresar la contraseña del mismo.

```
sudo passwd postgres
```

Luego, con los siguientes comandos se cambia el usuario a postgres y se ingresa a la base de datos.

```
su postgres
psql
```

Cuando se visualice el *promt* de postgres, se debe ingresar el siguiente comando para colocar autenticación para el usuario postgres de la base de datos.

```
ALTER USER postgres PASSWORD 'my_postgres_password';
```

Finalmente, para poder administrar las bases de datos de PostgreSQL se debe ejecutar el siguiente comando, para descargar la librería de Python necesaria.

```
pip install psycopg2
```

## 8.4. TelegramBot

Inicialmente se debe crear el Bot en la aplicación de Telegram, para ello se debe buscar el siguiente Bot **@BotFather**, dentro de la aplicación. Seguido de esto, hay que envirale el comando /**newbot** y luego dentro del chat se le solicitarán los datos para la configuración del nuevo Bot. Una vez reciba el mensaje de confirmación del Bot, dentro de ese mensaje se encontrará el token para acceder a la API, el cual se utilizará para programar al Bot. Para poder programar un Bot de Telegram se debe ejecutar el siguiente comando para descargar la librería de Python necesaria.

```
pip install pyTelegramBotAPI
```

## 8.5. React

Para utilizar React basta con haber realizado el proceso de instalación de npm/node descrito anteriormente.

## 8.6. Aprendizaje automático

Para poder utilizar las herramientas diseñadas para aplicaciones de aprendizaje automático, es necesario descargar las siguientes librerías.

```
pip install -U scikit-learn
pip install pandas
pip install sqlalchemy
pip install seaborn
```

# CAPÍTULO 9

## Herramientas realizadas en la *Raspberry Pi*

## 9.1. Platypush

Para Platypush se realizó un archivo de configuración que permite interactuar con switches TP-Link, dispositivos asociados a una cuenta de SmartThings, dispositivos zwave, dispositivos zigbee y dispositivos capaces de utilizar el protocolo MQTT 16.1.1. Con este archivo se pueden controlar y consultar sensores del protocolo zigbee y MQTT, se puede encender, apagar y ver el estado de switches TP-Link, se puede recibir y publicar mensajes MQTT del tema seleccionado, usando el *broker* seleccionado y se pueden utilizar todas las funciones de acción y consulta disponibles en el API que ofrece SmartThings. Además, se pueden configurar otro tipo *plugins* utilizando exactamente la misma dinámica. Platypush ofrece una variedad de *triggers* automáticos, que dependiendo del *plugin* asociado, tienen diferentes causas de accionamiento. Utilizando estos, se realizó un archivo que al momento en el que reciba un mensaje de MQTT, separe los datos del mensaje recibido para ingresar cada campo a la base de datos, que en este ejemplo correponden a los estados de los sensores integrados con los microcontroladores. También, se tienen *triggers* que monitorean cambios en los dispositivos de la red zigbee y zwave para actualizar sus estados en la base de datos. Tanto el *trigger* de MQTT, como el de zigbee y el de zwave, luego de ingresar el estado del sensor que lo haya accionado, ingresan a la base de datos el estado del resto de los sensores en ese momento particular 16.1.2.

Figura 20: Foco Zigbee integrado con Platypush.

Figura 21: Sensor de temperatura y de apertura Zigbee integrado con Platypush.

Figura 22: Sensor de apertura zwave integrado con Platypush.

## 9.2. Bluetooth

El archivo que se muestra en 16.2 fue realizado con base en el archivo que se muestra en **sbm_bluetooth**. El archivo modificado realizaba la función de leer, decodificar e imprimir en terminal los mensajes de *broadcast* que enviaba el sensor de temperatura mediante Bluetooth. El cambio realizado consistió en remplazar la impresión en terminal por una inserción en base de datos. Además, la inserción en base de datos sucede únicamente cuando existe algún cambio en la temperatura o humedad del sensor y tambien se integra el estado de todos los demás sensores en el momento del cambio. El sensor utilizado `https://www.switch-bot.com/collections/store/products/switchbot-meter` es un sensor común en implementaciones de automatización de casa por lo que hay herramientas de decodificación disponibles, tales como el código utilizado. Sin embargo, utilizando las mismas funciones y estructura se puede decodificar mensajes Bluetooth de cualquier dispositivo, siempre que se tenga el contenido de cada parte del paquete.


Figura 23: Sensor Bluetooth de temperatura y humedad utilizado.

## 9.3. SmartThings y TP-Link

Dentro de los *plugins* disponibles para utilizar con Platypush, se pueden utilizar dispositivos de tipo switch de la marca TP-link y dispositivos asociados a una cuenta de SmartThings. Sin embargo, estos *plugins* no tienen *triggers* asociados, por lo que el registro de cambios no se pudo implementar como en el archivo 16.1.2. Para detectar los cambios de los sensores, se realizó un archivo que consulta por cambios cada 5 segundos. En el archivo 16.4 se tienen dos hilos que cada 5 segundos solicitan el estado de cada red correspondiente, al momento de encontrar cambios realizan una inserción del nuevo estado en base de datos, en conjunto con el estado de todos los demás sensores en ese momento. Para este trabajo, las consultas se hacen utilizando comandos de Platypush, sin embargo, la estructura del programa realizado se puede utilizar para consultar estado de cualquier tipo de sensor que pueda funcionar utilizando el lenguaje Python.

Figura 24: Foco wifi asociado a una cuenta de SmartThings.

Figura 25: Enchufe wifi TP-Link utilizado con la cafetera.

Figura 26: Enchufe wifi TP-Link utilizado en la sala.

## 9.4. TelegramBot

Se programó un Bot de Telegram para responder a solicitudes de los usuarios, respecto del estado de los sensores del sistema. Este Bot tiene integrado un proceso de autenticación en el que cada usuario nuevo debe ingresar una contraseña que le debe enviar el administrador de la casa para poder hacer solicitudes. Una vez autenticado, el nuevo usuario puede consultar individualmente el estado de cada uno de los sensores, puede solicitar el estado de todos los sensores juntos y puede alterar el estado binario de los sensores disponibles. Tanto para el proceso de autenticación como para las solicitudes, el Bot de Telegram utiliza la base de datos. De este modo, los inicios de sesión son persistentes y las consultas se resuelven de manera rápida y eficiente. Para esta aplicación se buscaba implementar la mayoría de funciones disponibles para personalizar un Bot de Telegram con Python. Se exploraron funciones de contestación inmediata, creación de mensajes, que el Bot mostrara el indicador de que estaba escribiendo, que el teclado del chat muestre únicamente las opciones disponibles y se mostró cómo guardar los pasos del usuario dentro del chat, como por ejemplo si el usuario ingresa mal la contraseña, no debe de reenviar el comando, el Bot guarda que ya realizó ese paso y vuelve a solicitarle la contraseña. Con las funciones mostradas en 16.3, se pueden hacer Bots personalizados que desempeñen funciones de varios niveles de complejidad, dentro de aplicaciones de cualquier tipo.

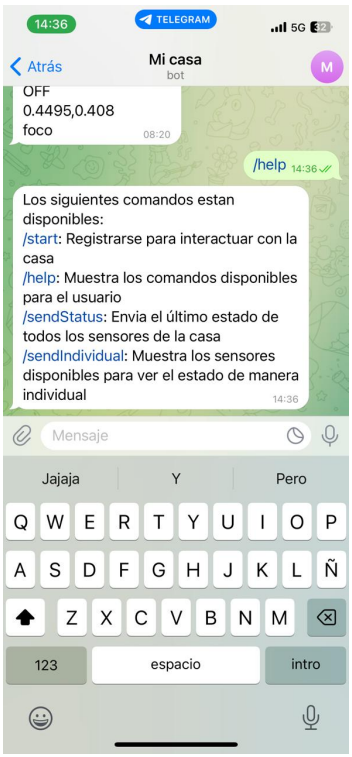


Figura 27: Comando de ayuda del Bot programado.

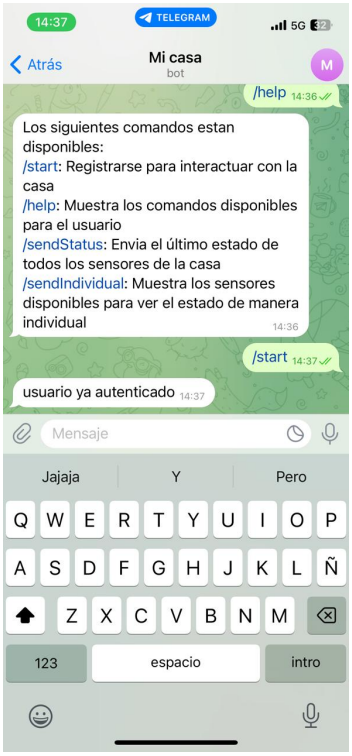


Figura 28: Comando de inicio de sesión del Bot programado.


Figura 29: Menú de los sensores disponibles.


Figura 30: Respuesta del Bot al estado de un sensor.

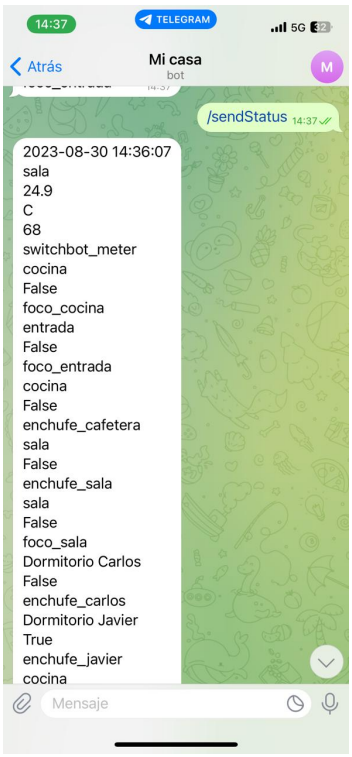


Figura 31: Respuesta del Bot al estado de todos los sensores.

## 9.5. PostgreSQL

Como parte del resultado del diseño de herramientas para integración de bases de datos en sistemas de IoT, se realizaron archivos de ejemplo que muestran como conectarse a una base de datos de PostgreSQL, como crear tablas en una base de datos, como borrar tablas en una base de datos, como insertar valores en una tabla, como consultar valores en una tabla y como actualizar valores en una tabla. Con los archivos mostrados en 16.5 se pueden realizar integraciones de bases de datos a cualquier programa desarrollado con lenguaje Python. Además, la utilidad de estas herramientas se ve incrementada al momento de integrar las funciones detalladas en conjunto con otros archivos, dando como resultado aplicaciones complejas como la que se realizó en este trabajo.

| id | device_id | date | location | state | color | temperature | temperature_unit | humidity | name |
|---|---|---|---|---|---|---|---|---|---|
| 112527 | 11 | 2023-08-30 11:37:07 | Dormitorio Carlos | OFF | 0.4495,0.408 | none | none | none | foco |
| 112526 | 10 | 2023-08-30 11:37:07 | entrada | True | none | none | none | none | puerta_entrada |
| 112525 | 9 | 2023-08-30 11:37:07 | cocina | True | none | 25 | C | none | Refrigeradora |
| 112524 | 8 | 2023-08-30 11:37:07 | Dormitorio Javier | True | none | none | none | none | enchufe_javier |
| 112523 | 7 | 2023-08-30 11:37:07 | Dormitorio Carlos | False | none | none | none | none | enchufe_carlos |
| 112522 | 6 | 2023-08-30 11:37:07 | sala | False | none | none | none | none | foco_sala |
| 112521 | 5 | 2023-08-30 11:37:07 | sala | False | none | none | none | none | enchufe_sala |
| 112520 | 4 | 2023-08-30 11:37:07 | cocina | False | none | none | none | none | enchufe_cafetera |
| 112519 | 3 | 2023-08-30 11:37:07 | entrada | False | none | none | none | none | foco_entrada |
| 112518 | 2 | 2023-08-30 11:37:07 | cocina | False | none | none | none | none | foco_cocina |
| 112517 | 1 | 2023-08-30 11:37:07 | sala | none | none | 25.8 | C | 63 | switchbot_meter |
| 112516 | 11 | 2023-08-30 11:36:07 | Dormitorio Carlos | OFF | 0.4495,0.408 | none | none | none | foco |
| 112515 | 10 | 2023-08-30 11:36:07 | entrada | True | none | none | none | none | puerta_entrada |
| 112514 | 9 | 2023-08-30 11:36:07 | cocina | True | none | 25 | C | none | Refrigeradora |
| 112513 | 8 | 2023-08-30 11:36:07 | Dormitorio Javier | True | none | none | none | none | enchufe_javier |
| 112512 | 7 | 2023-08-30 11:36:07 | Dormitorio Carlos | False | none | none | none | none | enchufe_carlos |
| 112511 | 6 | 2023-08-30 11:36:07 | sala | False | none | none | none | none | foco_sala |
| 112510 | 5 | 2023-08-30 11:36:07 | sala | False | none | none | none | none | enchufe_sala |
| 112509 | 4 | 2023-08-30 11:36:07 | cocina | False | none | none | none | none | enchufe_cafetera |
| 112508 | 3 | 2023-08-30 11:36:07 | entrada | False | none | none | none | none | foco_entrada |
| 112507 | 2 | 2023-08-30 11:36:07 | cocina | False | none | none | none | none | foco_cocina |
| 112506 | 1 | 2023-08-30 11:36:07 | sala | none | none | 25.9 | C | 63 | switchbot_meter |
| 112505 | 11 | 2023-08-30 11:32:07 | Dormitorio Carlos | OFF | 0.4495,0.408 | none | none | none | foco |

Figura 32: Tabla en PostgreSQL utilizada para almacenar los valores de los sensores.

| user_id | user_name | user_chatid |
|---|---|---|
| 1 | Carlos | 5389131177 |
| 2 | Paty | 6356321280 |

Figura 33: Tabla en PostgreSQL utilizada para almacenar los usuarios autenticados para Telegram.

## 9.6. React

Se realizó un servidor local en el que se pueden controlar los sensores del sistema. La implementación se realizó utilizando React js, donde se desarrollaron dos proyectos principales, el *frontend* y el *backend*. En el *backend* se realizaron diversos métodos que reaccionaban a publicaciones y requerimientos a diferentes URLs (ver 16.7). Estos métodos en su mayoría consistían en *queries* e inserciones diferentes tablas de la base de datos. Del lado del *frontend* se tiene un método que hace un requerimiento al *backend* cada segundo en el que se renderizan los *widgets* asociados a cada tipo de sensor, enchufe, foco, termómetro y sensor de apertura. Gracias a esta forma de implementación, al momento de integrar un nuevo sensor al sistema, no es necesario realizar cambios en el *frontend*, ya que si esta correctamente codificado este aparecerá automáticamente luego de la consulta recurrente al *backend*.


Figura 34: *frontend* del servidor local realizado.


Figura 35: *frontend* del servidor local realizado segunda vista.

## 9.7. Aprendizaje automático

Se realizaron programas que muestran como entrenar y usar modelos de clustering, regresión logística y regresión lineal. Para cada uno de los tipos de modelo se crearon archivos donde se explican a detalle cada una de las funciones utilizadas, se indica que es lo que se debe colocar en cada parámetro y que es lo que se espera como resultado. Así mismo, por la naturaleza de cada modelo, los datos que se ingresan para su entrenamiento, a pesar que vienen de la misma tabla de base de datos, no son los mismos. Los datos que ingresan a cada modelo deben pasar por un proceso de preprocesamiento, con el objetivo de modificar los

datos de manera conveniente para que al momento de ser utilizados, aporten valor al modelo y le permitan dar buenos resultados. Por otro lado, para la extracción de los datos que se utilizarán en el procesamiento y entrenamiento, se muestra una metodología en la que se genera un marco de datos manipulable en Python, de manera directa a partir de un requerimiento en base datos. La metodología utilizada presenta varias ventajas para cualquier tipo de procesamiento de datos, en especial cuando se deseen realizar aplicaciones complejas que incluyan una base de datos, pero no se tenga un dominio completo del lenguaje SQL. Cada archivo realizado se probó utilizando los datos generados por la aplicación demostrativa presentada en este trabajo, de modo que todos los procesos ilustrados en 16.8 se desarrollan en función de la aplicación, sin embargo, bajo el mismo concepto se pueden escalar para diferentes aplicaciones.

## 9.8. Servicios

Cada una de las aplicaciones mencionadas anteriormente se implementaron como servicios dentro de la *Raspberry Pi*. Para ello, se crearon todos los programas que se muestran en 16.6. La implementación de cada aplicación de esta forma, es útil por varias razones. En la administración, se puede iniciar, parar y reiniciar fácilmente cada servicio, además de que quedar configurado para iniciar automáticamente si la *Rasberry Pi* se reinicia. En la detección de errores, es útil porque dentro de los parámetros se colocó que se mostrarán las impresiones en terminal del servicio, de modo que en caso de errores queda registro de lo que sucedió. Finalmente, para aplicaciones más complejas, como la que se realizó en este trabajo, utilizar esta segmentación es útil para que si en dado caso algún servicio falle, no falle la aplicación completa, lo que reduce la vulnerabilidad y previene que el usuario final pierda en su totalidad acceso al sistema.

# CAPÍTULO 10

## Herramientas realizadas en microcontroladores

Dentro de un sistema de IoT se pueden incluir una gran variedad de sensores y actuadores que son diseñados con capacidades de conectividad para ser integrados en sistemas como estos. Sin embargo, por diferentes motivos como el precio, la accesibilidad o la disponibilidad no siempre se puede tener acceso a dispositivos que tengan estas capacidades de fábrica. Con el objetivo de desarrollar herramientas lo más versátiles posibles, se realizaron ejemplos de como integrar microcontroladores al sistema de IoT propuesto, con ello básicamente se están integrando cualquier tipo de sensor que pueda ser controlado con un microcontrolador. Es importante mencionar que los microcontroladores sí deben tener capacidades de conectividad, ya que estos serán los encargados de crear el enlace entre el *gateway* del sistema y los dispositivos que tengan conectados.

## 10.1. arduino nano 33 IoT

Se realizó un programa que revisa el estado de un sensor de movimiento para identificar si el valor digital recibido indica movimiento o no. Se hace un procesamiento para que la señal de salida se genere como un pulso, de modo que levanta una alerta cuando hay movimiento y la elimina cuando deja de detectarlo. Luego publica este estado en un tema específico utilizando el protocolo MQTT, para que pueda ser procesado e ingresado en la base de datos. 16.9.1

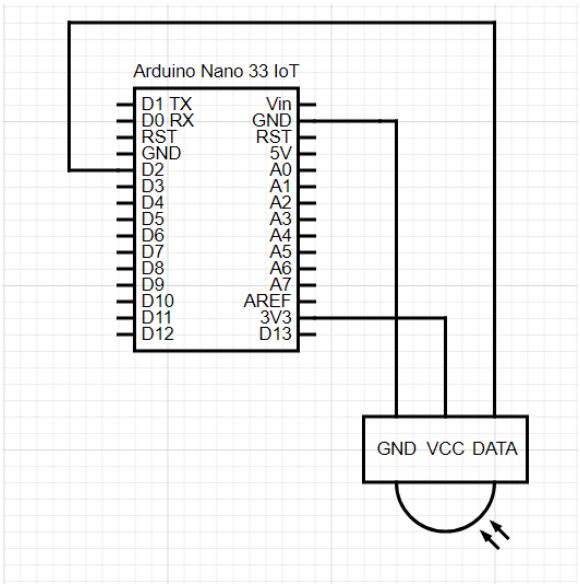


Figura 36: Esquemático de la conexión del sensor de movimiento.

## 10.2. ESP-WROOM-32

Se realizó un programa que revisa el estado de un sensor de luz cada 500ms, hace una comparación entre el valor actual y el valor anterior para detectar el un cambio suficiente como para indicar que hay luz. Luego, publica el estado correspondiente en un tema específico utilizando el protocolo MQTT, para que el estado pueda ser almacenado en la base de datos. 16.9.2

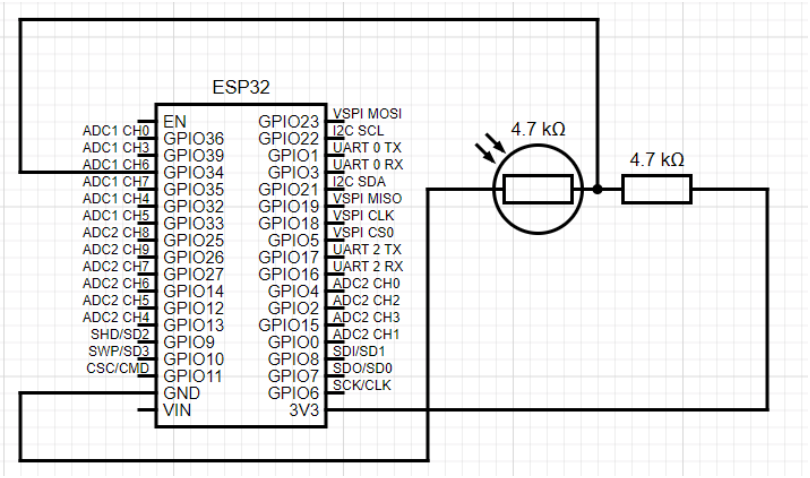


Figura 37: Esquemático del sensor de luz.

# CAPÍTULO 11

# Aplicación demostrativa realizada

## 11.1. Esquema de la aplicación

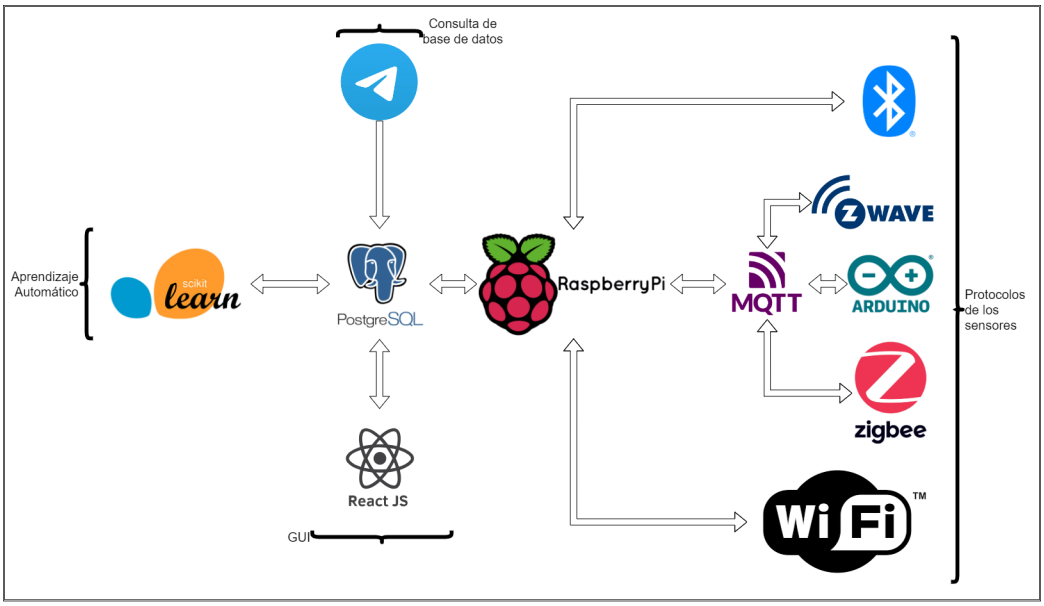


Figura 38: Arquitectura descriptiva de la aplicación realizada.

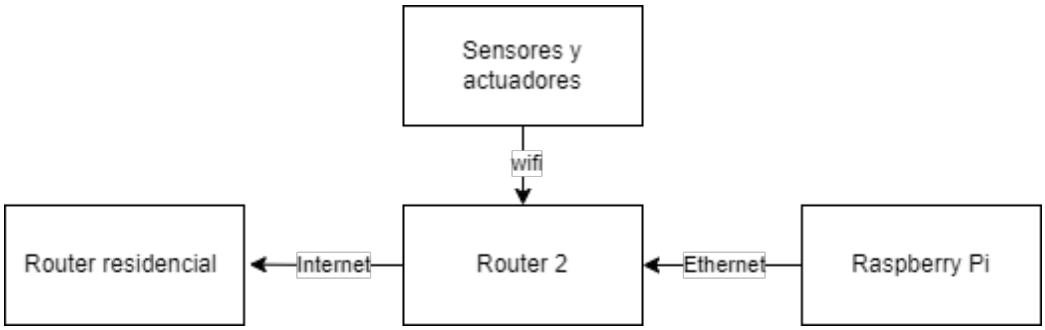


Figura 39: Diagrama físico de las conexiones realizadas.

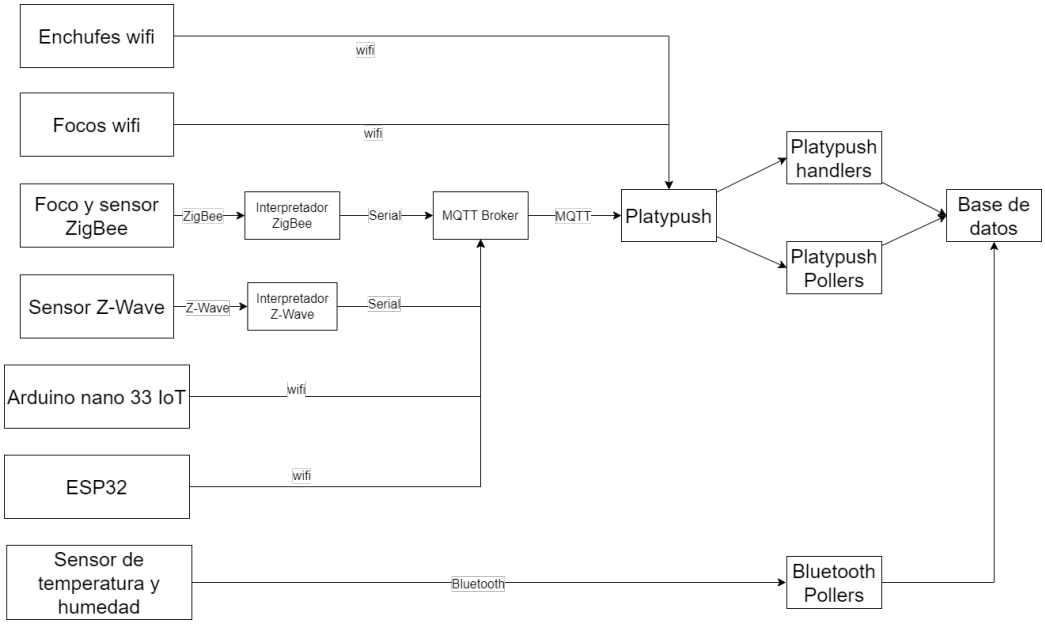


Figura 40: Diagrama del *Gateway*.

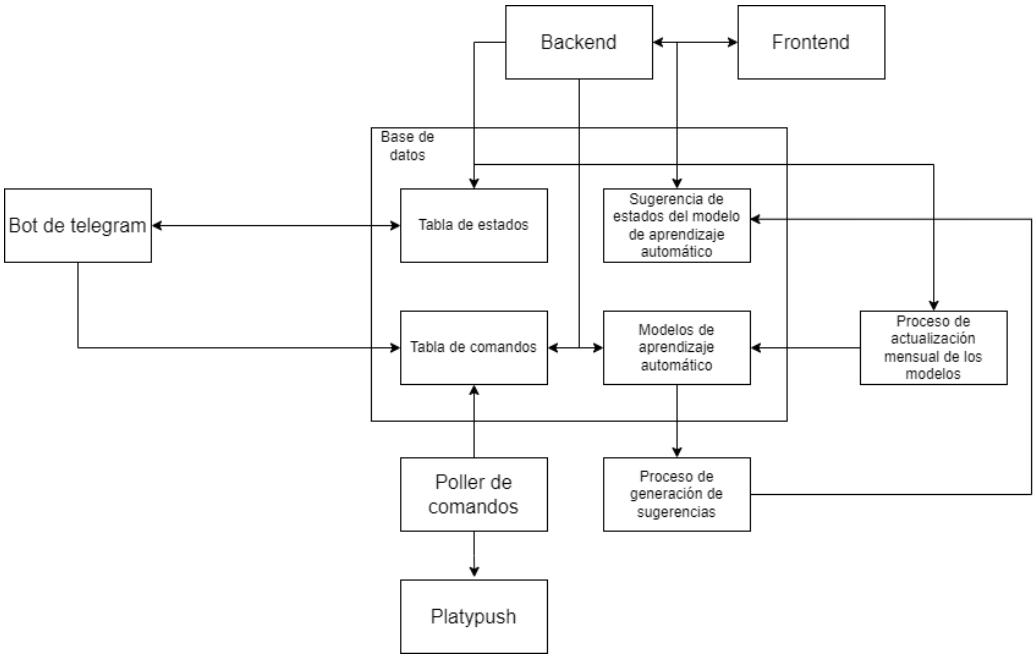


Figura 41: Diagrama de la interacción con la base de datos.

Figura 42: Maqueta demostrativa para probar las herramientas diseñadas.

## 11.2. Explicación de la aplicación

En la Figura 8 se observan de manera gráfica todas las partes que comprenden la aplicación planteada. La aplicación realizada consiste en una casa inteligente en la que se tienen sensores de 5 protocolos de red distintos, se puede consultar el estado de cada uno de estos sensores de manera remota usando un Bot de Telegram y de manera gráfica, con un servidor local realizado en react. El valor de cada sensor es almacenado en una base de datos de PostgreSQL cuando su valor cambia, además que se almacenan los valores de todos los demás sensores en el momento que sucedió el cambio. Los dos tipos de consultas que se integraron, desde Telegram y desde la página web, utilizan la base de datos como intermediario. Para el caso del Bot de Telegram, cuando recibe una solicitud para revisar el estado de algún sensor, lo que se revisa es el último estado almacenado en base de datos y ese valor es el que se utiliza para responderle al usuario. Para el caso del servidor web, al momento de mostrar el estado de los sensores se utilizará un botón en el que el usuario refrescará el

estado de la página web, cuando lo presione se realizará una solicitud del estado de todos los sensores disponibles y se desplegará gráficamente en la página. Para el caso en el que lo que el usuario desee sea cambiar el estado de algún sensor, lo que se hace es ingresar una solicitud a una tabla de la base de datos, otro archivo esta revisando constantemente esa tabla por nuevas solicitudes y cuando detecta una solicitud realiza la acción correspondiente y actualiza el estado del sensor en la página. Como último aplicativo, se tiene la integración de algoritmos de aprendizaje automático con el concepto de IoT de la aplicación. Para esta parte se desarrolló un crontab de Linux que se realiza mensualmente. La acción de este procedimiento ejecuta un código que realiza el entrenamiento y ajuste de varios modelos de predicción diferentes 16.8.3. Este código inicia con un requerimiento a la base de datos de los estados de los sensores del último mes, construye un *dataframe* con los valores de cada sensor, seguido de esto aisla un sensor de la tabla y toma su estado como evento objetivo para que el modelo de esa iteración estime su estado basado en los demás sensores. Por último, se escoge el modelo más preciso para ser almacenado en la base de datos, el cual actualiza el servicio de sugerencias 16.4. En este servicio se toma el modelo nuevo insertado y se generan las sugerencias en tiempo real, basadas en el estado de los demás sensores, para el sensor seleccionado y finalmente se muestran en el servidor local para que el usuario pueda escoger si las acepta o no. Para el procesamiento de las sugerencias se trabajó sobre la arquitectura existente para ejecutar comandos desde el *frontend*. Cuando el usuario acepta la sugerencia, esta es ingresada como un comando a la tabla de comandos, y se ejecuta como tal, con la diferencia que la acción no proviene del usuario sino que del modelo de aprendizaje automático.

# CAPÍTULO 12

## Resultados obtenidos de los algoritmos de aprendizaje automático

Para la realización de las herramientas de aprendizaje automático se implementaron dos tipos de modelos diferentes, un modelo de clusterización y un modelo de predicción. La clusterización es una dinámica que forma parte del grupo de los modelos de aprendizaje automático no supervisado, que como su nombre lo dice busca la clasificación de elementos acorde con características que estos tengan en común. Específicamente, como el algoritmo detrás del proceso de separación se implementaron 2, Kmeans y mezclas Gaussianas. Kmenas es el algoritmo de clusterización más simple, utilizado en la gran mayoría de ejemplos didácticos del aprendizaje automático, por su practicidad de comprensión e implementación fue integrado dentro de las herramientas desarrolladas, sin embargo, este algoritmo tiene la limitante que se enfoca en la clasificación de clusters circulares o elípticos, lo que hace que se reduzca su campo de aplicación. Por esta razón, se implementó un ejemplo espejo utilizando el algoritmo de mezclas Gaussianas, con una dinámica de implementación más compleja se evita la limitante de la forma de los clusters para obtener una clasificación correcta. Finalmente, para el modelo de predicción se utilizó el planteamiento de una regresión logística. Este modelo consiste en el cálculo de coeficientes de peso para cada variable de entrada del modelo, los cuales se utilizan para estimar o predecir el estado de la variable marcada como salida del sistema.

## 12.1. Clustering

La primer prueba que se realizó utilizando el archivo creado para modelos de clustering (ver 16.8) consistió en validar si con los datos que se tenían, el modelo era capaz de clasificar los sensores por cada tipo diferente. Para ello, se tuvo que realizar un preproceasamiento para quitar las columnas que no se iban a utilizar y para modificar los datos de las columnas que si se iban a utilizar. Las columnas que se utilizaron para el análisis fueron la ubicación,

el estado, la temperatura y la humedad. Todas las demás columnas no se integraron en esta prueba. Para el caso de la ubicación, se utilizó el método de *one hot* para hacer el cambio de variable cualitativa a cuantitativa, para que pudiera ser tomada en cuenta por el modelo. Por otro lado, la temperatura y la humedad, únicamente requirieron un remplazo de valores para todos aquellos sensores que no daban ese valor. Finalmente, para el caso del estado, se realizó una codificación de los estados, en las que los estados guardados como texto se colocaron como números enteros. Cada uno de estos procesos se detalla en el archivo de 16.8.

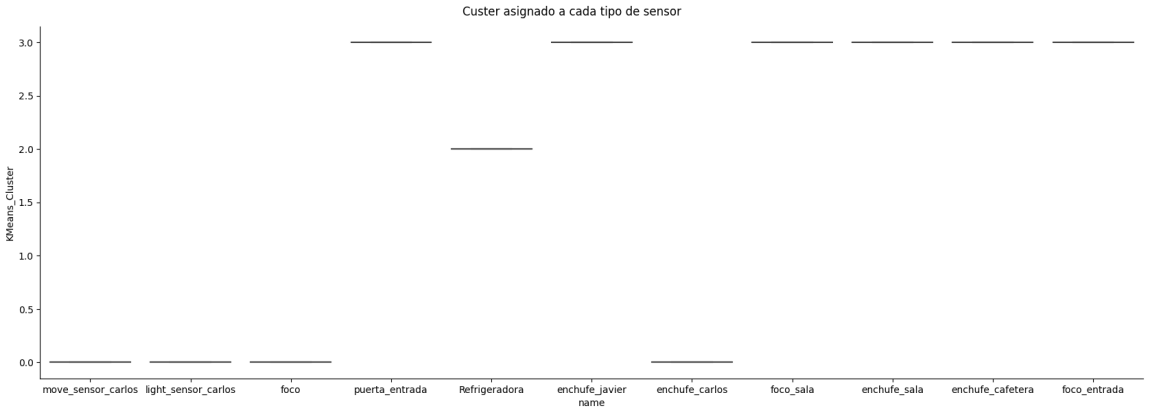


Figura 43: Cluster asignado para cada sensor.

En la Figura 43 se puede observar que el algoritmo de *clustering* separó consistentemente el grupo de sensores en 3 tipos de diferentes, termómetro, sensor de abierto con temperatura y sensores binarios. Cabe destacar que como se mencionó anteriormente, no se tomó en cuenta el color para el proceso de análisis, por esta razón, es coherente incluir a los focos, los enchufes y el sensor de apertura en el mismo grupo, ya que a nivel de estados manejan los mismos. Para este caso, podría parecer innecesario realizar una clasificación de los sensores por tipo, cuando este factor es conocido, sin embargo, esta metodología de experimentación permite validar el rendimiento de la herramienta desarrollada conociendo el resultado esperado para poder establecer si los resultados presentados son congruentes y confiables.

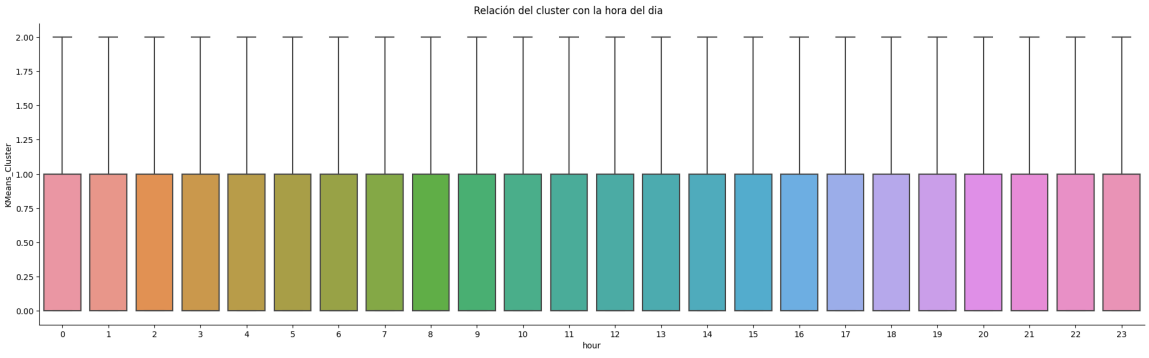


Figura 44: Cluster asignado para cada hora del día.

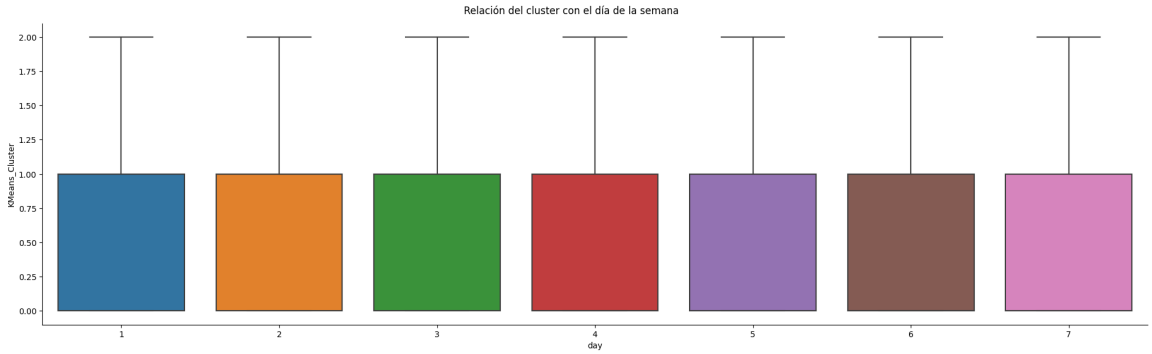


Figura 45: Cluster asignado para cada día.

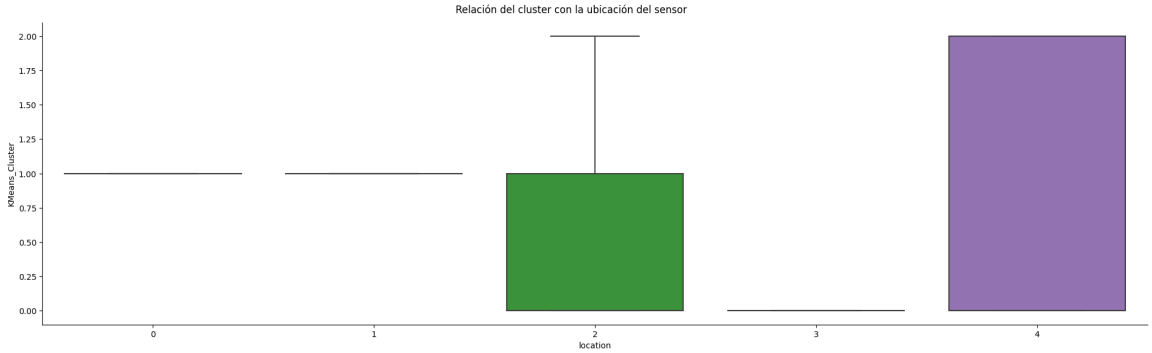


Figura 46: Cluster asignado para cada ubicación.

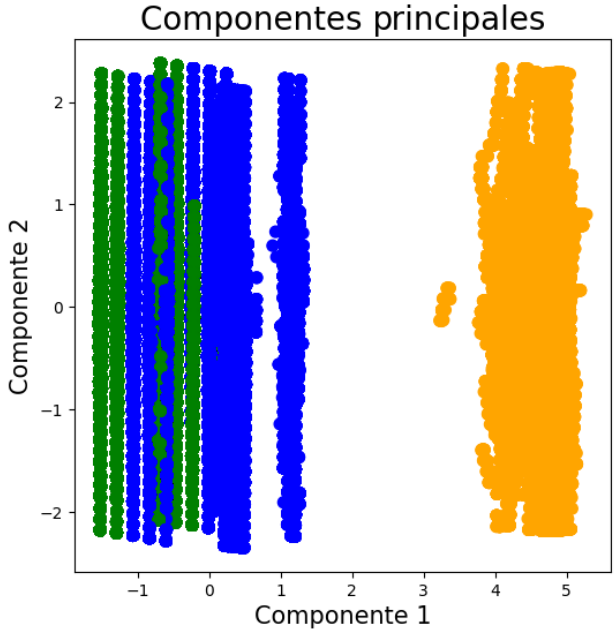


Figura 47: Análisis de componentes principales.

La segunda prueba usando el algoritmo de *clustering* consistió en integrar al grupo de variables de la primer prueba, el día de la semana y la hora asociada a cada evento. También, para esta prueba no se utilizó codificación *one hot* sino que se realizó una codificación manual. En las figuras 44 y 45 se puede observar que no se puede asociar ningún clúster específico con algún valor de hora o de día. Además, en la Figura 46 se observa que la única ubicación con más de un valor de clúster asociado es la cocina, lo cual es congruente con el hecho de que la mayoría de sensores se encuentran en esa ubicación. Finalmente, para complementar los resultados obtenidos, se realizó un análisis de componentes principales.

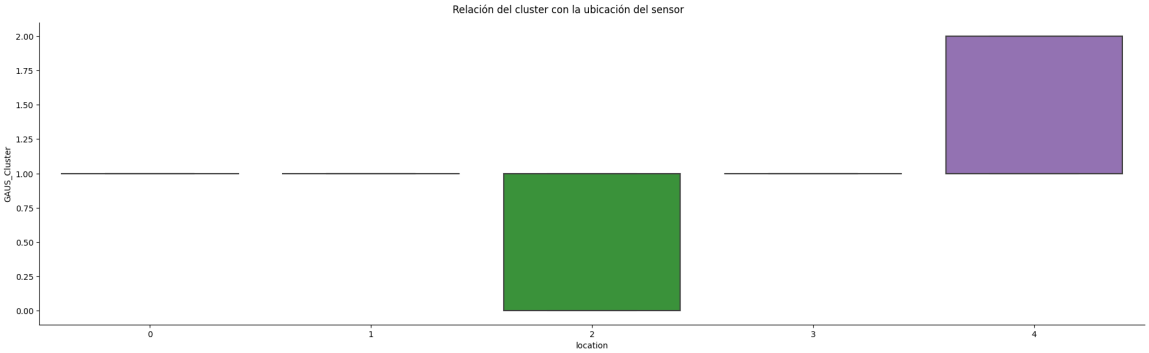


Figura 48: Cluster asignado para cada ubicación.

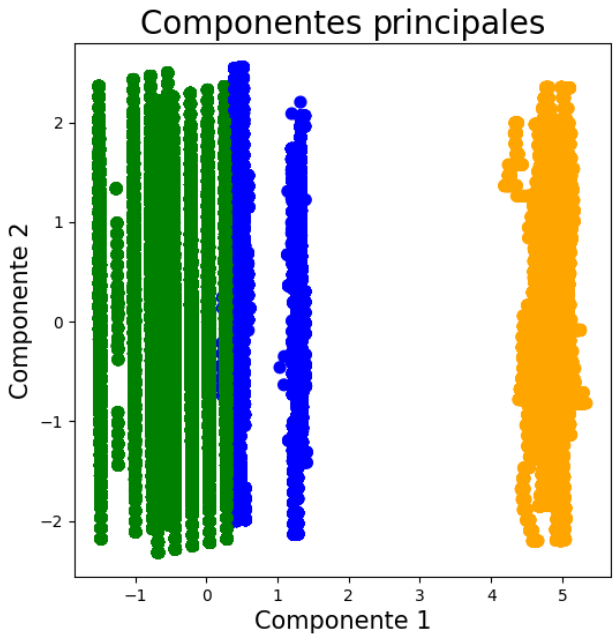


Figura 49: Análisis de componentes principales.

La tercer prueba consistió en repetir la prueba anterior, con la única diferencia de que en vez de utilizar el algoritmo de clusterización Kmeans, se utilizó mezclas Gaussianas. Con ello, se buscaba comparar ambas implementaciones para exponer los puntos a favor y en

contra de cada una. De modo que, al momento que se deseen implementar las herramientas presentadas en este trabajo, se tengan claros los alcances de cada una y así se les pueda sacar al máximo provecho. En la Figura 48 se observa que se identificó una distinción entre la ubicación de cada sensor con el clúster asignado, lo cual no se logró apreciar en la prueba anterior. Sin embargo, fue únicamente con la ubicación que se encontró una relación evidente con el clúster, y para todas las demás variables, sí se obtuvieron los mismos resultados evidenciados en las figuras anteriores.

El análisis de componentes principales es utilizado para observar los resultados del análisis de clusterización de una manera más práctica, especialmente cuando se toman varias características para el análisis. Inicialmente, se obtienen los componentes principales de todas las variables utilizadas, dos componentes en este caso, y luego se asocian con los valores de de cada uno de los clústers. Por último, se realiza un gráfico con las mismas dimensiones que componentes obtenidas, donde se utiliza un color diferente para cada clúster, para diferenciarlos sin dificultad. En la Figura 47 se observa una separación dudosa entre cada uno de los clústers, donde el de clúster amarillo muestra mayor tamaño, siendo este el que contiene a todos los enchufes y focos, que en conjunto conforman la mayoría de sensores del sistema. Por otro lado, en la Figura 49 sí se logra ver una separación precisa entre los clústers, indicando un proceso correcto de clasificación por parte del algoritmo de Mezclas Gaussianas, añadido a esto al ver esta gráfica se puede validar que se seleccionó la cantidad correcta de clústers para el análisis, dado que no se tienen clústers demasiado pegados ni colores que se traslapen. Además, por la separación que se observa es evidente que cada clúster tiene asociadas características muy particulares y específicas, las cuales son el tipo de datos que aportan y su ubicación. Tomando en cuenta los resultados obtenidos para todas las pruebas, para la forma particular de los clusters formados por sistema, la herramienta de Mezclas Gaussianas demostró un mejor rendimiento en el proceso de clasificación, sin embargo, este hecho no demerita la utilidad del algoritmo Kmenas que también ofrece buenos resultados siempre que se ajuste a las características de la aplicación.

## 12.2. Regresión logística

Para la implementación del modelo de predicción por regresión logística se desarrolló una herramienta que busca predecir el estado de un sensor basado únicamente en los estados disponibles. El código utilizado para la integración de aprendizaje automático con la aplicación realizada presenta el cálculo de coeficientes de un modelo de regresión tomando como variable objetivo cada sensor que puede ser modificado dentro del sistema. Además de este cálculo, se tienen métricas para evaluar las predicciones del modelo y con base en eso, seleccionar el mejor para ofrecerlo al usuario en forma de sugerencias. Con este modelo, se tiene una implementación mas compleja que con clusterización ya que en la arquitectura de la aplicación se tiene el entrenamiento y obtención del modelo con datos históricos para usarlos en la predicción de datos futuros. De modo que, le aporta un grado de automatización al sistema considerablemente más alto que los sistemas de IoT comunes, ya que el límite no son tareas programables o el enlace de la acción de un sensor basado en el estado de otro, con esto se ofrece un rastreo de comportamiento completo, una estimación confiable y una actualización del modelo para que tenga una mejor adaptabilidad.

| id | device_id | date | coef | bias | accuracy |
|---|---|---|---|---|---|
| 4 | 3 | 2023-10-15 13:50:00 | 0.09447169 0.05778194 -0.03802922 -0.0271692 -0.02771607 0.31458246 -0.0165161 -0.02747453 0.03529752 -0.13511916 0. 0.12690878 0.10465507 | -9.023178433986011 | 0.9994456762749445 |
| 5 | 3 | 2023-10-15 21:18:48 | 0.09447169 0.05778194 -0.03802922 -0.0271692 -0.02771607 0.31458246 -0.0165161 -0.02747453 0.03529752 -0.13511916 0. 0.12690878 0.10465507 | -9.023178433986011 | 0.9994456762749445 |
| 6 | 3 | 2023-10-23 20:33:52 | 0.09447169 0.05778194 -0.03802922 -0.0271692 -0.02771607 0.31458246 -0.0165161 -0.02747453 0.03529752 -0.13511916 0. 0.12690878 0.10465507 | -9.023178433986011 | 0.9994456762749445 |
| 7 | 7 | 2023-11-07 20:21:11 | 0.34115821 -0.26964236 -0.40049038 0.5500421 -1.30171448 -0.03781405 -0.30947351 -1.74883373 1.32923905 1.20255656 -0.05529318 -2.4682203 -1.21759267 | -8.476082979371457 | 0.9947929580957104 |
| 8 | 7 | 2023-11-09 07:47:46 | 0.1952143 -0.26761669 -0.41590731 0.51225945 -1.20758132 -0.01570004 -0.1764378 -1.51425254 1.29070767 1.18993933 -0.62216382 -2.64682879 -1.22768839 | -8.598487631823941 | 0.992414974308784 |
| 9 | 8 | 2023-11-23 22:47:56 | -0.36993012 2.57889876 -1.00954525 0.38400741 -0.87880335 -0.87481966 -0.23371436 0.08401939 0.44617002 0.21633587 -0.70485507 -2.87933784 -1.28344941 | -10.69517134012661 | 0.9916540212443096 |
| 10 | 7 | 2023-11-24 01:54:40 | -0.78568245 -0.1292762 -0.1313687 0.5252524 -0.81890028 -0.44315302 -0.20467373 -0.08124299 -0.92199374 0.20931861 -0.34403698 -1.51338764 -0.71257636 | -6.791523779852815 | 0.989138671381662 |
| 11 | 7 | 2023-11-24 17:40:55 | -0.77120019 -0.18796781 -0.14160763 0.5078221 -0.69454178 -0.48394135 -0.16996146 -0.02619424 -0.91796972 0.20715042 -0.32324137 -1.38357683 -0.76096585 | -6.711003544720266 | 0.9885902636916836 |

Figura 50: Modelos de regresión obtenidos.

En la Figura 50 se observa un histórico de los modelos de predicción obtenidos en el intervalo de tiempo del desarrollo de este trabajo. El primer factor que es importante considerar es que la precisión de los modelos es extremadamente buena, lo cual podría parecer dudoso. Este factor esta estrictamente relacionado con la naturaleza de la aplicación, ya que en el histórico de estados de un sensor se pueden tener pocas variaciones o sensores cuyo estado esté estrechamente relacionado con el de otro sensor, lo que genera un ajuste del modelo casi perfecto. Sin embargo, en conjunto con su correcto funcionamiento, bajo la misma metodología se pueden realizar implementaciones en otros campos donde no se presenten fenómenos como el que se menciona anteriormente. Y con base en la aplicación mostrada en este trabajo, se abre la posibilidad a la exploración de sistemas donde crece la complejidad pero también el aporte que el sistema propuesto ofrece.

# CAPÍTULO 13

## Conclusiones

- Se compararon entre sí, las plataformas de ThingsBoards, Thinger.io, Mainflux, OpenRemote y PlatyPush, delimitando a favor de PlatyPush, para utilizar a la *Raspberry Pi* como *Smart Gateway* para dispositivos que tengan o carezcan de funcionalidades de IoT.
- Se desarrollaron diferentes servicios para la *Raspberry Pi* que, usando PlatyPush y librerías nativas de python, le dan la capacidad de almacenar en base de datos el estado de sensores de cinco protocolos de comunicación diferentes y sensores sin capacidades IoT.
- Se diseñó un servidor web local y un bot telegram, como herramientas de administración de estados para los sensores conectados a la *Raspberry Pi*, dándole al usuario la capacidad de consultar y controlar los sensores conectados de manera remota y local.
- Se diseñaron herramientas de aprendizaje automático, utilizando las librerías de scikit-learn, para entrenar modelos de clusterización y predicción logística, validar sus resultados y presentarlos de manera legible al usuario final.
- Se logró desarrollar el modelo de una casa inteligente que integrara todas las herramientas diseñadas, ejemplificando y validando el funcionamiento de cada una de ellas.
- Se integraron exitosamente sugerencias del modelo de predicción logística en tiempo real, permitiendo que el usuario pueda darle el poder de control al sistema de un sensor específico.
- Se agregó exitosamente la funcionalidad IoT a un sensor de luz y de movimiento, utilizando los microcontroladores ESP32 y arduino nano 33 IoT.

# CAPÍTULO 14

# Recomendaciones

Al utilizar herramientas de código abierto se tienen la ventaja de poder acceder al código fuente de la aplicación que se esta integrando. Durante este trabajo se identificaron problemas con algunas funciones disponibles en el código fuente de Platypush [33], por lo que se plantean las siguientes recomendaciones.

Se recomienda revisar detenidamente el *plugin* y *backend* de HTTP, para validar que se muestren correctamente los estados de todos los sistemas integrados en Platypush. Además, se recomienda verificar los *widgets* que le permiten al usuario cambiar el estado de ciertos sensores, para comprobar que se ejecute la acción solicitada y que se muestre correctamente el cambio en la página web.

Se recomienda revisar el funcionamiento de cada una las funciones disponibles para el *plugin* de Zigbee. Asegurarse que cada función ejecute la acción acorde con lo que se indica en la documentación y que pueda ser ejecutada sin ningún problema.

En la actualidad se tiene una diversidad muy grande de sistemas de bases de datos de código abierto, con enfoques y capacidades diferentes. Por ello, se recomienda comparar el rendimiento de las bases de datos investigadas en este trabajo para establecer bajo que condiciones puntuales, con ciertas métricas de rendimiento, alguna de las opciones no utilizada podría desempeñar una mejor función, que el sistema de bases de datos seleccionado para este trabajo.

Por último, se reconoce que las herramientas diseñadas tienen la capacidad de ser implementadas en cualquier aplicación. Por ello, se recomienda probar las herramientas presentadas en este trabajo con sensores y actuadores de industrias distintas a la domótica, con la finalidad de evaluar el rendimiento de de las herramientas con dicha aplicación e integrar este nuevo concepto a industrias en las que se le podría sacar mayor provecho.

# CAPÍTULO 15

## Bibliografía


[1] M. Botterman, "Internet of Things: an early reality of the Future Internet," en *Workshop Report, European Commission Information Society and Media*, sn, vol. 15, 2009.

[2] A. S. Abdul-Qawy, P. Pramod, E. Magesh y T. Srinivasulu, "The internet of things (IoT): An overview," *International Journal of Engineering Research and Applications*, vol. 5, n.º 12, págs. 71-82, 2015.

[3] T. O. Ayodele, "Machine learning overview," *New Advances in Machine Learning*, vol. 2, págs. 9-18, 2010.

[4] Gordon Henderson, *Wiring Pi | GPIO Interface library for the Raspberry Pi*, `http://wiringpi.com`, Consultado: 2023-04-22, 2019.

[5] Webthings.io, *WebThings*, `https://webthings.io/`, Consultado: 2023-04-23, 2023.

[6] Gitbook.io, *Picroft - Mycroft AI*, `https://mycroft-ai.gitbook.io/docs/using-mycroft-ai/get-mycroft/picroft`, Consultado: 2023-04-23, 2022.

[7] S. Naik y E. Sudarshan, "Smart healthcare monitoring system using raspberry Pi on IoT platform," *ARPN Journal of Engineering and Applied Sciences*, vol. 14, n.º 4, págs. 872-876, 2019.

[8] A. Zhang, Y. Chang, S. Esche y Z. Zhang, "Application of Internet of Things in Online Robotics Class," en *2022 ASEE Annual Conference & Exposition*, 2022.

[9] L. O. Aghenta y M. T. Iqbal, "Low-cost, open source IoT-based SCADA system design using thinger. IO and ESP32 thing," *Electronics*, vol. 8, n.º 8, pág. 822, 2019.

[10] N. Singh, S. Chaturvedi y S. Akhter, "Weather forecasting using machine learning algorithm," en *2019 International Conference on Signal Processing and Communication (ICSC)*, IEEE, 2019, págs. 171-174.

[11] S. Zhong, K. Zhang, M. Bagheri et al., "Machine learning: new ideas and tools in environmental science and engineering," *Environmental Science & Technology*, vol. 55, n.º 19, págs. 12 741-12 754, 2021.

[12] M. Maksimović, V. Vujović, N. Davidović, V. Milošević y B. Perišić, "Raspberry Pi as Internet of things hardware: performances and constraints," *design issues*, vol. 3, n.º 8, págs. 1-6, 2014.

[13] Oracle.com, *¿qué es una base de datos?* https://www.oracle.com/mx/database/what-is-database/, Consultado: 2023-05-18, 2014.

[14] Amazon Web Services, Inc., *¿qué es una api? - explicación de interfaz de programación de aplicaciones¿qué es una api? - explicación de interfaz de programación de aplicaciones*, https://aws.amazon.com/es/what-is/api/, Consultado: 2023-05-20, 2023.

[15] B. Balon y M. Simić, "Using Raspberry Pi Computers in Education," en *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2019, págs. 671-676. DOI: 10.23919/MIPRO.2019.8756967.

[16] Arduino Official Store, *arduino nano 33 iot 2022*, https://store.arduino.cc/products/arduino-nano-33-iot, Consultado: 2023-05-18, 2022.

[17] *ESP32-WROOM-32*, Version 3.4, Espressif Systems, mayo de 2023.

[18] C. M. Ramya, M. Shanmugaraj y R. Prabakaran, "Study on ZigBee technology," en *2011 3rd International Conference on Electronics Computer Technology*, vol. 6, 2011, págs. 297-301. DOI: 10.1109/ICECTECH.2011.5942102.

[19] C. Paetz, *Z-Wave Essentials*. eBook Partnership, 2017.

[20] B. Fouladi y S. Ghanoun, "Security evaluation of the Z-Wave wireless protocol," *Black hat USA*, vol. 24, págs. 1-2, 2013.

[21] C. Bisdikian, "An overview of the Bluetooth wireless technology," *IEEE Communications Magazine*, vol. 39, n.º 12, págs. 86-94, 2001. DOI: 10.1109/35.968817.

[22] L. Nastase, "Security in the internet of things: A survey on application layer protocols," en *2017 21st international conference on control systems and computer science (CSCS)*, IEEE, 2017, págs. 659-666.

[23] D. Dinculeană y X. Cheng, "Vulnerabilities and limitations of MQTT protocol used between IoT devices," *Applied Sciences*, vol. 9, n.º 5, pág. 848, 2019.

[24] D. Soni y A. Makwana, "A survey on mqtt: a protocol of internet of things (iot)," en *International conference on telecommunication, power analysis and computing techniques (ICTPACT-2017)*, vol. 20, 2017, págs. 173-177.

[25] S. Kaushik, "An overview of technical aspect for WiFi networks technology," *International Journal of Electronics and Computer Science Engineering (IJECSE, ISSN: 2277-1956)*, vol. 1, n.º 01, págs. 28-34, 2012.

[26] A.-S. Brylinski y A. Bhattacharjya, "Overview of HTTP/2," en *Proceedings of the Second International Conference on Internet of things, Data and Cloud Computing*, 2017, págs. 1-6.

[27] IETF, *Hypertext Transfer Protocol – HTTP/1.1*, https://www.rfc-editor.org/rfc/pdfrfc/rfc2068.txt.pdf, Consultado: 2023-05-30, 2013.

[28] Home Assistant, *Home Assistant*, https://www.home-assistant.io/, Consultado: 2023-05-18, 2023.

[29] thingsboard, *What is ThingsBoard?* https://thingsboard.io/docs/getting-started-guides/what-is-thingsboard/, Consultado: 2023-05-18, 2017.

[30] Thinger.io, *overview - thinger.io*, https://docs.thinger.io/, Consultado: 2023-05-19, 2021.

[31] Mainflux Labs, *Mainflux — full-stack open-source, patent-free iot platform and consulting services*, https://mainflux.com/cloud.html, Consultado: 2023-05-19, 2020.

[32] OpenRemote, *open source enterprise iot platform | openremote*, https://openremote.io/product/, Consultado: 2023-05-19, 2022.

[33] Platypush.tech, *platypush*, https://platypush.tech/, Consultado: 2023-05-19, 2023.

[34] mariadb.org, *mariadb: about*, https://mariadb.org/about/, Consultado: 2023-05-19, 2023.

[35] Postgresql.org, *postgresql: about*, https://www.postgresql.org/about/, Consultado: 2023-05-19, 2023.

[36] Redis, *Introduction to Redis*, https://redis.io/docs/about/, Consultado: 2023-05-19, 2023.

[37] TensorFlow, *tensorflow core | aprendizaje automático para principiantes y expertos*, https://www.tensorflow.org/overview?hl=es-419, Consultado: 2023-05-19, 2023.

[38] F. Pedregosa, G. Varoquaux, A. Gramfort et al., "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, págs. 2825-2830, 2011.

[39] pytorch.org, *pytorch*, https://pytorch.org/features/, Consultado: 2023-05-19, 2021.

# CAPÍTULO 16

# Anexos

## 16.1. Platypush

### 16.1.1. Archivo de configuración

```
##############################################################################
# Sample platypush configuration file.
# Edit it and copy it to /etc/platypush/config.yaml for system installation or to
# ~/.config/platypush/config.yaml for user installation (recommended).
##############################################################################
#Se habilita el plugin de los switches tplink
switch.tplink:
    enabled: True
#Se habilita el plugin de smartthings
smartthings:
    access_token: 11eb4f3c-5dcd-4dff-bf85-b09277c2d82f
#Se habilita el plugin y backend de zwave js
zwave.mqtt:
    host: localhost
    name: zwave-js-ui
    topic_prefix: zwave
zigbee.mqtt:
    host: localhost
#Se habilita el plugin y backend de zigbee2mqtt
backend.zigbee.mqtt:
    enabled: true
#Se habilita el plugin y backend para escuchar publicaciones
#realizadas en localhost con los temas descritos
backend.mqtt:
    host: localhost
    topic: mcus
    subscribe_default_topic: mcus
    listeners:
        - host: localhost
          topics:
            - mcus
            - switchbot_meter/c8:16:02:7d:fa:0c
mqtt:
    host: localhost
```

### 16.1.2. Archivo de lectura de sensores

```
from platypush.event.hook import hook
from platypush.utils import run
from platypush.message.event.mqtt import MQTTMessageEvent
from platypush.message.event.zigbee.mqtt import ZigbeeMqttDevicePropertySetEvent
from platypush.message.event.zwave import ZwaveValueChangedEvent
import psycopg2
import datetime
import json
import telebot
from telebot import types

#hooks para procesar alarmas de MQTT, zigbee y zwave
@hook(MQTTMessageEvent)
def send(event, **context):
    string = event.msg #se almacena el mensaje recibido con la alarma
    separator = ","
    time = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    #se realiza la conexión a base de datos
    conn = psycopg2.connect(database = "house_sensors",user = "postgres",
                            host= 'localhost',password = "charlye72",port = 5432)
    cur = conn.cursor()
    try: #se actualiza el estado del sensor que cambio
        cur.execute(f" INSERT INTO casa_carlos(device_id,date,location,state,color, \
                    temperature,temperature_unit,humidity,name) \
                    VALUES({string.split(separator)[0]},'{time}','{string.split(separator)[3]}','{string.split(separator)[2]}','none', \
                    'none','none','none','{string.split(separator)[1]}')");
        for i in range (1,14): #se guarda el estado actual de los demás sensores en el momento del cambio de estado
            if i != int(string.split(separator)[0]):
                cur.execute(f'SELECT * FROM casa_carlos where device_id = {i}\
                            ORDER BY id DESC LIMIT 1;')
                row = cur.fetchall()
                for out in row:
                    cur.execute(f" INSERT INTO casa_carlos(device_id,date,location,state,color,\
                                temperature,temperature_unit,humidity,name) \
                                VALUES({out[1]},'{time}','{out[3]}','{out[4]}','{out[5]}',\
                                    '{out[6]}','{out[7]}','{out[8]}','{out[9]}')");
    except:
        print(f"no se pudo ingresar a la base de datos la entrada {str(string)}")
    conn.commit()
    cur.close()
    conn.close()
#el trigger es un cambio de estado en el dispositivo zigbee que se llama Refrigeradora
@hook(ZigbeeMqttDevicePropertySetEvent, device='Refrigeradora')
def on_change_sensor(event, **context):
    string = event.properties #se obtiene el mensaje recibido con la alarma
    time = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    #se realiza la conexión a base de datos
    conn = psycopg2.connect(database = "house_sensors",user = "postgres",
                            host= 'localhost',password = "charlye72",port = 5432)
    cur = conn.cursor()
    try:
```

```
        for i in range (1,14): #se actualiza el estado de los sensores en el momento del cambio
            cur.execute(f'SELECT * FROM casa_carlos where device_id = {i}\
                        ORDER BY id DESC LIMIT 1;')
            row = cur.fetchall()
            for out in row:
                if i!=9:
                    cur.execute(f" INSERT INTO casa_carlos(device_id,date,location,state,color,\
                                temperature,temperature_unit,humidity,name) \
                                    VALUES({out[1]},'{time}','{out[3]}','{out[4]}','{out[5]}',\
                                     '{out[6]}','{out[7]}','{out[8]}','{out[9]}')");
                else: #se revisa que información viene en el paquete para ver como se inserta
                    #puede traer temperatura y apertura, solo apertura o solo temperatura
                    if 'temperature' in string and 'contact' in string:
                        cur.execute(f" INSERT INTO casa_carlos(device_id,date,location,state,color, \
                        temperature,temperature_unit,humidity,name) \
                        VALUES(9,'{time}','cocina','{string['contact']}','none', \
                        '{string['temperature']}','C','none','Refrigeradora')");
                    elif 'temperature' in string and 'contact' not in string:
                        cur.execute(f" INSERT INTO casa_carlos(device_id,date,location,state,color, \
                        temperature,temperature_unit,humidity,name) \
                        VALUES(9,'{time}','cocina','{out[4]}','none', \
                        '{string['temperature']}','C','none','Refrigeradora')");
                    elif 'temperature' not in string and 'contact' in string:
                        cur.execute(f" INSERT INTO casa_carlos(device_id,date,location,state,color, \
                        temperature,temperature_unit,humidity,name) \
                        VALUES(9,'{time}','cocina','{string['contact']}','none', \
                        '{out[6]}','C','none','Refrigeradora')");

    except:
        print(f"no se pudo ingresar a la base de datos la entrada {str(string)}")
    conn.commit()
    cur.close()
    conn.close()
#hook para detectar cambios del equipo zigbee que se llama foco
@hook(ZigbeeMqttDevicePropertySetEvent, device='foco')
def on_change_foco(event, **context):
    string = event.properties #se recibe el mensaje que viene con la alarma
    time = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    #se realiza la conexión a base de datos
    conn = psycopg2.connect(database = "house_sensors",user = "postgres",
                            host= 'localhost',password = "charlye72",port = 5432)
    cur = conn.cursor()
    try: #se actualiza el estado del foco, el color se inserta como coordenadas en un plano cuyos
        #ejes son la saturación y el brillo
        cur.execute(f" INSERT INTO casa_carlos(device_id,date,location,state,color,\
                    temperature,temperature_unit,humidity,name) \
                    VALUES(11,'{time}','Dormitorio Carlos','{string['state']}','{string['color']['x']},\
                        {string['color']['y']}','none','none','none','foco')");
        for i in range (1,14): #se actualiza el estado de los demás sensores en el momento del cambio de estado
            if i != 11:
                cur.execute(f'SELECT * FROM casa_carlos where device_id = {i} ORDER BY id DESC LIMIT 1;')
                row = cur.fetchall()
                for out in row:
                    cur.execute(f" INSERT INTO casa_carlos(device_id,date,location,state,color,\
                                temperature,temperature_unit,humidity,name)\
                                 VALUES({out[1]},'{time}','{out[3]}','{out[4]}','{out[5]}',\
                                     '{out[6]}','{out[7]}','{out[8]}','{out[9]}')");
    except:
        print(f"no se pudo ingresar a la base de datos la entrada {str(string)}")
    conn.commit()
```

```
    cur.close()
    conn.close()
#hook que rastrea cambios de los equipos de la red zwave
@hook(ZwaveValueChangedEvent)
def on_change_door(event, **context):
    string = event.value #se obtiene el mensaje que viene con la alarma
    time = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    # se realiza la conexión a base de datos
    conn = psycopg2.connect(database = "house_sensors",user = "postgres",
                            host= 'localhost',password = "charlye72",port = 5432)
    trad = {22:'True',23:'False'}
    cur = conn.cursor()
    try:#se actualiza el estado del sensor que cambio
        print(f"ingresando {str(string)}")
        #se valida que el id de la propiedad sea el estado del switch magnetico
        if (string['id'] == '3-113-0-Access Control-Door state'):
            cur.execute(f" INSERT INTO casa_carlos(device_id,date,location,state,color,\
                        temperature,temperature_unit,humidity,name) \
                        VALUES(10,'{time}','entrada','{trad[string['data']]}','none','none',\
                            'none','none','puerta_entrada')");
            for i in range (1,14):
                if i != 10: #se actualiza el estado de los demás sensores al momento del cambio de estado
                    cur.execute(f'SELECT * FROM casa_carlos where device_id = {i}\
                                ORDER BY id DESC LIMIT 1;')
                    row = cur.fetchall()
                    for out in row:
                        cur.execute(f" INSERT INTO casa_carlos(device_id,date,location,state,\
                                    color,temperature,temperature_unit,humidity,name)\
                                     VALUES({out[1]},'{time}','{out[3]}','{out[4]}','{out[5]}',\
                                         '{out[6]}','{out[7]}','{out[8]}','{out[9]}')");
    except:
        print(f"no se pudo ingresar a la base de datos la entrada {str(string)}")
    conn.commit()
    cur.close()
    conn.close()
```

## 16.2. Bluetooth

```
#-------------------------------------------------------------------------------
#Autor: Carlos Gil
#fecha: 30/07/2023
#Comentarios: Modificación del archivo encontrado en https://qiita.com/warpzone/items/11ec9bef21f5b965bce3
#-------------------------------------------------------------------------------
import binascii
from bluepy.btle import Scanner, DefaultDelegate
import datetime
import json
import os
import psycopg2
macaddr = 'c8:16:02:7d:fa:0c'
lastTemperature = 0.0
lastHumidity = 0
class ScanDelegate( DefaultDelegate ):#creaciónj de la clase
  def __init__( self ):
```

```
    DefaultDelegate.__init__( self )

  def handleDiscovery( self, dev, isNewDev, isNewData ):
    global lastTemperature
    global lastHumidity
    if dev.addr == macaddr: #revisar si es la mac de nuestro sensor, para no parsear valores incorrector
      for ( adtype, desc, value ) in dev.getScanData(): #validar información recibida
        if ( adtype == 22 ): #tomar solo paquetes con información
            servicedata = binascii.unhexlify( value[4:] )

            battery = servicedata[2] & 0b01111111 #obtener la batería
            isTemperatureAboveFreezing = servicedata[4] & 0b10000000 #revosar si es temperatura negativa
            temperature = ( servicedata[3] & 0b00001111 ) / 10 + ( servicedata[4] & 0b01111111 )
            if not isTemperatureAboveFreezing:
                temperature = -temperature #convertir la temperatura
            humidity = servicedata[5] & 0b01111111 # obtener la humedad
            #leer banderas sobre los datos recibidos
            isEncrypted             = ( servicedata[0] & 0b10000000 ) >> 7
            isDualStateMode         = ( servicedata[1] & 0b10000000 ) >> 7
            isStatusOff             = ( servicedata[1] & 0b01000000 ) >> 6
            isTemperatureHighAlert = ( servicedata[3] & 0b10000000 ) >> 7
            isTemperatureLowAlert  = ( servicedata[3] & 0b01000000 ) >> 6
            isHumidityHighAlert     = ( servicedata[3] & 0b00100000 ) >> 5
            isHumidityLowAlert      = ( servicedata[3] & 0b00010000 ) >> 4
            isTemperatureUnitF      = ( servicedata[5] & 0b10000000 ) >> 7
            #revisar si hubo un cambio en la temperatura o humedad
            if ((temperature != lastTemperature) or (humidity != lastHumidity)):
              time = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
              lastHumidity = humidity #actualizar valores anteriores
              lastTemperature = temperature
              if bool( isTemperatureUnitF ): #colocar la escala de la temperatura
                 temp_scale = "F"
              else:
                 temp_scale = "C"
              string = { #construir diccionario para enviar por MQTT
                    "time": time,
                    "temperature": temperature,
                    "humidity": humidity,
                    "battery": battery,
                    "temperature_scale": temp_scale,
                    "signal_strength": dev.rssi,
                }
              print(string) #imprimir diccionario resumen
              #no es necesario procesar en la recepción de MQTT por lo que se actualizará
              #la información dentro de este mismo servicio

              #se realiza la conexión a base de datos
              conn = psycopg2.connect(database = "house_sensors",user = "postgres",\
                                      host= 'localhost',password = "charlye72",port = 5432)
              cur = conn.cursor()
              try: #se actualiza el valor de temperatura y humedad en la tabla de estados
                cur.execute(f" INSERT INTO casa_carlos(device_id,date,location,state,color,\
                            temperature,temperature_unit,humidity,name) VALUES(1,'{string['time']}',\
                              'sala','none','none','{string['temperature']}',\
                                '{string['temperature_scale']}','{string['humidity']}','switchbot_meter')");
              for i in range (1,14):
                  if i != 1: # se copia el estado actual de los demás sensores al momento del cambio
                      cur.execute(f'SELECT * FROM casa_carlos where device_id = {i} ORDER BY id DESC LIMIT 1;')
                      row = cur.fetchall()
                      for out in row:
```

```
                            cur.execute(f" INSERT INTO casa_carlos(device_id,date,location,state,\
                                        color,temperature,temperature_unit,humidity,name) VALUES({out[1]},\
79                                        '{string['time']}','{out[3]}','{out[4]}','{out[5]}','{out[6]}',\
                                          '{out[7]}','{out[8]}','{out[9]}')");
                except:
                    print(f"no se pudo guardar en la base de datos {string}")
                conn.commit()
84              cur.close()
                conn.close() #se cierra la conexión a base de datos
scanner = Scanner().withDelegate(ScanDelegate())
scanner.scan(0)
```

## 16.3. Telegram

```
import time
import string
3 import random
import telebot
import psycopg2
import paho.mqtt.client as mqtt
import json
8 from telebot import types
from datetime import datetime
TOKEN = '6385483047:AAEtrANN1-oAnHCMg9hC1_6LrqMJVFNih2s' #token del bot del telegram
command_deviceid = 0
userStep = {}
13 password = ""
commands = {  # descripción de los comandos disponibles
    'start'       : 'Registrarse para interactuar con la casa',
    'help'        : 'Muestra los comandos disponibles para el usuario',
    'sendStatus'  : 'Envia el último estado de todos los sensores de la casa',
18  'sendIndividual'    : 'Muestra los sensores disponibles para ver el estado de manera individual',
    'setIndividual'    : 'Permite cambiar el estado del sensor seleccionado'
}
#configuración especial para mostrar en el teclado en la selección de sensores y los estados disponibles
sensorState = types.ReplyKeyboardMarkup(one_time_keyboard=True)
23 sensorState.add('True','False')
sensorSelect = types.ReplyKeyboardMarkup(one_time_keyboard=True)
sensorSelect.add('Sensor de temperatura y humedad','Foco entrada','Foco sala','Foco cocina','Foco','Sensor puerta principal', \
                 'Sensor de refrigeradora','Enchufe sala', 'Enchufe cafetera','Enchufe carlos','Enchufe Javier', 'Sensor de luz Carlos','Sensor de
                     movimiento Carlos')
sensor = ['nada','Sensor de temperatura y humedad','Foco cocina','Foco entrada','Enchufe cafetera',\
28          'Enchufe sala','Foco sala', 'Enchufe carlos','Enchufe Javier', 'Sensor de refrigeradora',\
            'Sensor puerta principal','Foco','Sensor de luz Carlos','Sensor de movimiento Carlos']
hideBoard = types.ReplyKeyboardRemove()  # para esconder el teclado
#se realiza la conexión a base de datos
conn = psycopg2.connect(database = "telegramUsers",user = "postgres",\
33                          host= 'localhost',password = "charlye72",port = 5432)
cur = conn.cursor()
#se obtienen los usuarios autenticados al reiniciar el programa
cur.execute("SELECT user_chatID FROM users;")
id_vals = cur.fetchall()
38 conn.commit()
cur.close()
```

```
conn.close()
for j in id_vals:
    userStep[j[0]] = 0
# para imprimir los mensajes recibidos en la terminal
def listener(messages):
    #cuando los mensajes le llegan al bot se llama a esta función
    for m in messages:
        if m.content_type == 'text':
            print(str(m.chat.first_name) + " [" + str(m.chat.id) + "]: " + m.text)

#se crea la instancia del bot
bot = telebot.TeleBot(TOKEN)
bot.set_update_listener(listener)  # se registra la función para imprimir los mensajes en la terminal
#handler que se ejecuta cuando el bor recibe un /pwd
#comando para solicitar el password
@bot.message_handler(commands=['pwd'])
def command_start(m):
    global password
    cid = m.chat.id
    if (cid == 5389131177): #revisa que sea el chat id del administrador
        length_of_string = 8
        #genera una contraseña alfanumérica random
        password = ''.join(random.choice(string.ascii_letters + string.digits)\
                           for _ in range(length_of_string))
        bot.send_message(cid, password) #le envia la contraseña al administrador
    else:
        bot.send_message(cid, "no eres el administrador")

# handler que se ejecuta cuando el bor recibe un /start
@bot.message_handler(commands=['start'])
def command_start(m):
    cid = m.chat.id
    #se realiza una conexión a base de datos
    conn = psycopg2.connect(database = "telegramUsers",user = "postgres",\
                            host= 'localhost',password = "charlye72",port = 5432)
    cur = conn.cursor()
    #se obtienen los usuarios registrados
    cur.execute("SELECT user_chatID FROM users;")
    id_vals = cur.fetchall()
    conn.commit()
    cur.close()
    conn.close()
    tempCid = (cid,)
    #se revisa si el usuario ya esta autenticado para dejarlo avanzar
    if tempCid in id_vals:
        bot.send_message(cid, "usuario ya autenticado")
        userStep[cid] = 0
    else:
        userStep[cid] = 1  # se actualiza el estado del chat del usuario y se le solicita la contraseña
        bot.send_message(cid, "Por favor ingrese la contraseña")

@bot.message_handler(func=lambda a: userStep[a.chat.id] == 1)
def command_default(m):
    global password
    # se revisa que la respuesta del usuario concuerde con la contraseña generada por el administrador
    if (m.text == password):
        #mensaje de bienvenida
        bot.send_message(m.chat.id, "Bienvenido a mi casa")
        #se actualiza el estado del chat del usuario
        userStep[m.chat.id] = 0
```

```
        command_help(m) #se le envían los comandos disponibles
        #se realiza una conexión a la base de datos
        conn = psycopg2.connect(database = "telegramUsers",
                        user = "postgres",
                        host= 'localhost',
                        password = "charlye72",
                        port = 5432)
        cur = conn.cursor()
        #se ingresa el usuario a la lista de usuarios authenticados
        try:
             cur.execute(f'INSERT INTO users(user_name, user_chatID) \
                          VALUES(\'{m.chat.first_name}\',{m.chat.id})');
             conn.commit()
        except:
             bot.send_message(m.chat.id, "Usuario ya autenticado")
        cur.close()
        conn.close() #se cierra la conexión a base de datos
    else: #si la contraseña es incorrecta el usuario no puede avanzar
        bot.send_message(m.chat.id, "Contraseña incorrecta, intente de nuevo")

# handler que se ejecuta cuando el bor recibe un /help
@bot.message_handler(commands=['help'])
def command_help(m):
    cid = m.chat.id
    help_text = "Los siguientes comandos estan disponibles: \n"
    for key in commands:  # se genera el mensaje de apoyo al usuario
        help_text += "/" + key + ": "
        help_text += commands[key] + "\n"
    bot.send_message(cid, help_text)  #se envía el mensaje al usuario

# handler que se ejecuta cuando el bot recibe un /sendIndividual
@bot.message_handler(commands=['sendIndividual'])
def command_image(m):
    cid = m.chat.id
    #se muestran en el teclado los sensores disponibles para consultar el estado
    bot.send_message(cid, "Escoja el sensor que desea ver", reply_markup=sensorSelect)
    userStep[cid] = 2  # se actualiza el estado del chat del usuario


@bot.message_handler(func=lambda a: userStep[a.chat.id] == 2)
def msg_image_select(m):
    cid = m.chat.id
    text = m.text
    #se hace una conexión a base de datos
    conn = psycopg2.connect(database = "house_sensors",user = "postgres",\
                            host= 'localhost',password = "charlye72",port = 5432)
    cur = conn.cursor()
    sensors = []
    mensaje = ""
    try: #se solicita el estado del sensor solicitado por el usuario
        for a in range (1,14):
            cur.execute(f'SELECT * FROM casa_carlos where device_id = {a} \
                        ORDER BY id DESC LIMIT 1;')
            row = cur.fetchall()
            for out in row:
                sensors.append(out)
    except:
        print(f"no se pudo ingresar consultar la base de datos")
    conn.commit()
```

```
    cur.close()
    conn.close()
    #se muestra la animación de escribiendo en el chat
    bot.send_chat_action(cid, 'typing')
    #se revisa que el sensor ingresado por el usuairo corresponda con el que trae el query
    if text in sensor:
        for i in range (0,len(sensors)):
            #para cada columna disponible por sensor, se construye un mensaje
            #solo con información útil
            if (sensors[i][1] == sensor.index(text)):
                for j in range(0,len(sensors[i])):
                    if (sensors[i][j] != "none" and j>1):
                        mensaje += f"{sensors[i][j]}\n"
        #se manda el mensaje al usuario
        bot.send_message(cid, mensaje)
        #se actualiza el estado del chat
        userStep[cid] = 0
    else:
        bot.send_message(cid, "El sensor no existe")
        bot.send_message(cid, "Intente de nuevo")

# handler que se ejecuta cuando el bot recibe un /setIndividual
@bot.message_handler(commands=['setIndividual'])
def command_image(m):
    cid = m.chat.id
    #se muestra al usuario los sensores disponibles para modificar
    bot.send_message(cid, "Escoja el sensor que desea modificar", reply_markup=sensorSelect)
    userStep[cid] = 3  # se actualiza el estado de lchat

@bot.message_handler(func=lambda a: userStep[a.chat.id] == 3)
def command_image(m):
    global command_deviceid
    cid = m.chat.id
    text = m.text
    #se obtiene el device_id de la respuesta dada por el usuario
    command_deviceid = sensor.index(text)
    print(command_deviceid)
    #se le muestran al usuario las opciones de comando disponibles (encender o apagar)
    bot.send_message(cid, "Escoja el estado para el sensor", reply_markup=sensorState)
    userStep[cid] = 4  # se actualiza el estado del chat



@bot.message_handler(func=lambda a: userStep[a.chat.id] == 4)
def msg_image_select(m):
    global command_deviceid
    cid = m.chat.id
    text = m.text

    bot.send_chat_action(cid, 'typing')
    #se hace una conexión a base de datos
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    conn = psycopg2.connect(database = "house_sensors",user = "postgres",\
                            host= 'localhost',password = "charlye72",port = 5432)
    cur = conn.cursor()
    sensors = []
    mensaje = ""
    error = ""
    try: #se ingresa a la tabla de comandos la solicitud del usuario
```

```
        cur.execute(f"INSERT INTO
            casa_carlos_commands(device_id,date,activo,color,procesado)values({command_deviceid},'{timestamp}',{text},'none',0);")
    except Exception as e:
        error = e
        print(f"no se pudo insertar el comando en la base de datos, {e}")
    conn.commit()
    cur.close()
    conn.close()#se cierra la conexión a base de datos
    #si todo fue exitoso se le confirma al usario que se modificó el estado del sensor
    if (error == ""):
        bot.send_message(cid, f"se modificó el estado del sensor con id= {command_deviceid}")
    userStep[cid] = 0 #se actualiza el estado del chat

# handler que se ejecuta cuando el bot recibe un /sendStatus
@bot.message_handler(commands=['sendStatus'])
def command_send_status(m):
    cid = m.chat.id
    #se hace la conexión a base de datos
    conn = psycopg2.connect(database = "house_sensors",user = "postgres",\
                            host= 'localhost',password = "charlye72",port = 5432)
    cur = conn.cursor()
    sensors = []
    mensaje = ""
    try: #se obtiene el estado de todos los sensores
        for a in range (1,14):
            cur.execute(f'SELECT * FROM casa_carlos where device_id = {a} \
                        ORDER BY id DESC LIMIT 1;')
            row = cur.fetchall()
            for out in row:
                sensors.append(out)
    except:
        print(f"no se pudo ingresar consultar la base de datos")
    conn.commit()
    cur.close()
    conn.close()#se cierra la conexión a base de datos
    bot.send_chat_action(cid, 'typing')
    for i in range (0,len(sensors)): #se construye el mensaje con el último estado de todos los sensores
        if i == 0:
            mensaje += f"{sensors[i][2]}\n"
        for j in range(0,len(sensors[i])):
            if (sensors[i][j] != "none" and j>2):
                mensaje += f"{sensors[i][j]}\n"
    bot.send_message(cid, mensaje) #se le manda el mensaje al usuario
    userStep[cid] = 0 #se actualiza el estado del chat
#para que se quede encerrado esperando por triggers para algún handler
bot.infinity_polling()
```

## 16.4. Smartthings, Tplink y sugerencias del modelo de aprendizaje automático

```
import threading
import json
import requests
import aiohttp
import asyncio
```

```
import pysmartthings
import time
from datetime import datetime
import numpy as np
import psycopg2
from platypush.context import get_plugin
import os
nameFocos = ['foco_sala','foco_cocina','foco_entrada']
lastStatefoco = ["False","False","False"]
ubiFocos = ['sala','cocina','entrada']
focos = [6,2,3]
enchufes = {"Enchufe Sala":"enchufe_sala","Enchufe Carlos":"enchufe_carlos",\
            "Enchufe cafetera":"enchufe_cafetera","Enchufe Javier":"enchufe_javier"}
lastStateEnchufe = {"Enchufe Sala":"false","Enchufe Carlos":"false",\
                    "Enchufe cafetera":"false","Enchufe Javier":"false"}
enchufesID = {"Enchufe Sala":5,"Enchufe Carlos":7,\
              "Enchufe cafetera":4,"Enchufe Javier":8}
ubiEnchufes = {"Enchufe Sala":"sala","Enchufe Carlos":"Dormitorio Carlos",\
               "Enchufe cafetera":"cocina","Enchufe Javier":"Dormitorio Javier"}
def poll_smarthings():
    global nameFocos
    global lastStatefoco
    global ubiFocos
    global focos
    while(1):
        try:
            #se obtiene el estado de cada sensor conectado a la red de smarthings
            dicttion = json.loads(str(get_plugin('smartthings').status()))
            #se hace un ciclo for de 3 iteraciones, comenzando en 3, porque en ese valor comienzan los
            #indices dentro del json de respuesta
            for j in range (3,6):
                #se revisa si el estado actual es distinto al anterior para procesar el nuevo estado
                if (f"{dicttion['response']['output'][j]['switch']}" != lastStatefoco[j-3]):
                    #se actualiza el nuevo estado
                    lastStatefoco[j-3] = f"{dicttion['response']['output'][j]['switch']}"
                    #se obtiene la fecha actual
                    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
                    #se crea la conexión a base de datos
                    conn = psycopg2.connect(database = "house_sensors",user = "postgres",\
                                            host= 'localhost',password = "charlye72",port = 5432)
                    cur = conn.cursor()
                    #se actualiza cada uno de los 13 sensores
                    for i in range (1,14):
                        #para los sensores que no cambiaron, se obtiene el estado anterior y se
                        #ingresa el mismo estado con la fecha actual
                        if i != focos[j-3]:
                            cur.execute(f'SELECT * FROM casa_carlos where device_id = {i} \
                                        ORDER BY id DESC LIMIT 1;')
                            row = cur.fetchall()
                            for out in row:
                                cur.execute(f" INSERT INTO casa_carlos(device_id,date,location,state,\
                                            color,temperature,temperature_unit,humidity,name) VALUES({out[1]},\
                                                '{timestamp}','{out[3]}','{out[4]}','{out[5]}','{out[6]}',\
                                                    '{out[7]}','{out[8]}','{out[9]}')");
                    #para el sensor que cambió, se ingresa el nuevo estado con la nueva fecha
                    cur.execute(f"select * from casa_carlos_commands where device_id = {focos[j-3]} order by id desc limit 1;")
                    command = cur.fetchall()
                    cur.execute(f" INSERT INTO casa_carlos(device_id,date,location,state,\
                                color,temperature,temperature_unit,humidity,name) VALUES({focos[j-3]},\
                                    '{timestamp}','{ubiFocos[j-3]}',\
```

```
                                        '{dicttion['response']['output'][j]['switch']}','{command[0][4]}',\
                                            'none','none','none','{nameFocos[j-3]}')");
                    conn.commit()
                    cur.close()
                    conn.close()
                    time.sleep(0.01)
        except Exception as e:
                print(f"no se pudo ingresar a la base de datos la entrada {str(dicttion)}, {e}")
        time.sleep(1)

def poll_kasa_sensors():
    while(1):
        try:
            #se obtiene el estado de cada sensor wifi
            diction = json.loads(str(get_plugin('switch.tplink').status()))
            #se repite el proceso para cada elemento de la red
            for i in range (0,len(diction['response']['output'])):
                #se compara el estado actual con el estado anterior de cada sensor, para
                #evaluar si alguno cambio de estado y asi procesar sus estados
                if (f"{diction['response']['output'][i]['on']}" != lastStateEnchufe[diction['response']\
                                                                                    ['output'][i]['name']]):
                    lastStateEnchufe[diction['response']['output'][i]['name']] = str(diction['response']\
                                                                                    ['output'][i]['on'])
                    #se obtiene la fecha actual
                    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
                    #se realiza la conexión con base de datos
                    conn = psycopg2.connect(database = "house_sensors",user = "postgres",\
                                            host= 'localhost',password = "charlye72",port = 5432)
                    cur = conn.cursor()
                    #se actualiza el estado del sensor que cambió
                    cur.execute(f" INSERT INTO casa_carlos(device_id,date,location,state,\
                                color,temperature,temperature_unit,humidity,name) \
                                VALUES({enchufesID[diction['response']['output'][i]['name']]},'{timestamp}',\
                                    '{ubiEnchufes[diction['response']['output'][i]['name']]}',\
                                        '{diction['response']['output'][i]['on']}','none','none',\
                                            'none','none','{enchufes[diction['response']['output'][i]['name']]}')");
                    #para los sensores que no cambiaron, se copia su estado anterior con la fecha del evento
                    for a in range (1,14):
                        if a != enchufesID[diction['response']['output'][i]['name']]:
                            cur.execute(f'SELECT * FROM casa_carlos where device_id = {a}\
                                        ORDER BY id DESC LIMIT 1;')
                            row = cur.fetchall()
                            for out in row:
                                cur.execute(f" INSERT INTO casa_carlos(device_id,date,location,state,\
                                            color,temperature,temperature_unit,humidity,name) \
                                            VALUES({out[1]},'{timestamp}','{out[3]}','{out[4]}',\
                                                '{out[5]}','{out[6]}','{out[7]}','{out[8]}','{out[9]}')");

                    conn.commit()
                    cur.close()
                    conn.close()
                    time.sleep(0.01)
        except Exception as e:
                print(f"no se pudo ingresar a la base de datos la entrada {str(diction)}, {e}")
        time.sleep(1)

def poll_machine_learning_action():
    coefs = []
    bias =0.0
    device_id = 0
```

```
    last_result = "False"
    last_device = 0
    os.system("python3 /home/pi/Documentos/Tesis/machine_learning/prediccion_logistica/process_data.py")
    while(1):
        try:
            #se obtiene la fecha de hoy
            timestamp = datetime.now()
            #se realiza la conexión a base de datos
            conn = psycopg2.connect(database = "house_sensors",user = "postgres",\
                                            host= 'localhost',password = "charlye72",port = 5432)
            cur = conn.cursor()
            #se obtiene el último modelo almacenado
            cur.execute("select * from casa_carlos_ML order by id desc limit 1;")
            rows = cur.fetchall()
            #se obtiene la diferencia de fechas entre el último elemento almacenado y la fecha actual
            datediff = timestamp - rows[0][2]
            #se revisa si el último modelo almacenado tiene menos de un día de diferencia y si es
            #diferente al último modelo almacenado
            if datediff.days < 1 and bias != float(rows[0][4]):
                #si tiene menos de un día y es diferente, se actualizan los coeficientes y el
                #device_id y objetivo
                coefs = [float(x) for x in rows[0][3].split(" ") if x != '']
                bias = float(rows[0][4])
                device_id = rows[0][1]
            #se obtiene el estado actual de todos los sensores y la fecha
            sensors = []
            cur.execute(f"SELECT temperature FROM casa_carlos where device_id = 1 order by id desc limit 1;")
            sensors.append(float(cur.fetchall()[0][0]))
            cur.execute(f"SELECT humidity FROM casa_carlos where device_id = 1 order by id desc limit 1;")
            sensors.append(int(cur.fetchall()[0][0]))
            for i in range(2,12):
                if i != device_id-1:
                    cur.execute(f"SELECT state FROM casa_carlos where device_id = {i} order by id desc limit 1;")
                    sensors.append(int(cur.fetchall()[0][0] == 'True'))
            cur.execute(f"SELECT date FROM casa_carlos where device_id = 1 order by id desc limit 1;")
            fecha = cur.fetchall()[0][0]
            sensors.append(fecha.day)
            sensors.append(fecha.hour)
            #se obtiene el cálculo de la probabilidad con base en el estado actual de los sensores
            result = 1 / (1 + np.exp(-(bias + np.dot(sensors, coefs))))
            if result > 0.5:
                result_string = "True"
            else:
                result_string = "False"
            #se obtiene el nombre del dispositivo que se controlará con el modelo
            cur.execute(f"select name from casa_carlos where device_id = {device_id} limit 1")
            name = cur.fetchall()[0][0]
            #se revisa si hubo un cambio de estado o de dispositivo para procesar el cambio
            if (last_result != result_string) or (last_device != device_id):
                #se ingresa la sugerencia de estado a la tabla de estados brindados por el modelo de aprendizaje automático
                cur.execute(f"insert into casa_carlos_ML_states(device_id, date,state,name) values ({device_id},'{timestamp.strftime('%Y-%m-%d
                    %H:%M:%S')}','{result_string}','{name}')")
                last_result = result_string
                last_device = device_id
            conn.commit()
            cur.close()
            conn.close()
        except Exception as e:
            print(f"se presenento el error: {e}")
        time.sleep(1)
```

```
186
#si este archivo se ejecuta como principal, se crea 1 hilo por cada una de las funciones creadas anteriormente
if __name__ == "__main__":
    z = threading.Thread(target=poll_machine_learning_action)
191 z.start()
    y = threading.Thread(target=poll_kasa_sensors)
    y.start()
    x = threading.Thread(target=poll_smarthings)
    x.start()
196 x.join()
```

## 16.5. PostgreSQL

### 16.5.1. Crear y borrar tablas dentro de una base de datos

```
import psycopg2 #se crea la conexión a base datos
2 conn = psycopg2.connect(database = "house_sensors",
                          user = "postgres",
                          host= 'localhost',
                          password = "charlye72",
                          port = 5432)
7 # se hace el cursor
cur = conn.cursor()
# Comando para eliminat una tabla
cur.execute("DROP TABLE IF EXISTS casa_carlos")
cur.execute("DROP TABLE IF EXISTS users")
12 #comando para crear una tabla, otros tipos de variables que se pueden usar
#los que se muestran en este ejemplo son los utilizados en la aplicación
cur.execute("""CREATE TABLE casa_carlos(
            id SERIAL PRIMARY KEY,
            device_id INT NOT NULL,
17          date timestamp,
            location VARCHAR (50) NOT NULL,
            state VARCHAR (50),
            color VARCHAR (50),
            temperature VARCHAR (50) ,
22          temperature_unit VARCHAR (50) ,
            humidity VARCHAR (50) ,
            name VARCHAR (50) NOT NULL);
            """)
27 # se finaliza la consulta
conn.commit()
# se cierra la conexión con la base de datos
cur.close()
conn.close()
```

### 16.5.2. Insertar, borrar y actualizar valores de una tabla

```
import psycopg2
import datetime #se crea una conexión con la base de datos
conn = psycopg2.connect(database = "house_sensors",
                        user = "postgres",
                        host= 'localhost',
                        password = "charlye72",
                        port = 5432)
# se crea el cursor
cur = conn.cursor()
time = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
string = "13,move_sensor_carlos,OFF,Dormitorio Carlos"
separator = ","
#se muestra un ejemplo para insertar en base de datos
#notese la facilidad de insertar mezclando instrucciones de sql con python
cur.execute(f" INSERT INTO casa_carlos(device_id,date,location,state,color, \
                    temperature,temperature_unit,humidity,name) \
                    VALUES({string.split(separator)[0]},'{time}','{string.split(separator)[3]}','{string.split(separator)[2]}','none', \
                    'none','none','none','{string.split(separator)[1]}')");
string = "12,light_sensor_carlos,OFF,Dormitorio Carlos"
separator = ","
cur.execute(f" INSERT INTO casa_carlos(device_id,date,location,state,color, \
                    temperature,temperature_unit,humidity,name) \
                    VALUES({string.split(separator)[0]},'{time}','{string.split(separator)[3]}','{string.split(separator)[2]}','none', \
                    'none','none','none','{string.split(separator)[1]}')");

for i in range (1,12):
            cur.execute(f'SELECT * FROM casa_carlos where device_id = {i}\
                        ORDER BY id DESC LIMIT 1;')
            row = cur.fetchall()
            for out in row:
                cur.execute(f" INSERT INTO casa_carlos(device_id,date,location,state,color,\
                            temperature,temperature_unit,humidity,name) \
                                VALUES({out[1]},'{time}','{out[3]}','{out[4]}','{out[5]}',\
                                '{out[6]}','{out[7]}','{out[8]}','{out[9]}')");

#se borran las filas cuyo id es 1,2,3,4
#se pueden utilizar otras condiciones tambien
cur.execute("DELETE FROM casa_carlos WHERE id in (1,2,3,4);")
#en este ejemplo se borran las filas cuya fecha esta dentro del intervalo seleccionado
cur.execute("DELETE FROM casa_carlos WHERE date between '2023-10-15' and '2023-10-16';")

#se actualiza la columna procesado de la fila cuyo id es 1
cur.execute(f'update casa_carlos_commands set procesado=1 where id=1;')

#se finaliza la consulta
conn.commit()
# se cierra la conexión con la base de datos
cur.close()
conn.close()
```

### 16.5.3. Consultar valores de una tabla

```
import psycopg2 # se crea la conexión a base de datos, remplazar cada elemento por los nombre correspondientes
conn = psycopg2.connect(database = "house_sensors",
                        user = "postgres",
                        host= 'localhost',
                        password = "charlye72",
                        port = 5432)
cur = conn.cursor()#se crea un cursor
#se ejecuta una consulta
cur.execute('SELECT * FROM casa_carlos;')
#se realiza el fetch para jalar los valores de la consulta
rows = cur.fetchall()
#se hace un commit para finalizar la consulta
conn.commit()
#rows almacena una lista de listas, en la que cada linea de la base de datos
#se almacena como una lista y cada columna es un elemento de la misma
for row in rows:
    print(row)
cur.close()
conn.close()
```

## 16.6. Servicios

### 16.6.1. Platypush

```
# platypush systemd service example.
# Edit and copy this file to your systemd folder. It's usually
# /usr/lib/systemd/user for global installation or
# ~/.config/systemd/user for user installation. You can
# then control and monitor the service through
# systemd [--user] [start|stop|restart|status] platypush.service

[Unit]
Description=Platypush daemon
After=network.target redis.service

[Service]
# platypush installation path
User=pi
WorkingDirectory=/etc/platypush
ExecStart=/usr/local/bin/platypush
Restart=always
# How long should be waited before restarting the service
# in case of failure.
RestartSec=10

[Install]
WantedBy=multi-user.target
```

### 16.6.2. Bluetooth

```
[Unit]
Description=switchbot Meter
After=network.target

[Service]
ExecStart=sudo python3 /home/pi/Documentos/Tesis/Bluetooth/read_test.py
WorkingDirectory=/home/pi/Documentos/Tesis/Bluetooth
StandardOutput=inherit
StandardError=inherit
Restart=always
RestartSec=10s
User=pi

[Install]
WantedBy=multi-user.target
```

### 16.6.3. zwave2mqtt

```
[Unit]
Description=zwave2mqtt
After=network.target

[Service]
ExecStart=/home/pi/zwave-js-ui/zwave-js-ui
WorkingDirectory=/home/pi/zwave-js-ui/
StandardOutput=inherit
# Or use StandardOutput=null if you don't want Zigbee2MQTT messages filling syslog, for more options see systemd.exec(5)
StandardError=inherit
Restart=always
RestartSec=10s
User=pi

[Install]
WantedBy=multi-user.target
```

### 16.6.4. Telegram Bot

```
[Unit]
Description=Telegram Bot
After=network.target

[Service]
ExecStart=python3 /home/pi/Documentos/Tesis/telegram/tele_meter.py
WorkingDirectory=/home/pi/Documentos/Tesis/telegram
StandardOutput=inherit
```

```
StandardError=inherit
Restart=always
RestartSec=10s
User=pi

[Install]
WantedBy=multi-user.target
```

### 16.6.5. Poleo de sensores de Smartthings, Tplink y sugerencias del modelo de aprendizaje automático

```
[Unit]
Description=Service to poll sensors from kasa and smartthings
After=network.target postgresql.service

[Service]
ExecStart=python3 /home/pi/Documentos/Tesis/Poleo_sensores/poll_sensors.py
WorkingDirectory=/home/pi/Documentos/Tesis/Poleo_sensores
StandardOutput=inherit
StandardError=inherit
Restart=always
RestartSec=10s
User=pi

[Install]
WantedBy=multi-user.target
```



### 16.6.6. *Backend*

```
[Unit]
Description=backend service
After=network.target

[Service]
Environment=NODE_ENV=production
ExecStart=/usr/bin/npm start
WorkingDirectory=/home/pi/Documentos/Tesis/GUI/backend
StandardOutput=inherit
StandardError=inherit
Restart=always
RestartSec=10s
User=pi

[Install]
WantedBy=multi-user.target
```

### 16.6.7. *frontend*

```
[Unit]
Description=frontend service
After=network.target backend.service

[Service]
Environment=NODE_ENV=production
ExecStart=/usr/bin/npm run start-prod
WorkingDirectory=/home/pi/Documentos/Tesis/GUI/frontend
StandardOutput=inherit
StandardError=inherit
Restart=always
RestartSec=10s
User=pi

[Install]
WantedBy=multi-user.target
```

### 16.6.8. Procesamiento de comandos de usuario

```
[Unit]
Description=Service to execute commands from frontend
After=network.target postgresql.service

[Service]
ExecStart=python3 /home/pi/Documentos/Tesis/GUI/reader.py
WorkingDirectory=/home/pi/Documentos/Tesis/GUI
StandardOutput=inherit
StandardError=inherit
Restart=always
RestartSec=10s
User=pi

[Install]
WantedBy=multi-user.target
```

## 16.7. React

### 16.7.1. *Backend*

```
const express = require('express');
const cors = require('cors');
const knex = require('knex');
```

```
require('dotenv').config();
const db = knex({ //instanacia de la conexión a base de datos
    client: 'pg',
    connection: {
        host: process.env.DATABASE_HOST,
        user: process.env.DATABASE_USERNAME,
        password: process.env.DATABASE_PASSWORD,
        database: process.env.DATABASE,
    },
});
const app = express();
app.use(express.urlencoded({ extended: false }));
app.use(express.json()); //si detecta peticiones en formato json hace la traduccion
// urls desde donde se le pueden hacer solicitudes al backend
const allowedOrigins = ['http://25.44.71.139:3000', 'http://192.168.1.13:3000'];
app.use(cors({
    origin: allowedOrigins,
    methods: 'GET,HEAD,PUT,PATCH,POST,DELETE',
    credentials: true,
  })); //método get para que el frontend actualice el estado de todos los sensores
  app.get('/', (req, res) => {
    const subquery = db
        .select('*')
        .from('casa_carlos')
        .orderBy('date', 'desc')
        .limit(13);

    db.select('*')
        .from({ subtable: subquery })
        .orderBy('device_id', 'desc')
        .then((data) => {
            console.log(data);
            res.json(data);
        })
        .catch((err) => {
            console.log(err);
            res.status(500).json({ error: 'Error en la consulta.' });
        });
});
// método get para que el frontend obtenga el último comando ingresado
app.get('/commands', (req, res) => {
    db.select('*')
        .from('casa_carlos_commands')
        .orderBy('id', 'desc')
        .limit(1)
        .then((data) => {
            res.json(data);
        })
        .catch((err) => {
            console.log(err);
        });
}); //método get para que el frontend obtenda la última sugerencia
app.get('/ML', (req, res) => { //aportada por el modelo de ML
    db.select('*')
        .from('casa_carlos_ml_states')
        .orderBy('id', 'desc')
        .limit(1)
        .then((data) => {
            res.json(data);
        })
```

```
        .catch((err) => {
            console.log(err);
        });
});
// metodo para que el frontend inserte comandos en la tabla de comandos
app.post('/:deviceID', (req, res) => {
    const deviceID = req.params.deviceID;
    const fecha = new Date();
    const { device_id, activo, color, procesado } = req.body;
    db('casa_carlos_commands')
        .insert({
            device_id: device_id,
            date: fecha,
            activo: activo,
            color: color,
            procesado: procesado,
        })
        .then(() => {
            console.log(req.body)
            console.log('Command added');
            return res.json({ msg: 'Command Added' });
        })
        .catch((err) => {
            console.log(req.body)
            console.log(err);
        });
});
const port = process.env.PORT || 5000;
app.listen(port, () => console.log(`Server running on port ${port}, http://localhost:${port}`));
```

### 16.7.2. *frontend*

```
import React, { Fragment, useState, useEffect } from 'react';
import { HslaStringColorPicker } from 'react-colorful';
import "./styles.css";
import './App.css';
const App = () => {
    const API_URL = 'http://192.168.1.13:5000/'; //URL del frontend
    const [apiData, setApiData] = useState([]);//variable para los estados de los sensores
    const [apiCommand, setApiCommand] = useState([]);//variable para las sugerencias del modelo de ML
    const [color, setColor] = useState('hsla(169, 0%, 0%, 1)'); //variables para almacenar el color de los focos
    const [focoColors, setFocoColors] = useState({});
    const [checkboxState, setCheckboxState] = useState( //variables para almacenar el estado de los checkboxes
        localStorage.getItem('checkboxState') === 'true' || false
    );// Inicializa con el valor almacenado en localStorage
    const [checkboxStates, setCheckboxStates] = useState({});// Inicializa con el valor almacenado en localStorage
    const [previousInfoState, setPreviousInfoState] = useState(null); //variables para almacenar la última sugerencia del modelo de ML
    const [previousDeviceInfo, setPreviousDeviceInfo] = useState(null);
    const colorPickerStyle = { //estado inicial del color picker
        width: '200px',
        height: '40px',
        border: 'none', // Elimina el borde
      };
    let dataUpdateInterval; //variable para almacenar el valor del timer para actualizar el estado de los componentes
    // Función para manejar el cambio del checkbox
```

```
    const handleCheckboxChangeML = () => { //funcion para obtener el nuevo estado del checkbox de aceptar la sugerencia del modelo de ML
        const newState = !checkboxState; //y guardarlo localmente para que no se pierda al refrescar la página
        setCheckboxState(newState);
        localStorage.setItem('checkboxState', newState); // Almacena el estado en localStorage
    };

    const handleCheckboxChange = (deviceID) => {
        // Actualiza el estado local del componente
        setCheckboxStates((prevStates) => ({
            ...prevStates,
            [deviceID]: !prevStates[deviceID], // Invierte el estado
        }));
        // Actualiza localStorage con el nuevo estado
        localStorage.setItem('checkboxStates', JSON.stringify({
            ...checkboxStates,
            [deviceID]: !checkboxStates[deviceID],
        }));
        //llama el método de insersión del bachend para procesar el cambio de estado
        fetch(`${API_URL}${deviceID}`, {
            method: 'POST',
            headers: {
            'Content-Type': 'application/json',
            },//solo ingresa estado encendido o apagado, no ingresa color
            body: JSON.stringify({ device_id: deviceID, activo: !checkboxStates[deviceID], color : 'none', procesado:0}),
        })
            .then((response) => {
            if (!response.ok) {
                throw new Error('Error en la solicitud.');
            }
            return response.json();
            })
            .then((data) => {
            console.log('Estado actualizado en el backend:', data);
            })
            .catch((error) => {
            console.error('Error al actualizar el estado en el backend:', error);
            });
    };


    const handleBulbChange = (id,x) => {
        // Actualiza el estado local del componente
        setCheckboxStates((prevStates) => ({
            ...prevStates,
            [id]: !prevStates[id], // Invierte el estado
        }));
        // Actualiza localStorage con el nuevo estado
        localStorage.setItem('checkboxStates', JSON.stringify({
            ...checkboxStates,
            [id]: !checkboxStates[id],
        }));
        const initialFocoColors = JSON.parse(localStorage.getItem('focoColors')) || {};
        // Realiza un POST request al backend para actualizar el estado del foco
        fetch(`${API_URL}${id}`, {
            method: 'POST',
            headers: {
            'Content-Type': 'application/json',
            },//ingresa estado de encendido, apagado y color de las bombillas
            body: JSON.stringify({ device_id: id, activo: !checkboxStates[id], color : initialFocoColors[id], procesado:0}),
        })
```

```
            .then((response) => {
            if (!response.ok) {
                throw new Error('Error en la solicitud.');
            }
            return response.json();
            })
            .then((data) => {
            console.log('Estado actualizado en el backend:', data);

            })
            .catch((error) => {
            console.error('Error al actualizar el estado en el backend:', error);
            });
    };
    const handleBulbColorChange = (id,x) => {
        // Actualiza el estado local del componente
        setFocoColors((prevStates) => ({
            ...prevStates,
            [id]: x,
        }));
        // Actualiza localStorage con el nuevo estado
        localStorage.setItem('focoColors', JSON.stringify({
            ...focoColors,
            [id]: x,
        }));
        // Realiza un POST request al backend para actualizar el estado
        fetch(`${API_URL}${id}`, {
            method: 'POST',
            headers: {
            'Content-Type': 'application/json',
            },
            //solo se ingresa el color, cuando se realiza una acción en el color picker
            body: JSON.stringify({ device_id: id, activo: "none", color : x, procesado:0}),
        })
            .then((response) => {
            if (!response.ok) {
                throw new Error('Error en la solicitud.');
            }
            return response.json();
            })
            .then((data) => {
            console.log('Estado actualizado en el backend:', data);
            console.log(`Color guardado en el Local Storage para el ID ${id}: ${x}`);
            })
            .catch((error) => {
            console.error('Error al actualizar el estado en el backend:', error);
            });
    };
    const fetchSensorData = () => {
        // solicita al backend el estado más reciente de los sensores
        // y los almacena en apiData

        fetch(API_URL)
            .then((response) => {
            if (!response.ok) {
                throw new Error('Error en la solicitud.');
            }
            return response.json();
            })
            .then((data) => {
```

```
            setApiData(data);
            })
            .catch((error) => {
            console.error('Error al obtener datos:', error);
            });

    };

    const fetchCommandData = () => {
        // Solitica al backend la última sugerencia del modelo de ML
        // y la almacena en setApiCommand

        fetch(`${API_URL}ML`)
            .then((response) => {
            if (!response.ok) {
                throw new Error('Error en la solicitud.');
            }
            return response.json();
            })
            .then((data) => {
            setApiCommand(data);
            })
            .catch((error) => {
            console.error('Error al obtener datos:', error);
            });
    };

      // Función para actualizar los datos periódicamente
    const updateDataPeriodically = () => {
        dataUpdateInterval = setInterval(() => {
            fetchSensorData();
            fetchCommandData();

        }, 1000); // actualizar estado cada segundo
    };
    const insertMLSuggestions = (id,state) =>{
        fetch(`${API_URL}${id}`, {
            method: 'POST',
            headers: {
            'Content-Type': 'application/json',
            },
            //insersión de las sugerencias del modelo de ML como un comando del frontend
            body: JSON.stringify({ device_id: id, activo: state, color : 'none', procesado:0}),
        })
            .then((response) => {
            if (!response.ok) {
                throw new Error('Error en la solicitud.');
            }
            return response.json();
            })
            .then((data) => {
            console.log('Estado actualizado en el backend:', data);
            })
            .catch((error) => {
            console.error('Error al actualizar el estado en el backend:', error);
            });
    };
    useEffect(() => {

        fetchSensorData(); // Obtener datos al cargar el componente
```

```
        fetchCommandData();
        updateDataPeriodically(); //llamar la función para actualizar
        // Recupera los estados almacenados en localStorage
        const savedCheckboxStates = JSON.parse(localStorage.getItem('checkboxStates')) || {};
        // Inicializa el estado focoColors con los colores de los sensores 'foco'
        const initialFocoColors = JSON.parse(localStorage.getItem('focoColors')) || {};
        setCheckboxStates(savedCheckboxStates);
        setFocoColors(initialFocoColors);

    },[]);
    return (
        <Fragment>
            <header>
                <h1>Sensores de la casa de Carlos</h1>
            </header>
            {apiCommand.map((info)=>{
                            return(
                                <section>
                                <div>
                                    <h1>El modelo sugiere que el estado del sensor: {info.name} podría ser {info.state}</h1>


                                    <label class="switch-button" for={"ML"}>
                                        <div class="switch-outer">
                                            <input id="ML" type="checkbox" checked={checkboxState} onChange={handleCheckboxChangeML}/>
                                            <div class="button">
                                            <span class="button-toggle"></span>
                                            <span class="button-indicator"></span>
                                            </div>
                                        </div>
                                    </label> {/*Revisar si hay cambios en el comando para no ejecutarlo más de una vez*/}
                                    {checkboxState && (previousInfoState !== info.state ||previousDeviceInfo !== info.device_id) &&(
                                        <Fragment> {/*Insertar el comando y actualizar información previa */}
                                        {insertMLSuggestions(info.device_id,info.state)}
                                        {setPreviousInfoState(info.state)}
                                        {setPreviousDeviceInfo(info.device_id)}
                                        </Fragment>
                                    )}
                                </div>
                                </section>
                            );
                        })}
            <main>
<section>
{apiData.map((sensor) => { {/*Mapear el estado de cada sensor en la variable sensor*/}
return (
<div> {/*Renderizar componentes acorde con el tipo, hay un objeto estandar para foco, enchufe, sensor de arduino y sensor de temperatura*/}
{(sensor.name.includes('enchufe') &&
    <div className="sensor-container" key={String(sensor.device_id)}>

        <h1>{sensor.name}</h1>
        <label class="container">
        <input type="checkbox" checked={sensor.state==='True'? true:false} for={String(sensor.device_id+100)} id={String(sensor.device_id+100)}/>
        <div class="checkmark">
            <svg xmlSpace="preserve" viewBox="0 0 49.548 49.549" xmlnsXlink="http://www.w3.org/1999/xlink" xmlns="http://www.w3.org/2000/svg"
                id="Capa_1" version="1.1">
        <g>
            <g>
                <g>
                    <path d="M30.203,4.387v4.385c7.653,2.332,13.238,9.451,13.238,17.857c0,10.293-8.373,18.667-18.667,18.667
```

```
                        S6.106,36.922,6.106,26.629c0-8.405,5.585-15.525,13.238-17.857V4.387C9.323,6.835,1.855,15.866,1.855,26.629
                        c0,12.639,10.281,22.92,22.919,22.92s22.92-10.281,22.92-22.92C47.694,15.865,40.224,6.835,30.203,4.387z"></path>
                </g>
                <g>
                    <path d="M24.776,27.225c-1.41,0-2.554-1.145-2.554-2.555V2.554c0-1.41,1.144-2.554,2.554-2.554c1.41,0,2.554,1.144,2.554,2.554
                        V24.67C27.33,26.08,26.186,27.225,24.776,27.225z"></path>
                </g>
            </g>
        </g>
        </svg>
        </div>
        </label>
        <label class="switch-button" for={String(sensor.device_id)}>
        <div class="switch-outer">
            <input id={String(sensor.device_id)} type="checkbox" checked={checkboxStates[sensor.device_id]} onChange={()
                =>handleCheckboxChange(sensor.device_id)}/>
            <div class="button">
            <span class="button-toggle"></span>
            <span class="button-indicator"></span>
            </div>
        </div>
        </label>
        </div>

)}
{(sensor.name.includes('foco') &&
<div className="sensor-container" key={String(sensor.device_id)} >
        <h1>{sensor.name}</h1>
        <label class="container">
        <input type="checkbox" checked={sensor.state==='True'? true:false} for={String(sensor.device_id+100)} id={String(sensor.device_id+100)}/>
        <div class="checkmark">
            <svg xmlSpace="preserve" viewBox="0 0 49.548 49.549" xmlnsXlink="http://www.w3.org/1999/xlink" xmlns="http://www.w3.org/2000/svg"
                id="Capa_1" version="1.1">
        <g>
            <g>
                <g>
                    <path d="M30.203,4.387v4.385c7.653,2.332,13.238,9.451,13.238,17.857c0,10.293-8.373,18.667-18.667,18.667
                        S6.106,36.922,6.106,26.629c0-8.405,5.585-15.525,13.238-17.857V4.387C9.323,6.835,1.855,15.866,1.855,26.629
                        c0,12.639,10.281,22.92,22.919,22.92s22.92-10.281,22.92-22.92C47.694,15.865,40.224,6.835,30.203,4.387z"></path>
                </g>
                <g>
                    <path d="M24.776,27.225c-1.41,0-2.554-1.145-2.554-2.555V2.554c0-1.41,1.144-2.554,2.554-2.554c1.41,0,2.554,1.144,2.554,2.554
                        V24.67C27.33,26.08,26.186,27.225,24.776,27.225z"></path>
                </g>
            </g>
        </g>
        </svg>
        </div>
        </label>
                                        <div className="color-picker-container">
                                        <HslaStringColorPicker
                                        color={focoColors[sensor.device_id]} // Usa el color correspondiente del estado focoColors
                                        style={colorPickerStyle}
                                        onChange={(color) => { {/*Se ejecuta al haber un cambio en algún color picker*/}
                                            //llama la función para insertar en base de datos
                                            handleBulbColorChange(sensor.device_id,color);
                                            // Actualiza el color seleccionado para el sensor 'foco'
                                            setFocoColors((prevFocoColors) => ({
                                            ...prevFocoColors,
                                            [sensor.device_id]: color,
```

```
                                        }));
                                        setColor(color);
                                    }}
                                    />
                                    </div>
                                    <label class="switch-button" for={String(sensor.device_id)}>
                                    <div class="switch-outer"> {/*Switch que muestra el estado de la variable almacenada y ejecuta un handler
                                         cuando su estado cambia */}
                                        <input id={String(sensor.device_id)} type="checkbox" checked={checkboxStates[sensor.device_id]}
                                             onChange={() =>handleBulbChange(sensor.device_id,color)}/>
                                        <div class="button">
                                        <span class="button-toggle"></span>
                                        <span class="button-indicator"></span>
                                        </div>
                                    </div>
                                    </label>
                                    </div>

                                )}
{(sensor.name.includes('meter')&&
<div className="sensor-container" key={String(sensor.device_id)}>
<h1>{sensor.name}</h1>        {/*Mustra el valor de temperatura y humedad, junto con un boton para refrescar*/}
<h2>Temperatura: {sensor.temperature} {sensor.temperature_unit}</h2>
<h2>Humedad: {sensor.humidity} %</h2>
<button class="button_r" onClick={fetchSensorData}>
<svg class="svg-icon" fill="none" height="20" viewBox="0 0 20 20" width="20" xmlns="http://www.w3.org/2000/svg"><g stroke="#ff342b"
     stroke-linecap="round" stroke-width="1.5"><path d="m3.33337 10.8333c0 3.6819 2.98477 6.6667 6.66663 6.6667 3.682 0 6.6667-2.9848 6.6667-6.6667
     0-3.68188-2.9847-6.66664-6.6667-6.66664-1.29938 0-2.51191.37174-3.5371 1.01468"></path><path d="m7.69867 1.58163-1.44987
     3.28435c-.18587.42104.00478.91303.42582 1.0989l3.28438 1.44986"></path></g></svg>
<span class="lable">Refresh</span>
</button>
</div>
)} {/*Widget para mostrar el estado de los sensores de apertura*/}
                                {(sensor.name.includes('puerta')&&
                                <div className="sensor-container" key={String(sensor.device_id)}>
                                <h1>{sensor.name}</h1>
                                <div class="checkbox-wrapper-10">
                                    <input checked={sensor.state==='True'? true:false} type="checkbox" id="cb51" class="tgl tgl-flip"/>
                                    <label for="cb51" data-tg-on="Abierta" data-tg-off="Cerrada" class="tgl-btn"></label>
                                  </div>
                                </div>

                                )}
                                {(sensor.name.includes('Refrigeradora')&&
                                <div className="sensor-container" key={String(sensor.device_id)}>
                                <h1>{sensor.name}</h1>
                                <h2>Temperatura: {sensor.temperature} {sensor.temperature_unit}</h2>
                                <div class="checkbox-wrapper-10">
                                    <input checked={sensor.state!=='True'? true:false} type="checkbox" id="cb52" class="tgl tgl-flip"/>
                                    <label for="cb52" data-tg-on="Abierta" data-tg-off="Cerrada" class="tgl-btn"></label>
                                  </div>
                                </div>

                                )}{/*Widget para mostrar el estado de los sensores de los arduinosr*/}
                                {(sensor.name.includes('light')&&
                                <div className="sensor-container" key={String(sensor.device_id)}>
                                <h1>{sensor.name}</h1>
                                <div class="checkbox-wrapper-10">
                                    <input checked={sensor.state==='ON'? true:false} type="checkbox" id="cb53" class="tgl tgl-flip"/>
                                    <label for="cb53" data-tg-on="Hay luz" data-tg-off="No hay luz" class="tgl-btn"></label>
```

```
                                </div>
                            </div>

                            )}
                            {(sensor.name.includes('move')&&
                            <div className="sensor-container" key={String(sensor.device_id)}>
                            <h1>{sensor.name}</h1>
                            <div class="checkbox-wrapper-10">
                                <input checked={sensor.state==='ON'? true:false} type="checkbox" id="cb54" class="tgl tgl-flip"/>
                                <label for="cb54" data-tg-on="movimiento" data-tg-off="movimiento" class="tgl-btn"></label>
                              </div>
                            </div>

                            )}
                            </div>

                            );
                        })}
                    </section>

                </main>
            </Fragment>

        );
};
export default App;
```



### 16.7.3. Procesamiento de comandos de usuario

```
import psycopg2
from platypush.context import get_plugin
import time


trad = {"true":"on","false":"off"}
trad2 = {"true":"True","false":"False"}
focos_smartthings = [6,2,3]
last_id = 0
last_device_id = 0
#conversion de color para foco zigbee
def color_zigbee(x,y):
    if x >=0 and x<=30:
        return (0.640, 0.330)
    elif x >=31 and x<=65:
        return (0.560, 0.385)
    elif x >=66 and x<=97:
        return (0.440, 0.520)
    elif x >=98 and x<=170:
        return (0.220, 0.330)
    elif x >=171 and x<=229:
        return (0.150, 0.060)
    elif x >=230 and x<=272:
        return (0.280, 0.160)
    elif x >=273 and x<=360:
        return (0.320, 0.160)
```

```
    elif y == 0:
        return (0.310, 0.316)
#función para procesar comandos, reciben el id del dispositivo, la acción y color en caso aplique
def command_options(a, color,action):
    #para los focos se calcula el color como dos componentes entre la saturación y el tono
    #el estado pues es binario, true para encendido y false para apgado
    if (a==3):
        if color != "none":
            hue = float(color.split("(")[1].split(",")[0])*100/360.0
            sat = float(color.split("(")[1].split(",")[3].split(")")[0])*100
        if (action.lower() == "false"):
            get_plugin('smartthings').execute(device = "Foco entrada",capability="switch",command=trad[action.lower()])
        elif (action.lower() == "true"):
            get_plugin('smartthings').execute(device = "Foco entrada",capability="switch",command=trad[action.lower()])
            if color != "none":
                get_plugin('smartthings').execute(device = "Foco entrada",capability="colorControl",command="setColor",args=[{"hue":hue,
                    "saturation":sat}])
        else:
            get_plugin('smartthings').execute(device = "Foco entrada",capability="colorControl",command="setColor",args=[{"hue":hue,
                "saturation":sat}])
    elif (a==6):
        if color != "none":
            hue = float(color.split("(")[1].split(",")[0])*100/360.0
            sat = float(color.split("(")[1].split(",")[3].split(")")[0])*100
        if (action.lower() == "false"):
            get_plugin('smartthings').execute(device = "foco sala",capability="switch",command=trad[action.lower()])
        elif (action.lower() == "true"):
            get_plugin('smartthings').execute(device = "foco sala",capability="switch",command=trad[action.lower()])
            if color != "none":
                get_plugin('smartthings').execute(device = "foco sala",capability="colorControl",command="setColor",args=[{"hue":hue,
                    "saturation":sat}])
        else:
            get_plugin('smartthings').execute(device = "foco sala",capability="colorControl",command="setColor",args=[{"hue":hue,
                "saturation":sat}])
    elif (a==2):
        if color != "none":
            hue = float(color.split("(")[1].split(",")[0])*100/360.0
            sat = float(color.split("(")[1].split(",")[3].split(")")[0])*100
        if (action.lower() == "false"):
            get_plugin('smartthings').execute(device = "Foco de Dormitorio Paty",capability="switch",command=trad[action.lower()])
        elif (action.lower() == "true"):
            get_plugin('smartthings').execute(device = "Foco de Dormitorio Paty",capability="switch",command=trad[action.lower()])
            if color != "none":
                get_plugin('smartthings').execute(device = "Foco de Dormitorio
                    Paty",capability="colorControl",command="setColor",args=[{"hue":hue, "saturation":sat}])
        else:
            get_plugin('smartthings').execute(device = "Foco de Dormitorio Paty",capability="colorControl",command="setColor",args=[{"hue":hue,
                "saturation":sat}])
    elif (a==11):
        if color != "none":
            (hue,sat) = color_zigbee(int(color.split("(")[1].split(",")[0]),float(color.split("(")[1].split(",")[3].split(")")[0]))
        if (action.lower() == "false"):
            get_plugin('zigbee.mqtt').device_set(device=str('foco'),property="state",value=trad[action.lower()].upper())
        elif (action.lower() == "true"):
            get_plugin('zigbee.mqtt').device_set(device=str('foco'),property="state",value=trad[action.lower()].upper())
            if color != "none":
                get_plugin('zigbee.mqtt').device_set(device=str('foco'),property="color",value={"x":hue,"y":sat})
        else:
            get_plugin('zigbee.mqtt').device_set(device=str('foco'),property="color",value={"x":hue,"y":sat})
    elif (a==4):
```

```
        get_plugin('smartthings').execute(device = "Enchufe cafetera",capability="switch",command=trad[action.lower()])
    elif (a==5):
        get_plugin('smartthings').execute(device = "Enchufe Sala",capability="switch",command=trad[action.lower()])
    elif (a==7):
        get_plugin('smartthings').execute(device = "Enchufe Carlos",capability="switch",command=trad[action.lower()])
    elif (a==8):
        get_plugin('smartthings').execute(device = "Enchufe Javier",capability="switch",command=trad[action.lower()])
while(1):
    try:
        #se realiza la conexión a base de datos
        conn = psycopg2.connect(database = "house_sensors",
                            user = "postgres",
                            host= 'localhost',
                            password = "charlye72",
                            port = 5432)
        cur = conn.cursor()
        #se obtienen los últimos 5 comandos ingresados a la tabla
        cur.execute('SELECT * FROM casa_carlos_commands order by id desc limit 5;')
        rows = cur.fetchall()
        conn.commit()
        #se realiza para cada uno de los comandos
        for row in rows:
            #se revisa que el comando no haya sido procesado
            if row[5] == 0:
                #se ejecuta la acción correspondiente acorde con el comando
                command_options(row[1],row[4],row[3])
                #se actualiza el estado del comando a procesado
                cur.execute(f'update casa_carlos_commands set procesado=1 where id={row[0]};')
                #se valida nuevamente que sean comandos diferentes
                if last_id != row[0]:
                    last_id = row[0]
                    #Para los focos de smarthings, el color no viene en la respuesta al estado de la API
                    #de modo que se debe de actualizar el color ingresado manualmente acorde con el color
                    #ubicado en el comando
                    #Para el foco zigbee, el trigger no se levanta cuando el cambio es generado desde platypush
                    #por lo que igual se debe actualizar el estado basado en el comando
                    for i in range (1,14):
                        #se obtiene el estado del sensor
                        cur.execute(f'SELECT * FROM casa_carlos where device_id = {i} \
                                                    ORDER BY id DESC LIMIT 1;')
                        state = cur.fetchall()
                        for out in state:
                            if row[4] != "none" or row[1] not in focos_smartthings: #hacer la actualizacion solo cuando el comando sea de cambio
                                de color
                                #si es un cambio de estado el archivo de revisión de estado va a actualizar el estado en
                                #base de datos, normalmente solo pasa para los switches
                                #se revisa que la iteración corresponda al sensor que recibió el comando
                                if i == row[1]:
                                    #si no trae estado se ingresa el color y la fecha, con el estado anterior del foco
                                    if row[3] == "none":
                                        cur.execute(f" INSERT INTO casa_carlos(device_id,date,location,state,\
                                                    color,temperature,temperature_unit,humidity,name) VALUES({out[1]},\
                                                        '{row[2]}','{out[3]}','{out[4]}','{row[4]}','{out[6]}',\
                                                            '{out[7]}','{out[8]}','{out[9]}')");
                                    else: #si trae estado, se actualiza la fecha, el color el estado
                                        cur.execute(f" INSERT INTO casa_carlos(device_id,date,location,state,\
                                                    color,temperature,temperature_unit,humidity,name) VALUES({out[1]},\
                                                        '{row[2]}','{out[3]}','{trad2[row[3]]}','{row[4]}','{out[6]}',\
                                                            '{out[7]}','{out[8]}','{out[9]}')");
```

```
                            else: #si on es un sensor afectado, se copia el estado anterior con la fecha actual
                                cur.execute(f" INSERT INTO casa_carlos(device_id,date,location,state,\
                                            color,temperature,temperature_unit,humidity,name) VALUES({out[1]},\
                                                '{row[2]}','{out[3]}','{out[4]}','{out[5]}','{out[6]}',\
                                                    '{out[7]}','{out[8]}','{out[9]}')");


            conn.commit()
        time.sleep(1)
        cur.close()
        conn.close()
    except Exception as e:
        print(f"error: {e}")
```

## 16.8. Aprendizaje automático

### 16.8.1. Clustering con Kmeans

```
import psycopg2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sqlalchemy import create_engine
from sklearn.cluster import KMeans
import seaborn as sns
from sklearn.preprocessing import StandardScaler

# Create an engine instance
alchemyEngine   = create_engine('postgresql+psycopg2://postgres:charlye72@localhost:5432/house_sensors', pool_recycle=3600);

# Connect to PostgreSQL server
conn    = alchemyEngine.connect();
#se genera un data fram de pandas directamente de un query a la base de datos
data = pd.read_sql('SELECT * FROM casa_carlos order by id desc;', conn)
'''
antes de comenzar con el proceso de clustering se recomienda analizar la data para entrar en contexto
para ello, se muestran algunas funciones útiles
data.info()

data.head()

data_variables.describe()
'''
#normalmente se tienen columnas que no se van a utilizar para el análisis,
#que tienen valores útiles pero no son numéricos, con nommbres de
#columna no explicitos y demás situaciones como estas
#para ello se presentan las siguientes métodos para realizar diferentes
#ajustes como los mencionados

#método para eliminar una columna del data frame
data = data.drop(['id'], axis = 1)
```

```
#extracción de la hora y el dia de la columna de fecha
data['day'] = data['date'].dt.day_name()
data['hour'] = data['date'].dt.hour


#método para dummificar una variable cualitativa,
#la dummificación es solo una manera de hacerlo, hay otras
#location = pd.get_dummies(data['location'])
#day = pd.get_dummies(data['day'])

#método para cambiar el nombre de las columnas
#data.rename(columns={"nombre anterior":"nombre nuevo"}, inplace = True)




#otro método para pasar de una variable cualitativa a una cuantitativa
#para las variables que sean como el ejemplo mostrado, donde tiene sentido
#asignarle un valor numérico a uno cualitativo
#Dormitorio Carlos  Dormitorio Javier  cocina  entrada  sala
data['location'].replace(["Dormitorio Carlos","Dormitorio Javier","cocina","entrada","sala"],[0,1,2,3,4],inplace=True)

data['day'].replace(["Monday","Tuesday","Wednesday","Thursday","Friday","Saturday","Sunday"],[1,2,3,4,5,6,7],inplace=True)

data["state"].replace(["OFF","False","false"],0, inplace = True)
data["state"].replace(["none"],-1, inplace = True)
data["state"].replace(["ON","True","true"],1, inplace = True)
data["temperature"].replace("none",-2,inplace=True)
data["humidity"].replace("none",-3,inplace=True)

#correccion para una columna con valores NaN
data['state'] = data['state'].fillna(0)

#finalmente vamos a armar un dataframe unicamente con las columnas que nos interesan para el analisis
final_data_1 = pd.concat([data['day'],data['hour'],data['location'],data['state'],data['temperature'].astype(float),data['humidity']],axis=1)
print(final_data_1.isnull().sum())
#Le restamos a cada celda el valor minimo y divido entre el maximo y el minimo (el rango)
#todas tendran un valor minimo de 0 y un valor maximo de 1
#final_data_norm = (final_data - final_data.min())/(final_data.max() - final_data.min())

print(final_data_1)

final_data = final_data_1.dropna() #se borran los valores que sean NaN
#escalamos los datos
scaler = StandardScaler()
d_scaled = scaler.fit_transform(final_data_1)

final_data = pd.DataFrame(d_scaled)

wscc = [] #Within cluster sum of squares

for i in range(1,11):
    kmeans = KMeans(n_clusters = i, max_iter = 300)
    kmeans.fit(final_data) #Aplicamos KMeans a los datos
    wscc.append(kmeans.inertia_)
#para escoger la cantidad de clusters se debe observar cuando los valores de wscc
#reduzcan su taza de cambio
print(wscc)
```

```
clustering = KMeans(n_clusters = 3, max_iter = 300) #Creamos el modelo
clustering.fit(final_data) #Aplica el modelo a los datos

#Los resultados del clustering se guardan en labels_ dentro del modelo
data['KMeans_Cluster'] = clustering.labels_

print(data)
plt.figure(figsize=(11, 11))
var = ["name", "hour", "day","location"]
var_dict ={"name":"el Nombre del sensor", "hour":"la hora del dia", "day":"el día de la semana", "location":"la ubicación del sensor"}
for i in range(len(var)):
    g = sns.catplot(x=var[i], y="KMeans_Cluster", data=data, kind="box", height=5,aspect=3)
    g.fig.suptitle(f"Relación del cluster con {var_dict[var[i]]}", y=1.02)
    g.fig.set_size_inches(20, 6)
    g.savefig(f"cluser_{i}.png")

#Debido a que tenemos muchas características, es muy dificil graficar todas estas en un grafico
#Por eso utilizaros el analisis de componente Principal (PCA)

#Lo que hace este metodo es que reduce la cantidad de variables a analisar/visualizar
#Creando un conjunto de nuevas variables que representen lo mejor posible las variables
#originales

from sklearn.decomposition import PCA

pca = PCA(n_components = 2) #Grafico de dos dimensiones
pca_data = pca.fit_transform(final_data) #Obtenemos los componentes principales
pca_data_df = pd.DataFrame(data = pca_data, columns = ["Componente_1", "Componente_2"])
pca_nombres_data = pd.concat([pca_data_df, data[['KMeans_Cluster']]], axis = 1)
#Asignamos colores a los puntos en base al cluster al que pertenecen

fig = plt.figure(figsize = (6,6)) #Creamos figura de tamaño 6x6

ax = fig.add_subplot(1,1,1) #Le indico que solo creare un grafico en la figura
ax.set_xlabel("Componente 1", fontsize = 15)
ax.set_ylabel("Componente 2", fontsize = 15)
ax.set_title("Componentes Principales", fontsize = 20)

color_theme = np.array(['blue', 'green', 'orange','red'])

ax.scatter(x = pca_nombres_data.Componente_1, y = pca_nombres_data.Componente_2,
           c = color_theme[pca_nombres_data.KMeans_Cluster], s=50)
fig.savefig("significativo.png")
```

### 16.8.2. Clustering con GaussianMixture

```
import psycopg2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sqlalchemy import create_engine
from sklearn.cluster import KMeans
import seaborn as sns
from sklearn.mixture import GaussianMixture
from datetime import datetime
```

```
from sklearn.preprocessing import StandardScaler
from dateutil.relativedelta import relativedelta
# Create an engine instance
alchemyEngine   = create_engine('postgresql+psycopg2://postgres:charlye72@localhost:5432/house_sensors', pool_recycle=3600);

# Connect to PostgreSQL server
conn    = alchemyEngine.connect();
timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
timestamp_inf = (datetime.now() - relativedelta(months=1)).strftime("%Y-%m-%d %H:%M:%S")
#se genera un data fram de pandas directamente de un query a la base de datos
data = pd.read_sql(f"SELECT * FROM casa_carlos where date between '{timestamp_inf}' and '{timestamp}'order by id desc;", conn)
'''
antes de comenzar con el proceso de clustering se recomienda analizar la data para entrar en contexto
para ello, se muestran algunas funciones útiles
data.info()

data.head()

data_variables.describe()
'''
#normalmente se tienen columnas que no se van a utilizar para el análisis,
#que tienen valores útiles pero no son numéricos, con nommbres de
#columna no explicitos y demás situaciones como estas
#para ello se presentan las siguientes métodos para realizar diferentes
#ajustes como los mencionados

#método para eliminar una columna del data frame
data = data.drop(['id'], axis = 1)

#extracción de la hora y el dia de la columna de fecha
data['day'] = data['date'].dt.day_name()
data['hour'] = data['date'].dt.hour


#método para dummificar una variable cualitativa,
#la dummificación es solo una manera de hacerlo, hay otras
#location = pd.get_dummies(data['location'])
#day = pd.get_dummies(data['day'])

#método para cambiar el nombre de las columnas
#data.rename(columns={"nombre anterior":"nombre nuevo"}, inplace = True)




#otro método para pasar de una variable cualitativa a una cuantitativa
#para las variables que sean como el ejemplo mostrado, donde tiene sentido
#asignarle un valor numérico a uno cualitativo
#Dormitorio Carlos  Dormitorio Javier  cocina  entrada  sala
data['location'].replace(["Dormitorio Carlos","Dormitorio Javier","cocina","entrada","sala"],[0,1,2,3,4],inplace=True)

data['day'].replace(["Monday","Tuesday","Wednesday","Thursday","Friday","Saturday","Sunday"],[1,2,3,4,5,6,7],inplace=True)

data["state"].replace(["OFF","False"],0, inplace = True)
data["state"].replace(["none"],-1, inplace = True)
data["state"].replace(["ON","True"],1, inplace = True)
data["temperature"].replace("none",-2,inplace=True)
data["humidity"].replace("none",-3,inplace=True)

```

```
#correccion para una columna con valores NaN
data['state'] = data['state'].fillna(0)

#finalmente vamos a armar un dataframe unicamente con las columnas que nos interesan para el analisis
final_data_1 = pd.concat([data['day'],data['hour'],data['location'],data['state'],data['temperature'].astype(float),data['humidity']],axis=1)
print(final_data_1.isnull().sum())
#Le restamos a cada celda el valor minimo y divido entre el maximo y el minimo (el rango)
#todas tendran un valor minimo de 0 y un valor maximo de 1
#final_data_norm = (final_data - final_data.min())/(final_data.max() - final_data.min())

print(final_data_1)

final_data = final_data_1.dropna() #se borran los valores que sean NaN
#escalamos los datos
scaler = StandardScaler()
d_scaled = scaler.fit_transform(final_data_1)

final_data = pd.DataFrame(d_scaled)
wscc = [] #Within cluster sum of squares
gmm = GaussianMixture(n_components=3)

# Ajustar el modelo a tus datos
gmm.fit(final_data)

# Obtener las etiquetas de los clusters asignados a cada muestra
labels = gmm.predict(final_data)

#Los resultados del clustering se guardan en labels_ dentro del modelo
data['GAUS_Cluster'] = labels

print(data)
plt.figure(figsize=(11, 11))
var = ["name", "hour", "day","location"]
var_dict ={"name":"el Nombre del sensor", "hour":"la hora del dia", "day":"el día de la semana", "location":"la ubicación del sensor"}
for i in range(len(var)):
    g = sns.catplot(x=var[i], y="GAUS_Cluster", data=data, kind="box", height=5,aspect=3)
    g.fig.suptitle(f"Relación del cluster con {var_dict[var[i]]}", y=1.02)
    g.fig.set_size_inches(20, 6)
    g.savefig(f"cluser_gauss_{i}.png")

#Debido a que tenemos muchas características, es muy dificil graficar todas estas en un grafico
#Por eso utilizaros el analisis de componente Principal (PCA)

#Lo que hace este metodo es que reduce la cantidad de variables a analisar/visualizar
#Creando un conjunto de nuevas variables que representen lo mejor posible las variables
#originales

from sklearn.decomposition import PCA

pca = PCA(n_components = 2) #Grafico de dos dimensiones
pca_data = pca.fit_transform(final_data) #Obtenemos los componentes principales
pca_data_df = pd.DataFrame(data = pca_data, columns = ["Componente_1", "Componente_2"])
pca_nombres_data = pd.concat([pca_data_df, data[['GAUS_Cluster']]], axis = 1)
#Asignamos colores a los puntos en base al cluster al que pertenecen

fig = plt.figure(figsize = (6,6)) #Creamos figura de tamaño 6x6

ax = fig.add_subplot(1,1,1) #Le indico que solo creare un grafico en la figura
ax.set_xlabel("Componente 1", fontsize = 15)
ax.set_ylabel("Componente 2", fontsize = 15)
```

```
ax.set_title("Componentes Principales", fontsize = 20)

color_theme = np.array(['blue', 'green', 'orange'])

ax.scatter(x = pca_nombres_data.Componente_1, y = pca_nombres_data.Componente_2,
           c = color_theme[pca_nombres_data.GAUS_Cluster], s=50)
fig.savefig("significativo_gauss.png")
```

### 16.8.3. Predicción logística

```
import psycopg2
from datetime import datetime
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sqlalchemy import create_engine
import seaborn as sns
from collections import Counter
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score
from dateutil.relativedelta import relativedelta
coefs = []
bias = []
accus=[]
# Create an engine instance
alchemyEngine   = create_engine('postgresql+psycopg2://postgres:charlye72@localhost:5432/house_sensors', pool_recycle=3600);
#timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
#timestamp_inf = (datetime.now() - relativedelta(months=1)).strftime("%Y-%m-%d %H:%M:%S")
timestamp = "2023-11-30 23:59:59"
timestamp_inf = "2023-11-01 00:00:00"
# Connect to PostgreSQL server
conn    = alchemyEngine.connect();
#se genera un data fram de pandas directamente de un query a la base de datos
data = pd.DataFrame()
df = []
df.append(pd.read_sql(f"SELECT date FROM casa_carlos where device_id = 1 and date BETWEEN '{timestamp_inf}' and '{timestamp}' order by id desc;",
    conn))
df.append(pd.read_sql(f"SELECT temperature FROM casa_carlos where device_id = 1 and date BETWEEN '{timestamp_inf}' and '{timestamp}' order by id
    desc;", conn))
df.append(pd.read_sql(f"SELECT humidity FROM casa_carlos where device_id = 1 and date BETWEEN '{timestamp_inf}' and '{timestamp}' order by id
    desc;", conn))
for i in range (2,12):
    df.append(pd.read_sql(f"SELECT state FROM casa_carlos where device_id = {i} and date BETWEEN '{timestamp_inf}' and '{timestamp}' order by id
        desc;", conn))

for i in range (len(df)):
    data[f"device_{i}"] = df[i]
'''
antes de comenzar con el proceso de clustering se recomienda analizar la data para entrar en contexto
para ello, se muestran algunas funciones útiles
data.info()

data.head()

```

```
data_variables.describe()
'''
#normalmente se tienen columnas que no se van a utilizar para el análisis,
#que tienen valores útiles pero no son numéricos, con nommbres de
#columna no explicitos y demás situaciones como estas
#para ello se presentan las siguientes métodos para realizar diferentes
#ajustes como los mencionados

#método para eliminar una columna del data frame
#data = data.drop(['id'], axis = 1)

#extracción de la hora y el dia de la columna de fecha
data['day'] = data['device_0'].dt.day_name()
data['hour'] = data['device_0'].dt.hour


#método para dummificar una variable cualitativa,
#la dummificación es solo una manera de hacerlo, hay otras
#location = pd.get_dummies(data['location'])
#day = pd.get_dummies(data['day'])

#método para cambiar el nombre de las columnas
#data.rename(columns={"nombre anterior":"nombre nuevo"}, inplace = True)




#otro método para pasar de una variable cualitativa a una cuantitativa
#para las variables que sean como el ejemplo mostrado, donde tiene sentido
#asignarle un valor numérico a uno cualitativo
#Dormitorio Carlos  Dormitorio Javier  cocina  entrada  sala
#data['location'].replace(["Dormitorio Carlos","Dormitorio Javier","cocina","entrada","sala"],[0,1,2,3,4],inplace=True)

data['day'].replace(["Monday","Tuesday","Wednesday","Thursday","Friday","Saturday","Sunday"],[1,2,3,4,5,6,7],inplace=True)
data = data[data!='none'].dropna()
for i in range(3,13):
    data[f"device_{i}"].replace(["OFF","False","false"],0, inplace = True)
    data[f"device_{i}"].replace(["ON","True","true"],1, inplace = True)

#correccion para una columna con valores NaN
#data['state'] = data['state'].fillna(0)

#finalmente vamos a armar un dataframe unicamente con las columnas que nos interesan para el analisis
'''
final_data =
    pd.concat([data['device_id'],data['day'],data['hour'],data['location'],data['state'],data['temperature'].astype(float),data['humidity']],axis=1)
devices = list(final_data['device_id'].unique())
for i in devices:
    final_data[f'device_{i}'] = np.where((final_data['device_id']==i) & (final_data['state']==1), 1, 0)
print(final_data)
print(final_data.isnull().sum())
'''

#Le restamos a cada celda el valor minimo y divido entre el maximo y el minimo (el rango)
#todas tendran un valor minimo de 0 y un valor maximo de 1
#final_data_norm = (final_data - final_data.min())/(final_data.max() - final_data.min())
#data['Y'] = np.where((final_data['location']==2) & (final_data['device_id']==2) & (final_data['state']==1), 1, 0)
print(data)
for i in range (3,11):
```

```
    Y = data[f'device_{i}']
    final_data = data.drop([f'device_{i}','device_0'], axis =1)
    if 'Prediction' in final_data:
        final_data = final_data.drop(['Prediction'], axis =1)
    if 'Prediction_proba' in final_data:
        final_data = final_data.drop(['Prediction_proba'], axis =1)
    scaler = StandardScaler()

    #Estandirazar los datos de X y los almacenare en una variable que se llame d_scaled.
    #Ajustar y transformar los datos de una vez

    d_scaled = scaler.fit_transform(final_data)

    data_scaled = pd.DataFrame(d_scaled)

    #se comienza con la regresión

    #Divide los datos en conjunto de entrenamiento y Prueba

    #d_scaled Es el DF con las caracteristicas ya normalizadas

    #test_size — Le indicamos que queremos un 20% de los datos para testear

    #Random_state — Nos garantiza que los mismo conjuntos de entrenamiento y testeo se generan en cada corrida

    X_train, X_test, y_train, y_test = train_test_split(data_scaled, Y, test_size = 0.20, random_state = 42)

    # C— controla la regularizacion del modelo

    model = LogisticRegression(C = 0.1, max_iter = 500)

    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    print(y_pred)

    predicted_proba = model.predict_proba(X_test)

    positive_proba = predicted_proba[:, 1]

    print(positive_proba)

    #se obtienen las metricas de nuestro modelo de ML
    print(f'Weight Coefficient : {model.coef_}')
    print(f'Bias : {model.intercept_}')
    coefs.append(model.coef_)
    bias.append(model.intercept_)
    print(f'Test accuracy: {model.score(X_test, y_test)}')
    print(f'Train accuracy: {model.score(X_train, y_train)}')



    #se crea la matriz de confusión
    df = pd.DataFrame(confusion_matrix(y_test, y_pred), columns = ['Predicted Positive', 'Predicted Negative'],
                      index= ['Actual Positive', 'Actual Negative'])



    print("Accuracy : ", accuracy_score(y_test, y_pred) )
```

```
    accus.append(accuracy_score(y_test, y_pred))
    #se obtiene la predicción aplicada sobre el df inicial
    final_prov = model.predict_proba(data_scaled)
    #extraemos solo la probabilidad positiva
    final_positiv_proba = final_prov[:,1]
    print(final_positiv_proba)

    #obtenemos la predicción dada por el modelo
    final_tags = model.predict(data_scaled)
    print(final_tags)
    #se añaden al df la predicción y la probabilidad positiva numérica
    data["Prediction"] = final_tags
    data["Prediction_proba"] = final_positiv_proba
    #se revisa que hayan predicciones positivas
    print(data[(data['Prediction']==1)])
#se obtiene el modelo con mayor precisión
index = accus.index(max(accus))
print(index+2)
print(coefs[index])
print(bias[index])
#se hace una conexión con la base de datos
conn = psycopg2.connect(database = "house_sensors",
                        user = "postgres",
                        host= 'localhost',
                        password = "charlye72",
                        port = 5432)

cur = conn.cursor()
#se insertan los datos del modelo obtenido en la tabla de los modelos
timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
cur.execute(f"insert into casa_carlos_ML(device_id,date,coef,bias,accuracy)
    VALUES({index+2},'{timestamp}','{str(coefs[index])[2:len(str(coefs[index]))-2]}',{bias[index][0]},{accus[index]});")

conn.commit()
# se cierra la conexión con la base de datos
cur.close()
conn.close()
```

## 16.9. Microcontroladores

### 16.9.1. Arduino nano 33 IoT

```
#include <ArduinoMqttClient.h>
#include <WiFiNINA.h>
char ssid[] = "NETGEAR02";        //SSID
char pass[] = "rockysquash228";    //Contraseña

WiFiClient wifiClient; //instancia del cliente wifi
MqttClient mqttClient(wifiClient); //instancia del cliente mqtt

const char broker[] = "192.168.1.13"; //broker mqtt
```

```
int        port     = 1883; //puerto mqtt
const char topic[]  = "mcus";  //canal mqtt

const int LEDPin= 13;
const int PIRPin= 2;
bool last_value = false;
bool last_off = false;
bool last_on = false;
bool last_send = false;
void setup()
{
  pinMode(LEDPin, OUTPUT);
  pinMode(PIRPin, INPUT);
  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }
  // tratar de conectarse a la red wifi
  Serial.print("Intentando conectarse a la SSID: ");
  Serial.println(ssid);
  WiFi.begin(ssid, pass);
  while (WiFi.status() != WL_CONNECTED) {
    // si falla, intentar reconectar
    Serial.print(".");
    delay(5000);
  }

  Serial.println("Conexión exitosa");
  Serial.println();

  Serial.print("Intentando conectarse al MQTT broker: ");
  Serial.println(broker);

  if (!mqttClient.connect(broker, port)) {
    Serial.print("Falló la conexión con el broker Error code = ");
    Serial.println(mqttClient.connectError());

    while (1);
  }

  Serial.println("Conexión con el MQTT broker exitosa");
  Serial.println();

  // colocar el "callback" para recibir mensjes MQTT
  //mqttClient.onMessage(onMqttMessage);

  // subscribir el cliente al tema mqtt
  mqttClient.subscribe(topic);

  // desuscribirse el cliente al tema mqtt
  // mqttClient.unsubscribe(topic);
}

void loop()
{
  mqttClient.poll();
  int value= digitalRead(PIRPin);
  //indicador led de movimiento
  if (value == HIGH)
  {
```

```
    digitalWrite(LEDPin, HIGH);
    delay(50);
    digitalWrite(LEDPin, LOW);
    delay(50);
    last_value = true;
  }
  else
  { //implementación de un antirebote para la detección de la vibración

    if (last_value == true){

      digitalWrite(LEDPin, LOW);
      last_value = false;
      mqttClient.beginMessage(topic);
      mqttClient.print("13,move_sensor_carlos,ON,Dormitorio Carlos");
      mqttClient.endMessage();
      delay(5000);
      mqttClient.beginMessage(topic);
      mqttClient.print("13,move_sensor_carlos,OFF,Dormitorio Carlos");
      mqttClient.endMessage();
    }
  }
  delay(2000);
}
void onMqttMessage(int messageSize) {
  // Imprimir tema y tamaño del mensaje
  Serial.println("Received a message with topic '");
  Serial.print(mqttClient.messageTopic());
  Serial.print("', length ");
  Serial.print(messageSize);
  Serial.println(" bytes:");

  // imprimir el mensaje publicado
  while (mqttClient.available()) {
    Serial.print((char)mqttClient.read());
  }
  //fin de linea
  Serial.println();
  Serial.println();
}
```

### 16.9.2. ESP32

```
#include <ArduinoMqttClient.h>
#include <WiFi.h>
char ssid[] = "NETGEAR02";         // SSID
char pass[] = "rockysquash228";    // password

WiFiClient wifiClient; //instancia del cliente wifi
MqttClient mqttClient(wifiClient); //instancia del cliente mqtt

const char broker[] = "192.168.1.13"; //IP del broker mqtt
int        port     = 1883;//puerto mqtt
const char topic[]  = "mcus";//tema mqtt
const int LDR = 34;  //pin donde se conecta el LDR
```

```
String statu = "OFF"; //Estado inicial del sensor de liz
int LDRValue = 0; //Variable para guardar la lectura analógica actual
int lastLDRValue = 0;//Variable para guardar la lectura analógica anterior
void setup() {
  Serial.begin(115200);
  while (!Serial) {
    ; // esperar que se conecte el puerto serial
  }
  // tratar de conectarse a la red wifi
  Serial.print("Intentando conectarse a la SSID: ");
  Serial.println(ssid);
  WiFi.begin(ssid, pass);
  while (WiFi.status() != WL_CONNECTED) {
    // si falla, intentar reconectar
    Serial.print(".");
    delay(5000);
  }

  Serial.println("Conexión exitosa");
  Serial.println();

  Serial.print("Intentando conectarse al MQTT broker: ");
  Serial.println(broker);

  if (!mqttClient.connect(broker, port)) {
    Serial.print("Falló la conexión con el broker Error code = ");
    Serial.println(mqttClient.connectError());

    while (1);
  }

  Serial.println("Conexión con el MQTT broker exitosa");
  Serial.println();

  // colocar el "callback" para recibir mensjes MQTT
  //mqttClient.onMessage(onMqttMessage);

  // subscribir el cliente al tema mqtt
  mqttClient.subscribe(topic);

  // desuscribirse el cliente al tema mqtt
  // mqttClient.unsubscribe(topic);
}
void loop() {
  mqttClient.poll(); //polear actualizaciones mqtt
  // leer valor del LDR
  LDRValue = analogRead(LDR);
  //Revisar si hubo un cambio de estado significativo en la lectura del sensor
  Serial.println(LDRValue);
  if (abs(LDRValue - lastLDRValue)>50){
    mqttClient.beginMessage(topic);//iniciar mensaje
    if (LDRValue < 2800){ //asignar estado de la luz acorde con la lectura
    statu = "ON";
    }else{
    statu = "OFF";
    }
    //publicar mensaje MQTT
    mqttClient.print("12,light_sensor_carlos,"+statu+",Dormitorio Carlos");
    mqttClient.endMessage(); //Finalizar mensaje
    lastLDRValue = LDRValue; //actualizar el valor anterior de la lectura
```

```
  }
  delay(500);
}
void onMqttMessage(int messageSize) {
  // Imprimir tema y tamaño del mensaje
  Serial.println("Received a message with topic '");
  Serial.print(mqttClient.messageTopic());
  Serial.print("', length ");
  Serial.print(messageSize);
  Serial.println(" bytes:");

  // imprimir el mensaje publicado
  while (mqttClient.available()) {
    Serial.print((char)mqttClient.read());
  }
  //fin de linea
  Serial.println();
  Serial.println();
}
```