Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería Mecánica Eléctrica

DISEÑO DE PROTOCOLO DE INTERNET DE LAS COSAS BASADO EN HTTP PARA CONTROL DE ACTUADORES Y SENSORES EN DOMÓTICA

Eduardo José Muralles Bautista
Asesorado por el Ing. Byron Odilio Arrivillaga Méndez

Guatemala, marzo de 2020

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

DISEÑO DE PROTOCOLO DE INTERNET DE LAS COSAS BASADO EN HTTP PARA CONTROL DE ACTUADORES Y SENSORES EN DOMÓTICA

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

EDUARDO JOSÉ MURALLES BAUTISTA
ASESORADO POR EL ING. BYRON ODILIO ARRIVILLAGA MÉNDEZ

AL CONFERÍRSELE EL TÍTULO DE

INGENIERO ELECTRÓNICO

GUATEMALA, MARZO DE 2020

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANA	Inga. Aurelia Anabela Cordova Estrada
VOCAL I	Ing. José Francisco Gómez Rivera
VOCAL II	Ing. Mario Renato Escobedo Martínez
VOCAL III	Ing. José Milton de León Bran
VOCAL IV	Br. Christian Moisés de la Cruz Leal
VOCAL V	Br. Kevin Armando Cruz Lorente
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Pedro Antonio Aguilar Polanco
EXAMINADOR	Ing. Walter Giovanni Alvarez Marroquín
EXAMINADOR	Ing. José Aníbal Silva de los Angeles
EXAMINADOR	Ing. Helmunt Federico Chicol Cabrera
SECRETARIA	Inga. Lesbia Magalí Herrera López

HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

DISEÑO DE PROTOCOLO DE INTERNET DE LAS COSAS BASADO EN HTTP PARA CONTROL DE ACTUADORES Y SENSORES EN DOMÓTICA

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería Mecánica Eléctrica, con fecha 15 de mayo de 2019.

Eduardo José Muralles Bautista

Guatemala 24 de octubre, 2019

Ingeniero Armando Alonso Rivera Carrillo
Director de Escuela de Ingeniería Mecánica Eléctrica
Facultad de Ingeniería
Universidad de San Carlos de Guatemala

Por medio de la presente me permito informarle que he procedido a revisar el trabajo de graduación titulado **"DISEÑO DE PROTOCOLO DE INTERNET DE LAS COSAS BASADO EN HTTP PARA CONTROL DE ACTUADORES Y SENSORES EN DOMÓTICA"** elaborado por el estudiante Eduardo José Muralles Bautista quien se identifica con el número de DPI 2432 48237 0101 y registro académico 201214113. A su vez, quiero mencionar que el mismo cumple los objetivos trazados de acuerdo con el protocolo presentado, por lo que lo doy por APROBADO. De tal manera, se solicita darle trámite correspondiente.

Atentamente,

Byron Arrivillaga Méndez
Ingeniero Electrónico
Colegiado 5217

Ing. Byron Odilio Arrivillaga Méndez
Colegiado No. 5217

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERIA

Guatemala, 4 de noviembre de 2019

Señor Director
Armando Alonso Rivera Carrillo
Escuela de Ingeniería Mecánica Eléctrica
Facultad de Ingeniería, USAC

Estimado Señor Director:

Por este medio me permito dar aprobación al Trabajo de Graduación titulado **DISEÑO DE PROTOCOLO DE INTERNET DE LAS COSAS BASADO EN HTTP PARA CONTROL DE ACTUADORES Y SENSORES EN DOMÓTICA,** desarrollado por el estudiante **Eduardo José Muralles Bautista**, ya que considero que cumple con los requisitos establecidos.

Sin otro particular, aprovecho la oportunidad para saludarlo.

Atentamente,

ID Y ENSEÑAD A TODOS

Ing. Julio César Solares Peñate
Coordinador de Electrónica

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCION ESCUELA DE INGENIERIA MECANICA ELECTRICA
FACULTAD DE INGENIERIA

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERÍA

REF. EIME 77. 2019.

El Director de la Escuela de Ingeniería Mecánica Eléctrica, después de conocer el dictamen del Asesor, con el Visto bueno del Coordinador de Área, al trabajo de Graduación del estudiante: EDUARDO JOSÉ MURALLES BAUTISTA titulado: DISEÑO DE PROTOCOLO DE INTERNET DE LAS COSAS BASADO EN HTTP PARA CONTROL DE ACTUADORES Y SENSORES EN DOMÓTICA , procede a la autorización del mismo.

Ing. Armando Alonso Rivera Carrillo

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCION ESCUELA DE INGENIERIA MECANICA ELECTRICA
FACULTAD DE INGENIERIA

GUATEMALA, 14 DE NOVIEMBRE 2019.

Universidad de San Carlos
De Guatemala

Facultad de Ingeniería
Decanato

Ref. DTG.097-2020

La Decana de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería Mecánica Eléctrica, al trabajo de graduación titulado: **DISEÑO DE PROTOCOLO DE INTERNET DE LAS COSAS BASADO EN HTTP PARA CONTROL DE ACTUADORES Y SENSORES EN DOMÓTICA,** presentado por el estudiante universitario: **Eduardo José Muralles Bautista,** y después de haber culminado las revisiones previas bajo la responsabilidad de las instancias correspondientes, se autoriza la impresión del mismo.

IMPRÍMASE.

Inga. Aurelia Anabela Cordova Estrada
Decana

Guatemala, marzo de 2020

AACE/asga

ACTO QUE DEDICO A:

Dios Que me ha dado la capacidad para lograr mis objetivos y ha permitido que este momento suceda.

Mis padres David Eduardo Muralles y Ana Rebeca Bautista. Por su apoyo incondicional durante toda mi formación. Que este logro los llene de orgullo, pues parte de él les pertenece.

Mi hermana María Rebeca Muralles. Siempre ha ido un paso adelante motivándome a perseverar para alcanzar mis metas.

Mi hermana Ana Lorena Muralles. Que este logro se convierta en una inspiración y demostración de que todo se puede lograr con esfuerzo y constancia.

Mi novia

Nanci Paola Contreras. Por apoyarme cada vez que mi fuerza de voluntad se debilitó en los momentos más difíciles de la carrera.

AGRADECIMIENTOS A:

Universidad de San Carlos de Guatemala	Por ser la principal fuente de conocimiento académico durante mi carrera y haberme dado lecciones que van más allá de la academia.
Mis amigos de la Facultad	Kevin Monterroso, Miguel Rodas y Paulo Diéguez, con quienes hice equipo y logré completar varios proyectos académicos.
Mi tío	Juan Francisco Bautista, por darme su ayuda cuando más lo necesitaba
Mis buenos catedráticos	Porque compartieron su conocimiento sin egoísmo aun sin recibir más recompensa que la satisfacción de que alguien aprendiera.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

TABLAS

LISTA DE SÍMBOLOS

Símbolo	Significado
“	Pulgada
°	Grado
°C	Grado centígrado
µm	Micrómetro
A	Amperio
cm	Centímetro
dBm	Decibel milivatio
Ghz	Gigahercio
kΩ	Kiloohm
kHz	Kilohercio
m^2	Metro cuadrado
mA	Miliamperio
mil	Milésima de pulgada
mm	Milímetro
ms	Milisegundo
nm	Nanómetro
s	Segundo
V	Voltio
W	Vatio

GLOSARIO

Arduino	Compañía de software y hardware libres.
ASCII	Código estándar americano para el intercambio de información.
AWS	Servicios web que ofrece la compañía Amazon.
Banda ISM	Banda del espectro electromagnético utilizada por industrias y científicos para la investigación, libre de regulaciones comerciales.
Cableado estructurado	Tendido de cables en el interior de un edificio con el objetivo de implementar una red.
Código C	Lenguaje de programación compilado de propositito general.
Conector microUSB	Estándar de conector para tecnología USB de dimensiones reducidas.
Dato binario	Referente a un dato que únicamente puede tomar uno de dos valores conocidos.

Denegación de servicio	Impedimento de un servidor de presentar un servicio por una saturación en su capacidad de procesamiento de peticiones.
Dirección IP	Identificador, dentro del protocolo de Internet, de un dispositivo como parte de una red.
Encriptación asimétrica	Método criptográfico que utiliza dos llaves para cifrar los mensajes que se transmiten por el medio.
Endpoint	Es un punto de conexión entre diferentes servicios web.
ESP8266	Microcontrolador con conectividad wifi.
FHSS	Espectro ensanchado por salto de frecuencia, es la tecnología que utiliza Bluetooth que consiste en enviar la información con saltos por diferentes canales.
Firewall	Dispositivo de seguridad que permite filtrar la comunicación a través de los puertos de un dispositivo.
HTTP	Protocolo de transferencia de hipertexto.
IDE	Entorno integrado de desarrollo que se utiliza para programar.

IEEE	Instituto de Ingenieros Eléctricos y Electrónicos, es una entidad que se encarga de la estandarización de tecnologías de ingeniería.
IEFT	Fuerza de trabajo de ingeniería de internet, es una organización internacional que propone estándares para el crecimiento y seguridad del internet.
Inteligencia artificial	Aparente inteligencia que presenta una máquina para la resolución de un problema o ejecución de una acción.
Internetwork	Red conformada por varias redes de menor extensión.
Interoperabilidad	Capacidad de los dispositivos de interactuar entre ellos mediante el uso de estándares en la comunicación.
IoT	Internet de las cosas, tecnología que consiste en la comunicación de máquinas, dispositivos y humanos a través de la red.
ITU	Unión Internacional de Telecomunicaciones, es un ente especializado de la Organización de las Naciones Unidas para las tecnologías de información y comunicación.
JSON	Notación de objeto de JavaScript, es un formato de texto para el intercambio de datos entre aplicaciones.

LAN	Red de área local, es aquella red que tiene una extensión geográfica limitada a una casa o un edificio.
MD5	Algoritmo de criptografía de 128 bits.
Memoria *flash*	Es un tipo de memoria que permite la lectura y escritura de múltiples posiciones de memoria en una sola operación.
Mercado horizontal	Referente a la comercialización de dispositivos que permiten interoperabilidad y control centralizado.
Mercado vertical	Referente a la comercialización de dispositivos electrónicos que no permiten la interacción entre ellos.
Metadatos	Información que describe el contenido de un recurso.
Modelo OSI	Modelo en capas de estructuración de la red creado por la Organización Internacional de Normalización.
Multimedia	Referente a medios de comunicación audiovisuales.
Multiplexor	Dispositivo encargado de conmutar la información que se envía a través de un canal de comunicación.
P2P	Igual a igual, es un modelo de comunicación descentralizado donde todos los dispositivos pueden actuar como clientes y servidores.

Perro guardián	Es un temporizador integrado en algunos microcontroladores que reinicia el dispositivo cuando el procesador se encuentra en un bucle sin salida.
Proxy	Dispositivo intermedio entre un servidor y un cliente que reenvía las peticiones y respuestas.
Puerto	Canal lógico de comunicación de un dispositivo de red.
Python	Lenguaje de programación interpretado orientado a favorecer la creación de código legible.
PythonAnywhere	Servidor en internet que permite la implementación de servicios web programados en Python.
RAM	Memoria de acceso aleatorio.
Resistencia *pull-up*	Es una resistencia conectada entre la salida del circuito y el voltaje de alimentación para evitar los valores lógicos indeterminados.
REST	Transferencia de estado representacional, es un estilo de arquitectura de software para sistemas de hipermedia.
RF	Radiofrecuencia.

RIABAP	Red inalámbrica de área extensa, utilizando baja o muy baja potencia.
Router	Dispositivo de red que tiene como función principal intercomunicar diversas redes.
Servidor web	Dispositivo especializado que presta un servicio de HTTP.
SIT	Superintendencia de Telecomunicaciones de Guatemala.
Smartphone	Teléfono móvil que tiene capacidad de producción y reproducción multimedia y acceso a internet.
SPIFFS	Interfaz de periféricos serie de sistema de archivos en memoria *flash*, es un sistema de archivos que se instala en la memoria *flash*.
SSID	Identificador de conjunto de servicio, es una secuencia de caracteres que identifica una red inalámbrica.
Switch	Dispositivo concentrador de red que conmuta la comunicación entre varios dispositivos finales e intermedios.
Tablet	Dispositivo móvil que presenta una pantalla amplia como único método de entrada táctil.

Teleasistencia	Servicio de asistencia remota a través de medios de comunicación en tiempo real.
Texto plano	Texto que no contiene formato o encriptación, únicamente información legible.
TIC	Tecnologías de la información y comunicación.
UART	Transmisor-receptor asíncrono universal.
URI	Identificador de recurso uniforme, es una cadena de texto que identifica un recurso dentro de la red de forma unívoca.
URL	Localizador de recurso uniforme, es un identificador de la ubicación de un recurso dentro de la red.
URN	Nombre de recurso uniforme, es un identificador de un recurso en la red que no indica la ubicación.
VoIP	Voz sobre protocolo de internet, consiste en la transmisión de voz digitalizada a través de la red.
Voltaje TTL	Se refiere a los valores de voltaje que utiliza la tecnología de lógica de transistor a transistor.
W3C	Consorcio de la *World Wide Web*, es una entidad que genera recomendaciones y estándares para el crecimiento de la *World Wide Web*.

Wifi	Tecnología de intercomunicación inalámbrica entre dispositivos basada en el estándar IEEE 802.11.
World Wide Web	Sistema de distribución de documentos de hipertexto e hipermedia a través de Internet.
XML	Lenguaje de marcado extensible, es un formato estándar para el intercambio de información estructurada.

RESUMEN

El siguiente trabajo de graduación presenta una descripción del concepto, terminología, componentes y las consideraciones dentro de un sistema de domótica así como la definición, origen y requisitos de la tecnología denominada Internet de las Cosas y la forma en que se relaciona con la domótica. También se incluye una descripción del protocolo de transferencia de hipertexto que se toma como base.

En los capítulos posteriores a la teoría, se describe un protocolo de Internet de las Cosas genérico basado en HTTP con estándar REST orientado al uso de sensores y actuadores en un sistema de domótica. La descripción incluye las reglas, terminología, procesos, metadatos y topología del protocolo propuesto.

Por último, se expone un ejemplo de la implementación del protocolo en un circuito de conmutación de iluminación. Se detalla cada paso de la implementación para que cualquiera que tenga acceso a este documento pueda replicar el ejemplo que se muestra o adaptarlo a su necesidad realizando las modificaciones que considere.

OBJETIVOS

General

Describir los procedimientos, reglas y formato de datos de un protocolo de aplicación de IoT, para el control de actuadores y sensores en domótica basado en el protocolo HTTP.

Específicos

1. Definir el procedimiento para agregar y remover elementos de la red de dispositivos que trabajan en conjunto.

2. Especificar el modelo y los procesos de comunicación que utilizarán los dispositivos.

3. Detallar el formato de los datos que serán enviados entre los dispositivos para la comunicación de su estado.

4. Proponer las cabeceras necesarias para la comunicación de los dispositivos.

INTRODUCCIÓN

La tecnología avanza cada año y las máquinas son cada vez más capaces de realizar tareas automatizadas en servicio de los seres humanos, brindándoles más confort, seguridad y accesibilidad. Los hogares, y espacios habitables en general, integran dispositivos que se interconectan, a través de tecnologías de red, expandiendo las capacidades de automatización, control y monitoreo por parte de un usuario. El Internet de las Cosas se integra, entonces, con la domótica para mejorar la experiencia del usuario de un espacio habitable.

Los fabricantes de equipos domésticos con frecuencia utilizan protocolos propietarios para la intercomunicación de sus productos, sin embargo, mientas el mercado de dispositivos conectables en red aumenta y los productores de tecnología doméstica crecen en número, los protocolos propietarios no son una opción viable para favorecer la interconexión de los equipos en el hogar, requisito importante del Internet de las Cosas. Es necesario el uso de protocolos basados en estándares existentes, considerablemente distribuidos en el mundo y adaptados a los requerimientos de un sistema de domótica, para que puedan ser integrados por cualquier fabricante que utilice tecnologías de conexión en red estándar.

El propósito de este trabajo de graduación es el diseño de un protocolo de Internet de las Cosas para la comunicación máquina-máquina basado en estándares que permitan la integración de aplicaciones existentes e interfaces de usuario. Para esto se ha utilizado como base el protocolo de transferencia de hipertexto, un protocolo habilitado en una gran cantidad de equipos con capacidad de conexión en red y base de la *World Wide Web*, en conjunto con el

estándar de transferencia de estado representacional, un estilo de arquitectura de software para sistemas hipermedia que cada vez es más utilizado en la web.

Para entender el funcionamiento del protocolo presentado en este trabajo es necesario introducir al lector en una base teórica sobre campo de la automatización orientada a espacios habitables y sobre la tecnología de interconexión de dispositivos a través de la red, así como es importante que se presente un resumen sobre las especificaciones del protocolo base y el estándar de arquitectura de software que han sido considerados para la descripción del protocolo propuesto.

1. DOMÓTICA E IOT

1.1. Domótica

Domótica es la rama de la automatización que se aplica a un espacio habitable de cualquier tipo, su objetivo es cubrir las necesidades de los usuarios aportando servicios de gestión energética, seguridad, confort, comunicación y accesibilidad.

"La condición necesaria y suficiente que hace que la vivienda pueda considerarse como domótica es que, además de la inclusión de las TIC, disponga de sistemas integrados y que sean interactivos".[1]

La domótica incluye, pero no se limita a la automatización y control de iluminación, ventilación, calefacción, humedad, dispositivos multimedia, videoporteros, cerraduras y alarmas. Implementaciones más avanzadas pueden agregar inteligencia artificial que reconozca instrucciones dictadas en lenguaje natural, o representadas mediante lenguaje de señas para el control del usuario.

1.1.1. Servicios de la domótica

- Gestión energética: se refiere a los servicios de control, programación y ahorro de energía que se pueden implementar en una vivienda. El ahorro energético se centra en el control de la temperatura de los ambientes y de la desconexión de equipos cuando no se utilizan. Pueden instalarse

[1] JUNESTRAND, Stefan. *Domótica y hogar digital*. p. 4.

sistemas de control de calefacción, aire acondicionado, persianas y ventilación artificial por poner un ejemplo.

- Confort: estos servicios tratan de darle al usuario mayor comodidad en su hogar, automatizando sistemas y permitiéndole un control intuitivo y fácil de utilizar. En estos servicios se puede mencionar el control de iluminación, dispositivos multimedia, cerraduras electrónicas y el control del usuario por medio de aplicaciones móviles o por comandos de voz.

- Seguridad: entre estos servicios aparecen aquellos priorizan la protección del usuario y de su patrimonio. Pueden implementarse alarmas y detectores de intrusión, de incendios, de fugas de gas, acceso a cámaras de video y cerraduras electrónicas.

- Comunicación: se refiere a los servicios que tienen por objeto el envío de datos entre usuarios o entre usuarios y sistemas internos. Entre estos se pueden hallar sistemas de teleasistencia, notificación de alarmas a dispositivos móviles, intercomunicadores y control de sistemas desde internet.

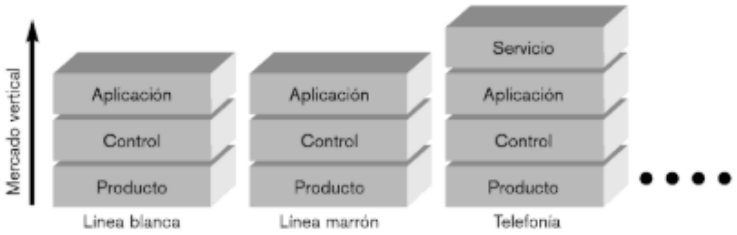
- Accesibilidad: este término se refiere a los servicios que buscan asistir a las personas con discapacidad. Entre estos se ubica la vigilancia remota de lugares inaccesibles, elevadores, servicios de mensajes de emergencia, control centralizado con aplicaciones móviles y servicios de voz para indicar el estado de los sistemas.

1.1.2. Evolución de la tecnología en el hogar

Los electrodomésticos fueron el primer paso para la inclusión de tecnología en el hogar y a ellos se les fueron agregando cada vez más características, como controles a distancia, mayor número de sensores y alarmas, y en casos más recientes, control desde aplicaciones móviles propias y conexión por tecnología inalámbrica. En su mayoría, estas características se fueron añadiendo para satisfacer las necesidades del usuario o brindarle algún grado mayor de comodidad, aunque los algunos fabricantes suelen añadir prestaciones adicionales por presentar alguna innovación vanguardista ante sus competidores que pueden resultar en la creación de una necesidad en el usuario o en que los usuarios no la aprendan a utilizar.

"Esta situación supuso el desarrollo de un mercado puramente vertical, donde los equipos domésticos que se desarrollaban eran totalmente independientes, es decir, que funcionaban de forma autónoma, sin necesidad de comunicarse con otros dispositivos en el hogar".[2]

Figura 1. **Mercado vertical**



Fuente: JUNESTRAND, Stefan, PASSARET, Xavier & VÁZQUEZ, Daniel. *Domótica y hogar digital*. p. 8.

[2] JUNESTRAND, Stefan. *Domótica y hogar digital*. p. 8.

Pero eso implica una dificultad para la prestación de servicios de automatización e imposibilita el control centralizado por parte del usuario.

La domótica en el mercado inició con un control de la alimentación de los dispositivos así, a pesar de que no se pudieran comunicar entre ellos, podían controlarse, aunque de forma limitada, centralizadamente. Por ello se podían implementar servicios limitados al apagado y encendido de aparatos y algunas notificaciones.

Algunos fabricantes, comenzaron a desarrollar una gama de productos que pudieran comunicarse entre ellos con protocolos propietarios y dar control centralizado al usuario. Aunque estas opciones comenzaron a formar parte de un mercado horizontal, se limitaban a ser interoperables únicamente con dispositivos de la misma marca.

Es la adopción de protocolos estandarizados la que expandió las posibilidades de servicios que se pueden prestar en un proyecto de domótica y permitió el desarrollo de un mercado horizontal, donde varios dispositivos se pueden integrar en un sistema, pueden comunicarse entre sí y ser controlados desde dispositivos centrales sin importar el fabricante.

Figura 2. **Mercado horizontal**

Fuente: JUNESTRAND, Stefan, PASSARET, Xavier y VÁZQUEZ, Daniel. *Domótica y hogar digital.* p. 10.

En la actualidad, se han implementado soluciones que además de integrar distintos equipos de fabricantes diversos, añaden inteligencia artificial. Esta se puede utilizar para que el sistema aprenda patrones del usuario, reacciones ante señales visuales, reconozca rostros, identifique intrusiones, acepte instrucciones del usuario dictadas en lenguaje natural e incluso mantenga conversaciones con el usuario mediante comandos de voz.

1.1.3. Implementación de un proyecto de domótica

A continuación, se describen las consideraciones por tomar en cuenta para la implementación de un proyecto de domótica, además, las arquitecturas, topologías, elementos y modelos de comunicación que se pueden utilizar.

1.1.3.1. Consideraciones

Existen algunas consideraciones que se deben de tomar en cuenta cuando se implementa un sistema de domótica:

- Un proyecto de domótica debe permitir el control y monitoreo de todos los sistemas por parte del usuario u operador que presta el servicio. Esto significa que a pesar de la automatización e inteligencia que se aplique a un sistema, siempre debe existir un control redundante delegado a una persona que puede ser el usuario o el operador encargado del servicio de domótica. Esto puede servir para el diagnóstico de fallas o desactivación de sistemas en caso de emergencia.

- Al utilizarse controles remotos, debe implementar algún tipo de autenticación para garantizar que solamente los usuarios autorizados puedan tener control de los sistemas. Es de suma importancia que ningún

usuario sin autorización pueda tomar control de alguno de los sistemas de forma local o remota, parcial o total. Esto, para asegurar la privacidad y seguridad de los habitantes de la vivienda.

- Es también recomendable, implementar sistemas y equipos que sigan estándares para permitir la interoperabilidad entre distintos fabricantes. Esta característica le da al sistema la flexibilidad de incluir los equipos que el usuario desee o necesite, adaptándose a diferentes opciones de presupuesto y calidad. Además, permite integrar nuevos equipos y reemplazarlos cuando sea necesario sin tener que modificar todo el sistema.

- Los sistemas deben permitir el mantenimiento y actualización a futuro. Es necesario que en las instalaciones existan interfaces o cajas de conexiones que permitan el mantenimiento preventivo o correctivo de los sistemas por parte de un técnico. Se puede agregar que mientras la tecnología avanza y el mercado de dispositivos para el hogar se expande, el usuario puede querer agregar nuevas características al sistema en su vivienda y es conveniente que los sistemas puedan se actualizados sin modificar por completo la instalación.

- El sistema debe buscar mejorar la calidad de vida de los habitantes o usuarios de la edificación. El objetivo de la domótica es la automatización de sistemas en un espacio habitable para proporcionar servicios al usuario. Sin embargo, estos servicios deben buscar ser cómodos para el habitante en lugar de representar una carga o una dificultad, por lo mismo, el control de los sistemas debe ser intuitivo para el usuario y el proyecto debe otorgarle la seguridad de que todo está bajo control.

1.1.3.2. Arquitectura de los sistemas

- Centralizada: este tipo de arquitectura se basa en un equipo que tiene acceso a la información de todos los elementos de los sistemas, por lo que la lectura y distribución de instrucciones se realiza de forma centralizada.

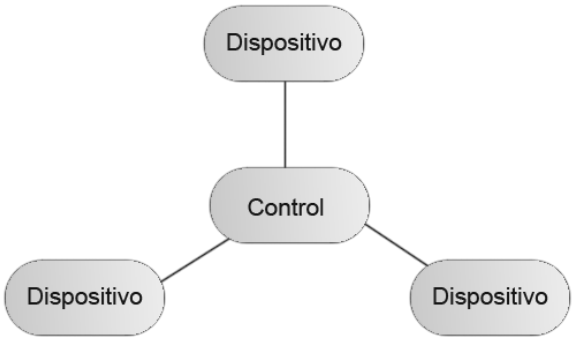
- Distribuida: en este tipo de arquitectura, la inteligencia del sistema se encuentra distribuida en los módulos de sensores y actuadores pudiendo actuar de forma independiente con respecto a otros sistemas.

- Mixta: esta arquitectura combina elementos de las arquitecturas centralizada y distribuida. En esta, el sistema se encuentra distribuido en módulos de sensores y actuadores que pueden procesar la información y distribuirla hacia un elemento de control central.

1.1.3.3. Topología de los sistemas

- Estrella: esta topología está orientada hacia sistemas con arquitectura centralizada, donde todos los dispositivos se encuentran conectados directamente hacia un elemento de control. Los sensores y actuadores no se encuentran conectados entre sí, por lo que el controlador central es el encargado de distribuir la información.

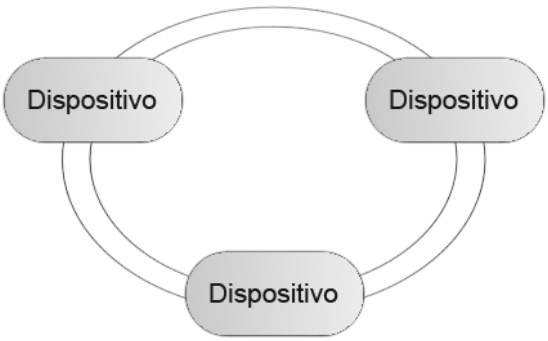
Figura 3. **Topología de estrella**



Fuente: elaboración propia.

- Anillo: cada uno de los dispositivos tiene dos enlaces hacia dispositivos vecinos formando un circuito cerrado. Además, cada uno es capaz de reenviar la información que no le corresponde para su operación.

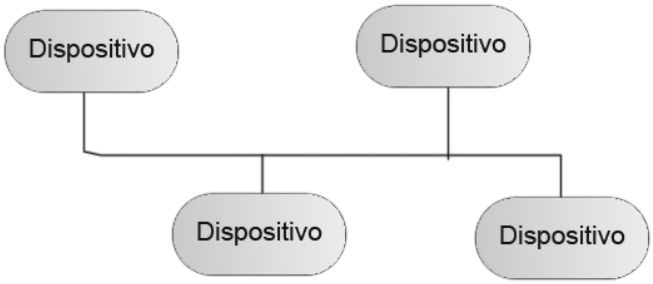
Figura 4. **Topología de anillo**



Fuente: elaboración propia.

- Bus: en esta topología, todos los elementos están conectados a la vez en un solo enlace por medio de derivadores. En esta topología todos los datos pasan a través del mismo enlace y cada dispositivo debe tomar la información que necesite para su operación.

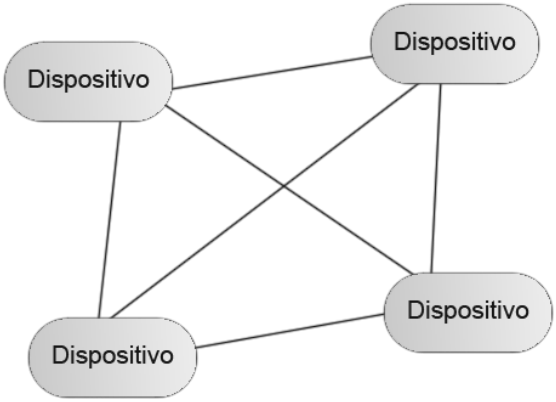
Figura 5. **Topología de bus**



Fuente: elaboración propia.

- Malla: en esta topología los sensores y actuadores se encuentran interconectados por varios enlaces con múltiples dispositivos vecinos a la vez. Esta topología permite tener una mayor disponibilidad y eliminar puntos únicos de falla.

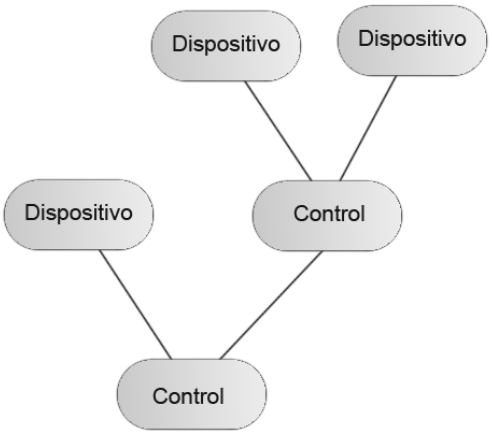
Figura 6. **Topología de malla**



Fuente: elaboración propia.

- Árbol: esta topología es una variante de la topología de estrella, pero a diferencia de esta, la topología de árbol está orientada hacia una arquitectura mixta. En este diseño, los sensores y actuadores se pueden conectar a controladores secundarios que, a su vez, se conectan hacia un controlador central. Esta topología permite distribuir el tráfico de los datos.

Figura 7. **Topología de árbol**



Fuente: elaboración propia.

1.1.3.4. Tipos de enlaces

- Cableado: aquí los elementos del sistema están conectados por cables, generalmente de cobre y pueden ser como se muestra a continuación.

    - Línea eléctrica**:** con modulación se transporta la señal de datos en la misma línea que transporta la tensión eléctrica.

    - Par trenzado**:** es un cable utilizado para datos en el que se pueden alcanzar velocidades más altas.

- Inalámbrico: en este tipo de enlace desaparece el uso de cables para interconectar los dispositivos del sistema. Es necesario apegarse a las reglamentaciones locales sobre la utilización de señales de radiofrecuencia. En este tipo se incluyen lo expuesto en las líneas siguientes.

- RF: se refiere a las comunicaciones por radiofrecuencia no estandarizadas. Puede utilizar cualquier tipo de modulación y enviar los datos de forma digital o analógica.

- Bluetooth: es un estándar de radiofrecuencia de bajo consumo que utiliza la banda ISM de 2.4 GHz, posee un algoritmo de espectro ensanchado por salto de frecuencia (FHSS) que le permite minimizar las interferencias con otros dispositivos y aumentar la seguridad en el enlace, reduciendo el riesgo de intrusión.

- Wifi: es otra tecnología basada en radiofrecuencia estandarizada. Opera en la banda de 2.4 GHz o en la de 5 GHz. Su especificación se encuentra bajo el estándar IEEE 802.11. Con esta tecnología le es posible a un dispositivo tener acceso a internet.

- Óptico: en este tipo de enlace se engloban las tecnologías que utilizan una fuente de luz, visible o invisible para la transmisión de datos.

  - Infrarrojo: utiliza emisores y receptores que utilizan luz fuera del espectro visible, de longitudes de onda mayores que 800 nm. La señal no puede atravesar objetos opacos, por lo que siempre debe haber visibilidad entre emisor y receptor.

  - Fibra óptica: para este enlace se utilizan hebras delgadas de vidrio o de polímeros transparentes que conducen señales de luz, generalmente infrarroja. Estos hilos funcionan por la reflexión interna de las señales de luz utilizando dos materiales con distintos índices de refracción, haciendo aplicación de la ley de Snell.

1.1.3.5. Elementos

Un proyecto de domótica tiene los siguientes elementos:

- Dispositivos del sistema: son todos los dispositivos que se encargan de hacer que funcione el sistema de domótica, son los que manejan las señales provenientes de los dispositivos de entrada dirigidas hacia los dispositivos de salida. Entre estos se encuentran los módulos de control, módulos de comunicación, acopladores de red, fuentes de poder y multiplexores de audio y video.

- Accesorios del sistema: son los elementos de la instalación que deben agregarse para el funcionamiento e interconexión de los equipos. Entre este grupo se encuentran los paneles de conexiones, cableado estructurado, terminales y fichas de conexión.

- Dispositivos de entrada: son todos los dispositivos que introducen información al sistema. Entre estos se pueden identificar dos grupos:

    - Sensores: son los elementos encargados de la medición de los parámetros que controla el sistema. Estos dispositivos recolectan la información y pueden enviarla hacia un dispositivo de control o enviarla directamente a un dispositivo de salida.

    - Teclados: son aquellos elementos que recogen información que el usuario desea ingresar al sistema, generalmente por medio de pulsadores, aunque también se pueden implementar pantallas táctiles.

- Dispositivos de salida: son todos los dispositivos que exteriorizan la información del sistema, ya sea para convertirla en una acción o en una señal para el usuario. Se pueden agrupar en dos conjuntos:

    - Actuadores: son los elementos que utiliza el sistema de control para modificar el estado de otros dispositivos o producir una acción mecánica. Pueden clasificarse dentro de este grupo las alarmas de incendios, rociadores de agua, válvulas.

    - Monitores: estos son los elementos que le permiten al usuario obtener información del sistema, ya sea de forma visual o auditiva. Se puede mencionar dentro de este grupo las pantallas, indicadores lumínicos, alarmas, y dispositivos de reproducción de indicaciones por voz.

- Dispositivos inteligentes: son aquellos que tienen sistemas completos que tienen sensores y actuadores internos integrados. Tienen un mayor procesamiento de información, además manejan protocolos internos y es posible controlarlos por comandos complejos. Entre estos están los teléfonos móviles, computadoras, reproductores de música y dispositivos de inteligencia artificial.

1.1.3.6. Modelos de comunicación

- Servidor cliente: es un modelo de comunicación centralizado donde existe un dispositivo que presta un servicio a la red y puede recibir peticiones de varios dispositivos, denominados clientes. El servidor es el encargado de completar las solicitudes de los clientes. En este modelo, el servidor se mantiene pasivo y la comunicación siempre es iniciada por los clientes.

- Maestro esclavo: es un modelo de comunicación centralizado donde existe un dispositivo, conocido como maestro, que controla y se comunica con varios dispositivos, conocidos como esclavos. Los esclavos siempre se encuentran a la espera de instrucciones y la comunicación siempre es iniciada por el maestro, quien elige con qué esclavo desea comunicarse.

- Red entre pares: es un modelo descentralizado de comunicaciones, conocido como P2P, por su nombre en inglés, en el que cada dispositivo de la red tiene las mismas capacidades, es decir, que puede actuar como servidor y cliente a la vez. De esta forma cualquiera de ellos puede iniciar la comunicación con otro dispositivo.

1.2. Redes

Desde los inicios de la humanidad, siempre ha existido una necesidad por comunicar mensajes hacia otros seres de la especie sin importar las barreras de tiempo o de espacio. Desde las pinturas rupestres y la invención de la escritura hasta la invención de la radio y la televisión, la humanidad, como inteligencia colectiva, ha demostrado buscar y crear nuevos medios de comunicación, cada vez más eficaces. Uno de los últimos avances en la comunicación fue la creación de redes.

Una red es una agrupación de dispositivos que pueden comunicarse entre ellos siguiendo un conjunto de reglas y procedimientos establecidos. Puede ser tan pequeña como dos computadoras o tan grande como millones de ellas.

Las redes revolucionaron la forma en que los seres humanos se pueden comunicar. “La creación y la interconexión de redes de datos sólidas tuvieron un

efecto profundo en la comunicación y se convirtieron en la nueva plataforma en la que se producen las comunicaciones modernas".[3]

1.2.1. Componentes de una red

La infraestructura de la red tiene tres categorías de componentes: dispositivos, medios y servicios.

1.2.1.1. Dispositivos

Los dispositivos son elementos de hardware que realizan el procesamiento de la información que viaja a través del enlace. Se pueden clasificar en dispositivos finales e intermedios.

1.2.1.1.1. Dispositivos finales

"Los dispositivos de red con los que las personas están más familiarizadas se denominan "dispositivos finales" o "*hosts*". Estos dispositivos forman la interfaz entre los usuarios y la red de comunicación subyacente".[4]

Entre estos se pueden encontrar computadoras, teléfonos VoIP, *smartphones*, cámaras de seguridad, *tablets*, entre otros.

Los dispositivos finales son generalmente el origen o el destino de un mensaje transmitido a través de la red. Para su identificación, cada dispositivo debe tener una dirección que envía junto con el mensaje, hacia la dirección del destino, para que el destinatario reconozca el origen de la comunicación.

[3] Cisco Networking Academy. *Principios básicos de enrutamiento y switching.* p. 16.
[4] Ibíd. p. 27.

1.2.1.1.2. Dispositivos intermedios

Son los dispositivos encargados de interconectar a los dispositivos finales. “Estos dispositivos proporcionan conectividad y operan detrás de escena para asegurar que los datos fluyan a través de la red. Conectan los hosts individuales a la red y pueden conectar varias redes individuales para formar una *internetwork*”.[5]

Entre estos dispositivos se pueden encontrar los que dan acceso a la red, como *switches* y puntos de acceso inalámbricos; los conectan diferentes redes, como *routers*; y los dispositivos de seguridad, como *firewalls*.

Estos dispositivos tienen a su cargo la administración de los datos que transitan la red. Entre sus funciones están, regenerar y retransmitir señales de datos, conservar información acerca de las rutas existentes, identificar rutas alternativas en caso de falla de una ruta, notificar a otros dispositivos sobre fallos en la comunicación y permitir o denegar el flujo de datos según la configuración de seguridad.

1.2.1.1.2. Medios

Los medios de la red son los elementos físicos que proporcionan un canal por donde se transporten los datos. Pueden ser hilos metálicos dentro de cables, fibra óptica u ondas de radio en enlaces inalámbricos.

Los datos deben ser codificados acorde con el medio. En los hilos metálicos, los datos se transportan como señales eléctricas; en la fibra óptica, los datos

[5] Cisco Networking Academy. *Principios básicos de enrutamiento y switching*. p. 28.

viajan como pulsos de luz visible o infrarroja; y en el enlace inalámbrico los datos viajan modulando ondas electromagnéticas, generalmente en frecuencia.

Es necesario añadir, que cada medio tiene ventajas y desventajas que lo hacen elegible para cierta conexión. Así que, antes de elegir un medio para una red es necesario considerar la distancia entre los dispositivos que se conectan, el entorno de la instalación, la velocidad necesaria de transmisión y el costo del medio y la instalación.

1.2.1.3. Servicios

Los servicios dentro de una red son los componentes de software que utilizan los dispositivos de la red para una tarea específica.

Hay todo tipo de servicios que realizan todo tipo de tareas; como hallar rutas en una red, otorgar direcciones a los dispositivos que se conectan, transferir archivos entre dispositivos, alojar páginas web, transmitir la hora y fecha, entre otros.

Los servicios están basados en protocolos, comúnmente, estandarizados para permitir que la mayoría de los dispositivos en la red puedan tener acceso a ellos, sin importar su fabricante.

1.3. Internet de las cosas

A continuación, se describe la definición de esta tecnología, su origen, sus requisitos, el modelo de referencia utilizado, los protocolos comunes y la regulación existente en Guatemala.

1.3.1. Definición

En la actualidad, no existe una única definición aceptada por la comunidad global sobre este término. Lo que es común en las definiciones otorgadas al término es que se trata de una tecnología que utiliza las redes para la transmisión de información por parte de dispositivos con cierto grado de autonomía.

Madakam, Ramaswamy y Tripathi lo definen como “una red abierta de objetos inteligentes que tienen la capacidad de auto organizarse, compartir información y recursos, reaccionar y actuar de acorde a los cambios en el ambiente”.[6]

En el Internet de las Cosas no participan únicamente dispositivos inteligentes como teléfonos móviles, computadoras y *tablets*, sino, también sensores, actuadores, controladores y dispositivos multimedia que tienen capacidad de conectarse a la red.

“Con el internet de las cosas se creó un nuevo paradigma de comunicación: máquina-máquina, en el que las máquinas o dispositivos conectados a internet pueden comunicarse entre sí, sin la interacción de los seres humanos. Esto cambia radicalmente el concepto inicial de internet y abre nuevas posibilidades en lo que puede hacerse a través de esta red”.[7]

El Internet de las Cosas permite que dispositivos de uso cotidiano, como bicicletas, refrigeradores, carreteras y automóviles, mediante sensores integrados, puedan recolectar información y enviarla hacia los usuarios y otros

[6] MADAKAM, Somayya. *Internet of Things (IoT): A Literature Review.* http://dx.doi.org/10.4236/jcc.2015.35021. Consulta: febrero de 2019.
[7] SOLIS, Diego. *La privacidad de la información generada por dispositivos de domótica en el Internet de las Cosas.* p. 11.

dispositivos. Esto permitirá que las personas puedan tener una mayor interacción con su entorno.

"Todos los servicios basados en el internet de las cosas proporcionarán mayor automatización a las tareas que involucren objetos y personas, con el objetivo de construir un mundo más inteligente, no solo en la industria sino también en el hogar y en el trabajo".[8]

1.3.2. Origen

El Internet de las Cosas es una tecnología aun en desarrollo, sin embargo, la tecnología aparece por primera vez en la década de 1980 y el término surge en 1999.

Se puede decir que la primera máquina en implementar esta tecnología fue una máquina expendedora de gaseosas en la Universidad Carnegie Melon, cuando unos programadores escribieron un programa en un servidor que tomara lectura de cuándo se había llenado por última vez la máquina. De esa manera, ellos podían conectarse a través de la red con la máquina, leer su estado y determinar si había o no una gaseosa que ellos pudieran tomar y así evitar el viaje en vano hacia la máquina expendedora.

No obstante, el término apareció por primera vez, cuando científicos del MIT, en Estados Unidos, propusieron una identificación electrónica de cualquier dispositivo que dotara a todos los dispositivos en el mundo de un código que no cambiara según su ubicación.

[8] SOLIS, Diego. *La privacidad de la información generada por dispositivos de domótica en el Internet de las Cosas*, p. 12.

El concepto de IoT adquirió popularidad en el 2003, gracias a Auto-ID Centre, con publicaciones relacionadas al análisis del mercado. Sin embargo, “no fue sino hasta 2010, cuando el término tomó importancia y varios libros y artículos científicos fueron publicados tratando el tema”.[9]

“En la actualidad, existen más de 100 millones de máquinas expendedoras, vehículos, detectores de humo y otros dispositivos que ya comparten información automáticamente, una cifra que los analistas de mercado de Berg Insight esperan que suba a 360 millones para el año 2016”.[10]

1.3.3. Modelo de referencia

Un modelo de referencia es un marco que permite la organización funcional de las capacidades de un sistema. Un sistema de Internet de las Cosas, debe tener una variedad de elementos para que funcione. El problema de esta tecnología es que aún no se encuentra estandarizada, por lo que no existe un común acuerdo sobre un modelo de IoT. Sin embargo, la Unión Internacional de Telecomunicaciones (ITU) propone un modelo de IoT que consiste en cuatro capas:

- Capa de aplicación**:** contiene las aplicaciones específicas.

- Capa de apoyo a servicios y aplicaciones: se refiere a capacidades comunes que pueden utilizar distintas aplicaciones.

[9] SOLIS, Diego. *La privacidad de la información generada por dispositivos de domótica en el Internet de las Cosas*. p.10.
[10] Cisco Networking Academy. *Principios básicos de enrutamiento y switching.* p. 520.

- Capa de red: ofrece funciones de conectividad a la red, autenticación y transporte.

- Capa de dispositivo: es la capacidad que permite que los dispositivos que se puedan conectar a la red.

1.3.4. Requisitos de alto nivel

Existen requisitos importantes que debe satisfacer un sistema de IoT:

- Conectividad basada en la identificación: esto significa que se debe procesar de forma unificada los identificadores heterogéneos de los dispositivos.

- Compatibilidad: es importante que diversos sistemas puedan interactuar entre ellos.

- Configuración automática de servicios: es necesario que los servicios se puedan configurar a partir de datos obtenidos de los dispositivos.

- Capacidades basadas en ubicación: un sistema de IoT debe poder brindar capacidades que trabajen según la ubicación de un dispositivo.

- Seguridad y privacidad: es necesario integrar técnicas de seguridad y autenticación en los dispositivos que envían datos a través de la red.

- Autoconfiguración: IoT debe permitir la integración sobre la marcha de dispositivos al sistema.

- Capacidad de administración: un sistema de IoT debe permitir la administración por parte de un usuario.

1.3.5. Protocolos utilizados

En la capa de aplicación del modelo TCP/IP existen tres protocolos que se utilizan principalmente para el desarrollo de Internet de las Cosas:

- CoAP (*Constrained Application Protocol*): es un protocolo web especializado que se utiliza en redes limitadas, ya sea por consumo energético, baja capacidad de procesamiento de datos o altas pérdidas. El protocolo está diseñado para comunicación máquina-máquina en aplicaciones de gestión energética y automatización de edificios.

- MQTT (*Message Queuing Telemetry Transport*): es un protocolo diseñado para ser ligero basado en transporte de mensajes a través de un modelo de publicación/suscripción. Es útil para conexiones en ubicaciones remotas donde se requiere una baja transferencia de datos, como en conexiones vía satélite y por dial-up.

- HTTP (*Hypertext Transfer Protocol*): es un protocolo de aplicación para sistemas de información distribuidos y colaborativos. Es un protocolo genérico y sin estado que puede ser utilizado para la gestión de objetos a través de sus métodos de petición, cabeceras y códigos de error. Una característica de HTTP es la negociación de la representación de datos, permitiendo a los sistemas ser construidos independientemente de los datos transferidos.

1.4. IoT en la domótica

A través de la red las variables de un sistema de domótica pueden ser monitoreadas en todo momento por el usuario o por una entidad administradora del mismo. Así mismo, con la conectividad a internet, se elimina la limitación geográfica para el administrador del sistema, pues puede obtener información y control en cualquier lugar.

La conectividad en red no solamente favorece el control del usuario y la comunicación humano-máquina, sino que permite la comunicación máquina-máquina fuera del límite doméstico, con esto es posible conectarse a servidores que prestan servicios especializados como inteligencia artificial, gestión energética, teleasistencia, bases de datos multimedia, entre otros, mejorando enormemente la experiencia de los usuarios del sistema de domótica.

Cada servicio de la domótica puede ser mejorado con equipos que se pueden conectar en red, es decir, con la tecnología de Internet de las Cosas. Como ejemplos, en cuanto al confort, el usuario puede conectar sus dispositivos a internet y reproducir contenido multimedia alojado en servidores externos y utilizar asistentes personales basados en inteligencia artificial para gestionar su itinerario y listas de pendientes.

Relacionado con la gestión energética, los datos recolectados por los equipos y sensores pueden ser enviados en tiempo real a un servicio en internet, otorgándole control de la energía, para que se encargue de mejorar la eficiencia energética y prevenir fallas eléctricas.

En accesibilidad, un usuario con limitación motriz puede hacer uso de asistentes basados en inteligencia artificial que procesan instrucciones en lenguaje natural y que le permitan administrar los equipos en su hogar.

En lo que refiere a seguridad, el usuario puede tener información de su hogar en todo momento, aun cuando se encuentra fuera de su domicilio y puede activar o desactivar características de sus electrodomésticos para evitar accidentes infantiles.

Por último, referente a la comunicación, los usuarios pueden utilizar sus los televisores y sistemas de audio existentes como interfaces de telepresencia sin necesidad de equipos especializados adicionales.

El Internet de las Cosas, hace posible que todos los equipos puedan compartir información no solo con los usuarios, sino, entre ellos y abre las puertas a sistemas interrelacionados más eficientes, más seguros, con más funciones y más cómodos para aquellos que ocupan los espacios habitables.

1.5. Regulación de redes de IoT en Guatemala

En Guatemala, la Superintendencia de Telecomunicaciones, regula las redes inalámbricas de área extensa, exteriores, mayores a 1000 m^2, que usan baja o muy baja potencia utilizadas para dispositivos de Internet de las Cosas, este tipo de red es llamado RIABAP.

Según la Superintendencia de Telecomunicaciones,[11] estas redes deben operar en las bandas definidas en el anexo 1 con un ancho de banda máximo de

[11] Superintendencia de Telecomunicaciones de Guatemala. *Resolución SIT-DSI-349-2019*. https://sit.gob.gt/2019/09/11/resolucion-sit-dsi_349-2019. Consulta: octubre de 2019.

500 kHz y tener una potencia isotrópica radiada equivalente máxima de 26,99 dBm desde el dispositivo hacia la radio base y de 36,00 dBm desde la radio base hacia los dispositivos.

Estas redes no tienen permitida la transmisión de voz y video por ser limitadas en cuanto a la cantidad de información que pueden enviar. Al ser orientadas a sensores y actuadores, principalmente, no deben superar los 40 bytes de información ni los 400 ms en la transmisión de cada mensaje. Además, los dispositivos en una RIABAP no cuentan con protección contra interferencias de cualquier otra estación ni deben producir interferencias en otras estaciones radioeléctricas.

Las RIABAP deben ser inscritas ante el Registro de Telecomunicaciones de la SIT y ser actualizadas cada cierre de año fiscal. Los cargos administrativos se encuentran en el anexo 2.

De acuerdo con la Superintendencia de Telecomunicaciones,[12] las redes desplegadas en ambientes interiores o semi exteriores que no se pueden clasificar como RIABAP pero trabajan en las mismas bandas electromagnéticas solamente deben respetar los límites de potencia establecidos para no causar interferencias en otras radios bases, pero no están limitados en cuanto a su transmisión ni deben ser inscritas ante la SIT.

[12] Superintendencia de Telecomunicaciones de Guatemala. *Resolución SIT-DSI-350-2019.* https://sit.gob.gt/2019/09/11/resolucion-sit-dsi-350-2019. Consulta: octubre de 2019.

2. PROTOCOLO DE TRANSFERENCIA DE HIPERTEXTO

2.1. Protocolos de red

Antes de que dos dispositivos puedan comunicarse, es necesario que ambos estén en común acuerdo sobre las reglas que enmarcarán la transmisión del mensaje. Estas reglas, o protocolos, deben respetarse para que el mensaje sea enviado y recibido correctamente.

"Los protocolos que se utilizan en las comunicaciones de red comparten muchas de las características fundamentales de los protocolos que se utilizan para regir las conversaciones humanas correctas".[13]

Los protocolos informáticos y de red, generalmente, deben definir la codificación, formato y encapsulación, tamaño, temporización y opciones de entrega de un mensaje.

- Codificación: "es el proceso mediante el cual la información se convierte en otra forma aceptable para la transmisión. La decodificación invierte este proceso para interpretar la información".[14]

- Formato y encapsulación: el formato se refiere a la sintaxis que manejará el mensaje, es decir, la estructura de la información. La encapsulación, por otra parte, hace referencia a los datos adicionales que deben agregarse a

[13] Cisco Networking Academy. *Principios básicos de enrutamiento y switching.* p. 119.
[14] Ibíd. p. 120.

los datos, pero que no son parte del mensaje. Ambos deben estar definidos para que dos computadoras puedan entender la información del mensaje.

- Tamaño del mensaje: es importante que el protocolo que manejan los dispositivos, determine el tamaño del mensaje de alguna forma. De esta manera, el emisor podrá conocer el límite de información que el emisor puede recibir o el receptor podrá saber en dónde termina un mensaje enviado.

- Temporización: la velocidad de transmisión debe estar definida para que un dispositivo no envíe información a una razón que el dispositivo receptor no pueda procesar. Además, deben establecerse tiempos de espera para que un dispositivo no interrumpa de forma indefinida su funcionamiento por estar a la expectativa de una respuesta cuando una conexión se ha interrumpido.

- Opciones de entrega: puede ocurrir, que un emisor desee enviar un mensaje hacia uno o varios receptores. Además, puede que el emisor necesite una confirmación de que el mensaje ha sido recibido correctamente o una notificación en caso de haber una falla en la recepción.

2.2. Interacción de protocolos

Cuando se envía información a través de una red, normalmente, entran en juego varios protocolos que trabajan en conjunto para asegurar que la información pueda llegar a su destino.

Los protocolos deben estar adaptados para poder funcionar uno sobre otro, como en una pila, cada uno con su tarea. La jerarquía que utilizan los protocolos se define con un modelo.

Cada protocolo se puede clasificar en una capa de un modelo conocido como TCP/IP, dependiendo de su rol en la transmisión del mensaje.

2.2.1. Modelo TCP/IP

El modelo TCP/IP segmenta la red en cuatro capas. Mantiene la clasificación de los protocolos de forma flexible, más que su competidor, el modelo OSI. A ello se debe que el modelo TCP/IP sea, en la actualidad, el más usado para clasificar los protocolos. Sus capas son:

- Capa 1: acceso a la red. Engloba los protocolos que permiten el control de flujo, direccionamiento físico y transmisión de datos en forma de bits en el medio.

- Capa 2: internet. Se conforma por los protocolos que proveen direccionamiento lógico y encuentran rutas a través de la red para el envío de paquetes.

- Capa 3: transporte. Se encarga de transportar los datos independientemente de la conexión física que se utilice.

- Capa 4: aplicación. Los protocolos clasificados en esta capa son los que regulan las comunicaciones entre las aplicaciones haciendo uso de las capas inferiores.

2.3. HTTP

El protocolo de transferencia de hipertexto es un estándar de las comunicaciones en internet que se utiliza para transferir información a través de

la *World Wide Web*. Fue creado en la década de los noventa en una colaboración del *World Wide Web Consortium* (W3C) y el *Internet Engineering Task Force* (IEFT).

Existen varias versiones de HTTP. La primera versión, HTTP/0.9, fue un protocolo diseñado para la transferencia de información en bruto a través del internet. A partir de esta se han creado otras versiones, siendo HTTP/1.1 la más utilizada en la *World Wide Web*.

Este protocolo funciona con un modelo de comunicación servidor/cliente, como muchas de las comunicaciones en red, donde un servidor atiende las peticiones de uno o varios clientes que requieren acceso a un recurso, identificado por un URI, y devuelve una respuesta a cada petición con algún código de respuesta y el contenido.

"La comunicación de HTTP usualmente se realiza sobre conexiones TCP/IP. El puerto por defecto es TCP 80, pero otros puertos se pueden usar. Esto no impide que HTTP sea implementado sobre cualquier otro protocolo en el internet, o en otras redes. HTTP solo requiere de un transporte seguro; cualquier protocolo que provea esa garantía puede ser utilizado".[15]

HTTP es un protocolo sin estado, es decir, que no mantiene información de conexiones anteriores. Por tanto, en cada conexión se debe enviar toda la información necesaria sobre la conexión.

[15] FIELDING, R. *Hypertext Transfer Protocol – HTTP/1.1.* https://tools.ietf.org/html/rfc2616. Consulta: marzo de 2019. p. 12.

2.3.1. Hipertexto

Este término se refiere a un texto que contiene enlaces hacia otros textos, que pueden estar alojados en el mismo recurso o pueden estar en un recurso diferente.

El término hipertexto ha sido extendido a hipermedia para referirse a que no solamente se puede utilizar texto que contenga enlaces a textos, sino, se puede incluir videos, imágenes y sonidos.

2.3.2. URI

Sus siglas, en inglés, significan identificador uniforme de recurso. Es la unión de un identificador uniforme de ubicación (URL) y un identificador uniforme de nombres (URN).

“Para HTTP, los Identificadores Uniformes de Recursos son cadenas de texto con formato simple que identifican, a través de nombre, ubicación o cualquier otra característica un recurso”.[16]

Un URI se puede representar de forma absoluta o relativa a otro URI. La diferencia radica en que uno absoluto siempre tiene un símbolo de dos puntos al inicio. Estos pueden ser de cualquier longitud, pues en HTTP no se establece un límite.

[16] FIELDING, R. *Hypertext Transfer Protocol – HTTP/1.1.* https://tools.ietf.org/html/rfc2616. Consulta: marzo de 2019. p. 17.

2.3.3. Mensaje de HTTP

Un mensaje de HTTP siempre está enviado en texto plano y puede ser de petición o de respuesta. Su estructura está formada por una línea inicial, cabeceras y contenido.

La línea de inicio siempre debe terminar en un retorno de carro y un salto de línea. Si es de petición debe incluir el método de petición y el identificador del recurso al que se desea acceso en conjunto con la versión que soporta el cliente. En caso del servidor, debe enviar la versión de HTTP utilizado seguido de un código de respuesta.

2.3.3.1. Métodos de petición

Los métodos de petición son instrucciones que el cliente envía al servidor para que realice alguna acción y devuelva una respuesta. En la versión HTTP/1.1 existen ocho métodos, aunque solamente GET y HEAD son obligatorios para implementar en todo servidor, los demás métodos son opcionales.

El método de petición se envía al inicio de una petición HTTP a un servidor, generalmente, seguido por un URI, y debe ser escrito en mayúsculas. Los métodos se detallan a continuación:

- OPTIONS: se utiliza en una petición de información acerca de las opciones disponibles en la comunicación. Este método le permite al cliente determinar las opciones disponibles sobre un recurso o un servidor sin necesidad de enviar una petición de un recurso.

- GET: es un método para obtener cualquier información, por tanto, debe incluir una URI en la petición. El servidor envía la información solicitada en el cuerpo de la respuesta. La respuesta puede ser una página web accedida a través de un navegador.

- HEAD: este método es similar al método GET, a excepción que el servidor no envía un cuerpo en la respuesta. Este método se utiliza para obtener las cabeceras o metadatos sin la necesidad de transferir un recurso.

- POST: es un método utilizado para pedir que el servidor acepte una entidad como un subordinado de un recurso especificado por un URI. Sin embargo, el resultado de la solicitud es determinado por el servidor.

- PUT: es un método utilizado para pedir que el servidor guarde una entidad en el URI especificado. La diferencia con el método POST es que PUT no solicita que la entidad sea subordinada de un recurso, sino un recurso con un URI específico.

- DELETE: este método le solicita al servidor que borre un recurso identificado con un URI. Este método puede ser sobrescrito por intervención humana por lo que no es garantizado que el recurso se elimine. No obstante, el servidor debe informar el resultado de la petición.

- TRACE: este método es utilizado por el cliente para obtener la información que el servidor está recibiendo como petición. El servidor debe enviar la petición obtenida en el cuerpo de la respuesta.

- CONNECT: este método se utiliza con un proxy para saber si está permitido el acceso a un host con ciertas condiciones.

2.3.3.2. Cabeceras

Las cabeceras son información que se envía en una petición o respuesta de HTTP para dar información acerca de la transacción.

Entre las cabeceras se puede enviar información sobre el tipo de información que acepta el cliente, el lenguaje, el conjunto de caracteres, fecha, credenciales de autorización, métodos de acceso permitidos, el navegador y sistema operativo desde donde se accede, entre otros.

2.3.3.3. Códigos de estado

Cuando un servidor recibe una petición debe enviar una respuesta hacia el cliente. En esta respuesta incluirá el estado de la petición que recibió al inicio, justo después de indicar la versión de HTTP. Estos códigos son de tres dígitos, el primero indica el tipo del código de estado.[17]

- Información (1xx): este tipo de códigos indica una respuesta provisional que consiste solamente del código de estado y cabeceras opcionales.

- Éxito (2xx): esta clase de código indica que la petición del cliente fue recibida, entendida y aceptada con éxito.

- Redirección (3xx): indica que el recurso solicitado no se encuentra en el URI especificado.

[17] FIELDING, R. *Hypertext Transfer Protocol – HTTP/1.1.* https://tools.ietf.org/html/rfc2616. Consulta: marzo de 2019. p. 17.

- Códigos de error de cliente (4xx): estos códigos son enviados por el servidor cuando el cliente ha realizado una petición con algún error. En el cuerpo de la respuesta puede enviarse una entidad indicando cual ha sido el error.

- Códigos de error de servidor (5xx): estos códigos indican que el servidor ha identificado que produjo un error y no puede satisfacer la petición.

La lista detallada de códigos de estado se puede ver en el anexo 3.

2.3.4. Servicios *RESTful*

"REST es un conjunto de principios de arquitectura, con los que se puede diseñar servicios web que se enfoquen en los recursos del sistema, incluyendo cómo los estados de los recursos son transferidos a través de HTTP por un gran rango de clientes escritos en distintos lenguajes".[18]

Un servicio web que implementa la arquitectura REST (transferencia de estado representativo) se conoce como *RESTful*. Esta arquitectura se basa en cuatro principios de diseño:

- Usa los métodos HTTP explícitamente: significa que utiliza los métodos de acceso como fueron definidos en la especificación de HTTP/1.1. Así:

    - Usa POST para crear un recurso en el servidor
    - Usa GET para obtener un recurso del servidor

[18] RODRÍGUEZ, Alex. *RESTful Web Services.* https://developer.ibm.com/articles/ws-restful/. Consulta: marzo de 2019.

- Usa PUT para cambiar el estado de un recurso o actualizarlo
- Usa DELETE para eliminar un recurso

- Funciona sin estado: se debe enviar toda la información que necesita el servidor dentro de la petición. Es decir, que se debe asumir que el servidor no mantiene información de peticiones anteriores.

- Expone los URI con la estructura de un directorio: los identificadores del recurso deben tener el formato de una dirección de directorio para que sea intuitiva, para el desarrollador, la ubicación de los recursos.

- Transfiere XML, JSON o ambos: estos son modelos de intercambio de objetos que reflejan el estado de un recurso y sus atributos.

2.3.5. Autenticación de HTTP

"HTTP provee algunos mecanismos opcionales de autenticación desafío-respuesta que pueden ser usados por un servidor para comprobar una petición de un cliente o por un cliente para proveer información de autenticación".[19]

La autenticación se puede realizar por un método de autenticación básica o por un método de *digest authentication*.

En el primero el cliente debe enviar sus credenciales dentro de la petición con el encabezado "*Authorization*" seguido de la palabra "*Basic*", un espacio y la pareja usuario: contraseña codificada en el sistema Base64. Si el cliente no envía las credenciales en la petición, el servidor enviar una respuesta con código 401,

[19] FIELDING, R. *Hypertext Transfer Protocol – HTTP/1.1.* https://tools.ietf.org/html/rfc2616. Consulta: marzo de 2019. p. 70.

indicando que se necesita autenticación. Además, deberá incluir el encabezado "*WWW-Authenticate*" seguido de las palabras "*Basic realm*", el signo de igual y una cadena de texto entre comillas que indique el recurso que está protegiendo.

En el segundo método, cuando el cliente realiza una petición, el servidor debe enviar una respuesta indicando con el encabezado "*WWW-Authenticate*" un valor llamado "*nonce*". El cliente debe tomar este valor, concatenarlo a la pareja usuario: contraseña y aplicar una función hash con el algoritmo MD5. Este se envía hacia el servidor con el encabezado "*Authorization*", seguido de la palabra "*Digest*", un espacio y el hash codificado en el sistema Base64.

De esta forma "*digest authentication*" mejora la seguridad al no enviar las credenciales en la petición.

3. PROTOCOLO DE IOT SOBRE HTTP

3.1. Alcance de la especificación

En esta sección se describirá un protocolo de aplicación que toma como base el protocolo HTTP en su versión 1.1 para comunicar dispositivos a través de una red, independientemente si tiene acceso a internet, en una arquitectura centralizada o mixta, con un modelo cliente/servidor, orientado para su aplicación en una topología de estrella o árbol.

Esta descripción definirá el formato de los datos transmitidos, la nomenclatura que utilizarán los dispositivos, el significado de cada uno de los campos en los metadatos, la función utilizada de cada uno de los cuatro métodos de acceso principales de HTTP, los procedimientos para publicar, actualizar, eliminar y obtener datos, los formatos de URI en las peticiones por utilizar y los encabezados obligatorios que contendrá cada petición al servidor.

También se especificará cómo se identificará cada sensor o actuador, la forma de autenticación de los dispositivos y una manera de agrupar las peticiones en una topología de árbol para optimizar el número de transmisiones.

3.2. Descripción general

Este protocolo pertenece a la capa de aplicación del modelo TCP/IP y a la capa de apoyo a servicios y aplicaciones del modelo de la Unión Internacional de Telecomunicaciones para Internet de las Cosas. Este protocolo utiliza como base el protocolo HTTP en su versión 1.1.

Trabaja con un modelo cliente/servidor, donde cada uno de los dispositivos envía peticiones de HTTP hacia un servidor, que almacena el estado de cada uno de los dispositivos, para actualizar los datos que ha obtenido o recuperar los datos que debe exteriorizar.

El servidor utiliza el estándar REST para permitir la interoperabilidad con otras tecnologías existentes en la web. Utiliza el método de acceso POST para añadir un dispositivo, PUT para actualizar los datos, GET para obtener los datos y DELETE para eliminar un dispositivo.

Utiliza los códigos de estado de HTTP para notificar al cliente sobre el resultado de la petición. Y los datos sobre el estado y propiedades de los dispositivos son enviados en las peticiones y respuestas de HTTP en objetos JSON o XML que representan a cada dispositivo con un identificador jerárquico único.

3.3. Compatibilidad del protocolo

El protocolo de transferencia de hipertexto en su versión 1.1 es el protocolo de transmisión de texto más utilizado en la *World Wide Web*. El objetivo de utilizar este protocolo de transmisión de texto como base para la comunicación de dispositivos de IoT es permitir la rápida adaptación de los desarrolladores que conocen el mismo, ya sean profesionales o aficionados.

De esta forma, no es necesario para los desarrolladores invertir una gran cantidad de tiempo para instruirse sobre un protocolo completamente distinto al momento de realizar una aplicación de Internet de las Cosas.

Además, al ser un estándar ampliamente distribuido en el mundo y por su minimalismo, comparado con otros protocolos en la web, un protocolo de IoT basado en HTTP, puede ser implementado en todos los dispositivos que tengan acceso a la red aun con capacidades mínimas.

A pesar de que el protocolo HTTP trasmite la información en texto plano, es completamente compatible, y por tanto actualizable, con su versión segura HTTPS, pues comparte la misma base de especificación, con la diferencia que el último agrega encriptación asimétrica a la comunicación.

Adicional a la ventaja de la rápida implementación de HTTP, el protocolo propuesto para Internet de las Cosas utiliza un servidor que implementa el estándar REST. Este es un estándar que ha ganado una gran cantidad de terreno en internet por permitir la intercomunicación entre aplicaciones heterogéneas en la red. Grandes empresas de productos en internet implementan aplicaciones con estándar REST para permitir la integración por parte de los desarrolladores.

Otra ventaja de compatibilidad que se ha incluido en el protocolo propuesto es el uso de objetos JSON y XML. Estos últimos son estándares para la transmisión de datos entre aplicaciones en la red y son implementados en una gran cantidad de aplicaciones en internet. Las aplicaciones que utilizan el estándar REST hacen uso de estos objetos, principalmente JSON que, por simpleza ante XML, lo ha ido desplazando en los años recientes.

Los objetos JSON y XML pueden ser manejados en muchos lenguajes de programación incluyendo los que se implementan para su ejecución en navegadores *web*.

Por último, todos los navegadores han sido creados para interpretar el protocolo HTTP y enviar peticiones con el método de acceso GET, por lo que el monitoreo de los sistemas de IoT se puede realizar con cualquier navegador desde cualquier dispositivo conectado a la red. Adicional al esto, fuera de la especificación del protocolo propuesto, un desarrollador podría implementar, en el mismo servidor que controla a los dispositivos, una interfaz de usuario que permitiera controlar el sistema.

3.4. Especificación

A continuación, se detalla cada uno de los elementos que conformarán el protocolo de IoT sobre HTTP.

3.4.1. Características

A continuación de define la arquitectura, modelo de comunicación y topología considerados para el diseño del protocolo propuesto.

3.4.1.1. Arquitectura

El protocolo propuesto funciona con un dispositivo central que debe contener toda la información de los dispositivos que conforman el conjunto. A este dispositivo deben llegar todas las actualizaciones del estado de cada uno de los sensores o actuadores, además de todas las instrucciones dictadas por un usuario.

A partir de él serán distribuidas todas las instrucciones hacia los dispositivos. Esto no excluye a que puedan existir en la red, dispositivos que realicen algún procesamiento independiente antes de enviar la información al

dispositivo central. Esta descripción hace referencia a una arquitectura centralizada, si no existen dispositivos que realizan procesamientos independientes, o mixta, si los hay.

Estas arquitecturas se han elegido para el protocolo propuesto por las siguientes razones:

- El uso de un dispositivo central permite que la información pueda ser accesible por cualquier dispositivo de la red sin importar el origen.

- El hecho que toda la información de los dispositivos se encuentre en un solo lugar le permite al usuario monitorear el sistema accediendo desde un solo punto.

- Cada sensor o actuador debe mantener únicamente una vía de comunicación por lo que el procesador no debe realizar ninguna conmutación en la comunicación.

- Un dispositivo central permite la conexión lógica de subconjuntos de dispositivos distribuidos en distintas áreas geográficas.

3.4.1.2. Modelo de comunicación

El protocolo propuesto funciona con un modelo cliente/servidor, donde existe un dispositivo que contiene la información del estado de cada uno de los sensores o actuadores en el conjunto. Este actúa con el rol de servidor y permanece de forma pasiva a la escucha de peticiones, mientras los datos le son solicitados o publicados por parte de los demás dispositivos que actúan con el rol de clientes.

Pueden existir en el conjunto dispositivos que realicen algún procesamiento antes de solicitar o publicar la información hacia el dispositivo central; estos funcionarán como servidor de otros dispositivos subordinados y a la vez como clientes del servidor central.

Este modelo de comunicación se ha elegido por las siguientes razones:

- La implementación de un servidor de HTTP requiere de mayores capacidades por parte del procesador de un dispositivo, en comparación con un cliente de HTTP, por tanto, este modelo utiliza el mínimo de dispositivos con mayores capacidades computacionales.

- Este modelo permite que el dispositivo central pueda ser alojado en un dominio de internet para que pueda ser accesible desde cualquier ubicación geográfica.

- Los navegadores web funcionan como un cliente de HTTP, por lo que este modelo permite el monitoreo del sistema a través de cualquier navegador sin software adicional.

- El uso de un servidor central que permanezca a la escucha de peticiones permite que se puedan agregar al conjunto distintos dispositivos con diferentes capacidades de procesamiento y diferentes velocidades de transmisión sin afectar a los otros dispositivos o la velocidad del sistema en general

3.4.1.3. Topología

Este protocolo está orientado para su implementación en una topología de estrella o árbol según su arquitectura y modelo de comunicación, sin embargo, la topología propuesta no es obligatoria y puede implementarse en la capa de enlace una topología en anillo, malla o bus. No obstante, el protocolo HTTP, a nivel de aplicación funciona únicamente con las topologías propuestas y aunque se implemente otra topología en la capa de enlace, a nivel lógico, se interpretará como una topología de estrella o árbol según sea la arquitectura centralizada o mixta respectivamente.

En una topología de estrella existe un solo dispositivo central al que se conectan todos los demás dispositivos y si los dispositivos aledaños necesitan comunicarse, la información obligatoriamente debe atravesar el dispositivo central. La topología de árbol es una variante de la topología de estrella, donde los dispositivos periféricos tienen a la vez otros dispositivos subordinados.

Se han propuesto estas topologías debido a las siguientes razones:

- El uso de un solo dispositivo central permite que el sistema continúe funcionando a pesar de que los dispositivos periféricos fallen.

- Al realizarse la distribución de datos a través de un dispositivo central es posible realizar mantenimiento en los dispositivos aledaños sin detener el funcionamiento general del sistema.

- En caso de una falla total del sistema, estas topologías presentan un único responsable, por lo que la identificación del problema es más rápida.

3.4.2. Definiciones

A continuación, se detallan los términos que se utilizan en la especificación del protocolo propuesto. Los términos se listarán en orden alfabético en español y se escribirá su traducción al inglés entre paréntesis. La traducción en inglés se utiliza ya que este idioma es el que predomina en la documentación de las tecnologías de la información. Se ha elegido una nomenclatura que hace referencia a la morfología vegetal por la similitud de la estructura externa de una planta con la forma en que se realiza la comunicación con el protocolo propuesto en su topología de árbol.

- Árbol (*tree*): es un conjunto de dispositivos que trabajan juntos en un sistema de domótica y que utilizan un dispositivo central para su comunicación. El grupo debe contener un dispositivo central y dispositivos periféricos. Puede contener, pero no de forma obligatoria, dispositivos que realicen procesamientos intermedios antes de enviar la información al dispositivo central.

- Dispositivo (*device*): es cualquier objeto físico o virtual que funcione como sensor, actuador, indicador visual, monitor, intérprete de instrucciones, almacén de datos, distribuidor de comunicación o procesador de información.

- Estado (*state*): este es el dato que participa directamente en el control del sistema. Este término hace referencia al dato obtenido por un sensor o al que debe exteriorizar un actuador o un indicador visual. El estado es la propiedad del dispositivo que se modifica en las actualizaciones periódicas.

- Llave (*key*): es la clave o contraseña que utilizarán los dispositivos pertenecientes a un grupo para identificarse como miembros de él y tener acceso a la información de los demás dispositivos. Con esta llave, el dispositivo central puede identificar que dispositivos se pueden agregar al grupo o pueden realizar cambios. La misma servirá a los usuarios para poder realizar modificaciones o pedir actualizaciones de estado.

- Metadatos (*metadata*): son las propiedades de un dispositivo. Entre estos se da una descripción del dispositivo, en ellos se encuentra información que permite al usuario, o al sistema de control, identificar la función, el nombre y la ubicación de un dispositivo.

- Nudo (*knot*): es un dispositivo que presta un servicio de HTTP y que puede realizar algún procesamiento intermedio antes de enviar la información al dispositivo central. Este dispositivo se encarga de agrupar la información de sus dispositivos subordinados y enviarla en una sola petición al servidor central, o a otro servidor secundario superior en la jerarquía, y distribuir la información que recibe del servidor central hacia sus dispositivos subordinados.

- Raíz (*root*): es el conjunto de todos los programas de control del grupo, tanto los que aportan automatización al sistema como los que interpretan las instrucciones por parte del usuario. Es la unión de la raíz primaria y secundaria. Ambos programas pueden estar en el mismo dispositivo.

- Raíz primaria (*primary root*): es el programa que se encarga de interpretar las instrucciones del usuario. Se ocupa de obtener las entradas por medio de texto, botones, comandos de voz o señales visuales que ingresa el usuario y traducirlas a modificaciones en el estado de actuadores o

dispositivos de visualización o a instrucciones sobre la modificación de dispositivos. Este programa puede estar en un dispositivo independiente o estar en el dispositivo central. Este programa es el que le otorga el control al usuario sobre el sistema de domótica, por tanto, siempre debe existir.

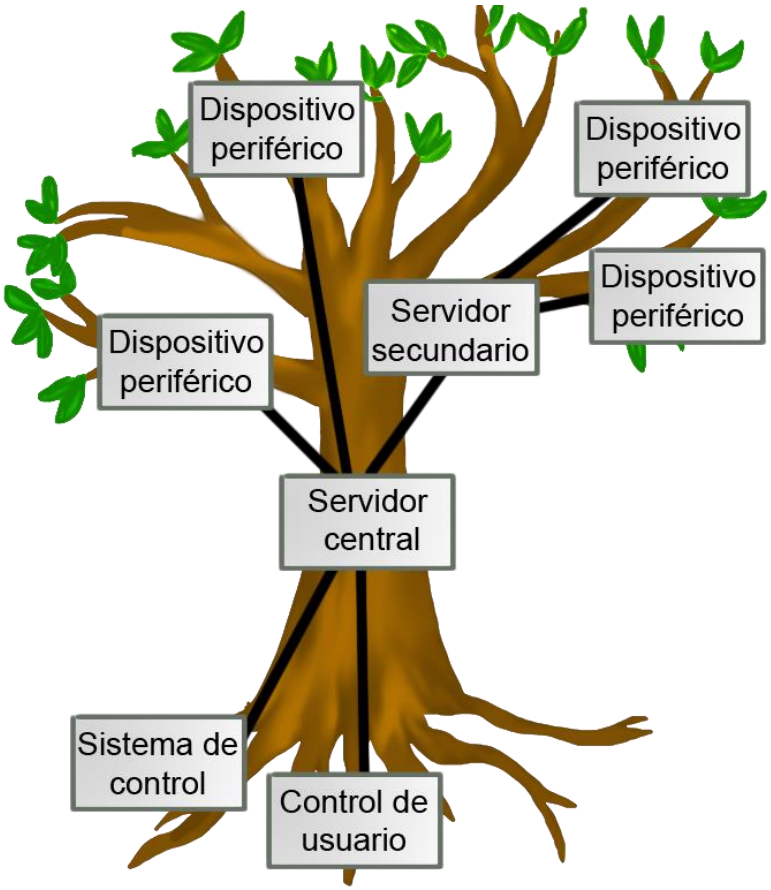
- Raíz secundaria (*secondary root*): es el programa que realiza la automatización o inteligencia del sistema. Se encarga de modificar el estado de los actuadores o dispositivos de visualización de forma automática, según los valores que toman los sensores. Este programa puede estar en un dispositivo independiente o en el dispositivo central. En sistemas de domótica donde no existe automatización, sino, únicamente el control por parte del usuario, este dispositivo no aparece, pero debe existir uno que interprete las instrucciones por parte del usuario.

- Rama (*branch*): es un dispositivo que se encuentra al final de la línea de comunicación. Se encarga de tomar datos de su entorno por medio de sensores y/o de exteriorizar datos enviados por otros dispositivos con actuadores o dispositivos de visualización.

- Tronco (*trunk*): es el dispositivo central que presta un servicio de HTTP y que almacena la información y estado de todos los dispositivos pertenecientes al grupo. Este dispositivo permanece a la escucha de las peticiones de los demás en el grupo y se encarga de distribuir la información según se le solicita. Este solamente entregará la información al dispositivo que se autentique con el grupo y la llave correctos.

- Representación (*representation*): esta será un conjunto del estado de un dispositivo y de sus propiedades ordenados en una estructura específica para su transmisión hacia y desde el servidor central o secundario.

Figura 8. **Topología de los dispositivos y programas**



Fuente: elaboración propia.

3.4.3. Formato de objetos

Cada rama tiene una representación virtual de sus propiedades y estado. Esta representación es transmitida entre los dispositivos por medio de estructuras estándar en la web, JSON o XML, según lo requiera el sistema. Sin embargo, el formato JSON, es siempre preferible ante XML ya que el primero es más eficiente

en cuanto a la cantidad de texto por información que se envía y es utilizado con mayor frecuencia en las aplicaciones en la *web* de los años recientes.

Existirán tres representaciones de una rama que pueden ser enviadas en una petición o una respuesta dependiendo de la información que se necesite.

La primera representación, la completa, muestra las propiedades de la rama y su estado, es útil cuando se quiere obtener o enviar la información completa de una rama. Sin embargo, esta representación es la que más texto transmite a través del medio, por tanto, debe ser utilizada únicamente cuando sea necesario obtener toda la información.

La segunda representación, la de los metadatos, muestra únicamente las propiedades de una rama. Esta representación es útil cuando se desea modificar o consultar alguna propiedad de una rama en operación. Transmite ligeramente menos texto que la primera, puesto que no se transmite el estado de la rama, sin embargo, no es recomendable transmitirla de forma periódica ya que las propiedades de un dispositivo no son datos que se modifican con alta frecuencia.

La tercera representación, la del estado, muestra únicamente el estado de la rama. Esta es la representación que menos cantidad de texto transmite, y esto debe ser así, ya que esta representación necesita ser enviada de forma periódica a través de la red para que cada rama pueda obtener una actualización de su estado.

Una rama, generalmente, solo envía o recibe la representación de sí misma, no obstante, los nudos, cuando los existe, envían y reciben la representación de un conjunto de ramas subordinadas. Por tanto, para cada representación que se detalle a continuación se incluirá una versión para una sola rama y otra para varias ramas.

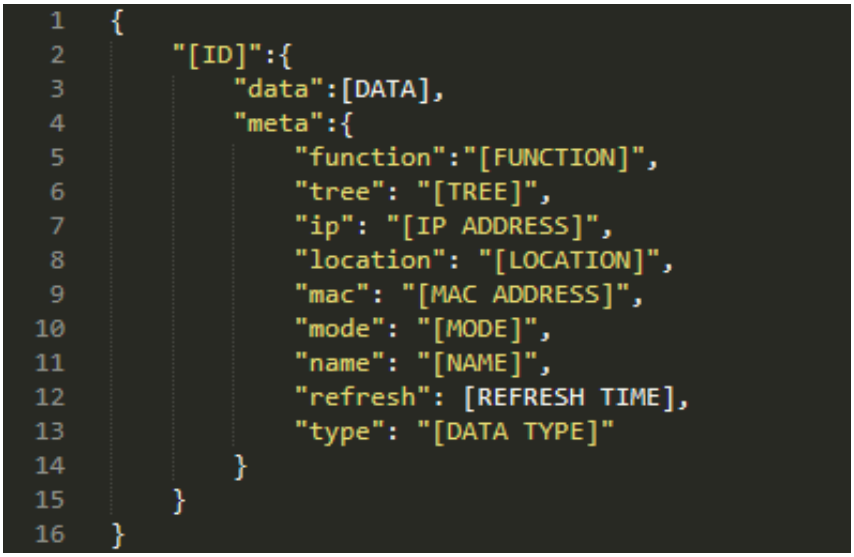
Nótese que los datos modificables aparecen entre corchetes y en mayúscula, mientras los nombres de los campos se escriben en minúscula, siempre entre comillas dobles, solo llevan una palabra y están escritos en inglés debido a que este idioma es el que predomina en las tecnologías de la información.

Las bibliotecas que manejan los objetos, normalmente, ordenan los datos de forma alfabética, sin embargo, esto no es necesario ya que el acceso a los datos, generalmente, se hace por medio de clave y no por posición.

3.4.3.1. JSON

A continuación, se presentan las tres representaciones con formato de objeto JSON para una y múltiples ramas.

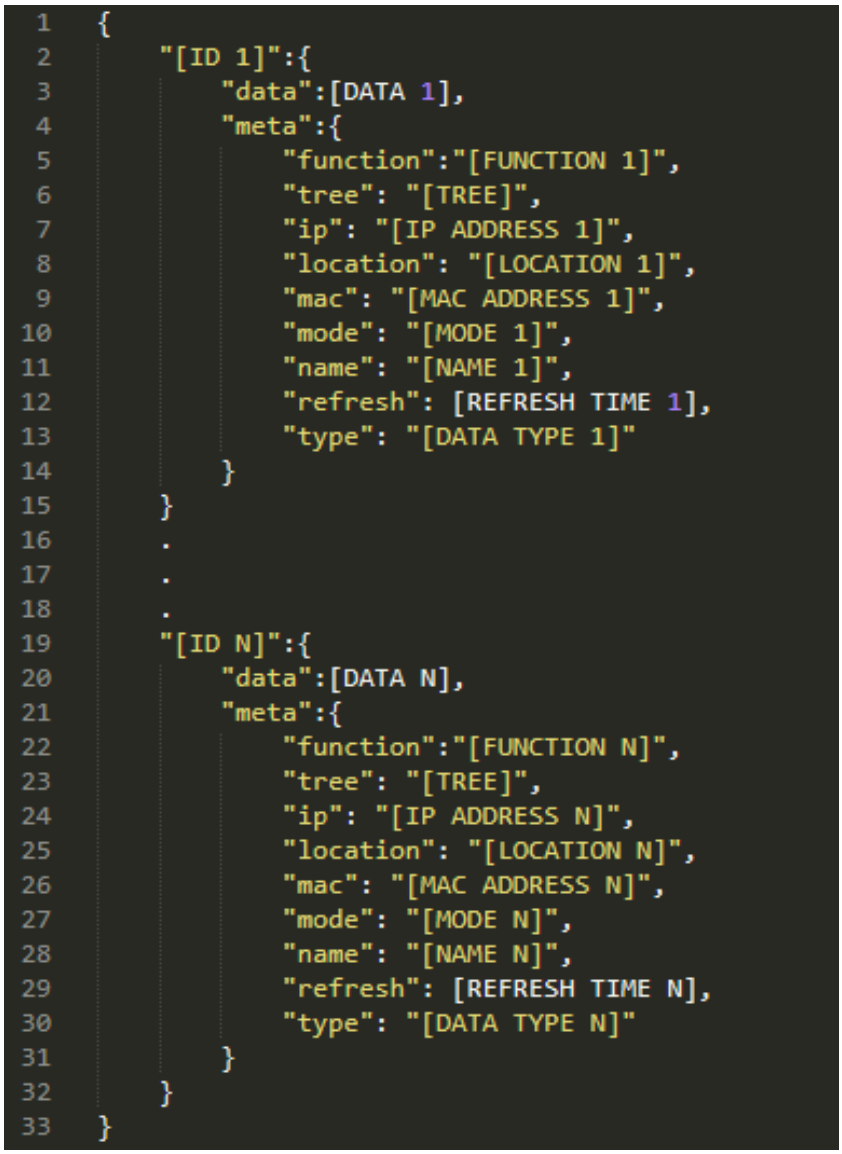
Figura 9. **Representación completa con objeto JSON para una rama**



```
{
    "[ID]":{
        "data":[DATA],
        "meta":{
            "function":"[FUNCTION]",
            "tree": "[TREE]",
            "ip": "[IP ADDRESS]",
            "location": "[LOCATION]",
            "mac": "[MAC ADDRESS]",
            "mode": "[MODE]",
            "name": "[NAME]",
            "refresh": [REFRESH TIME],
            "type": "[DATA TYPE]"
        }
    }
}
```

Fuente: elaboración propia.

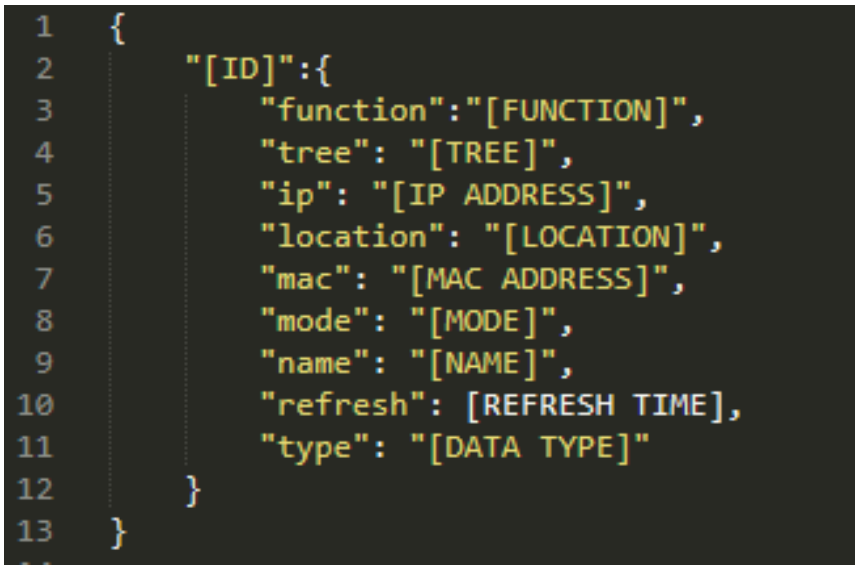
Figura 10. **Representación completa de objeto JSON para múltiples ramas**



```
{
    "[ID 1]":{
        "data":[DATA 1],
        "meta":{
            "function":"[FUNCTION 1]",
            "tree": "[TREE]",
            "ip": "[IP ADDRESS 1]",
            "location": "[LOCATION 1]",
            "mac": "[MAC ADDRESS 1]",
            "mode": "[MODE 1]",
            "name": "[NAME 1]",
            "refresh": [REFRESH TIME 1],
            "type": "[DATA TYPE 1]"
        }
    }
    .
    .
    .
    "[ID N]":{
        "data":[DATA N],
        "meta":{
            "function":"[FUNCTION N]",
            "tree": "[TREE]",
            "ip": "[IP ADDRESS N]",
            "location": "[LOCATION N]",
            "mac": "[MAC ADDRESS N]",
            "mode": "[MODE N]",
            "name": "[NAME N]",
            "refresh": [REFRESH TIME N],
            "type": "[DATA TYPE N]"
        }
    }
}
```

Fuente: elaboración propia.

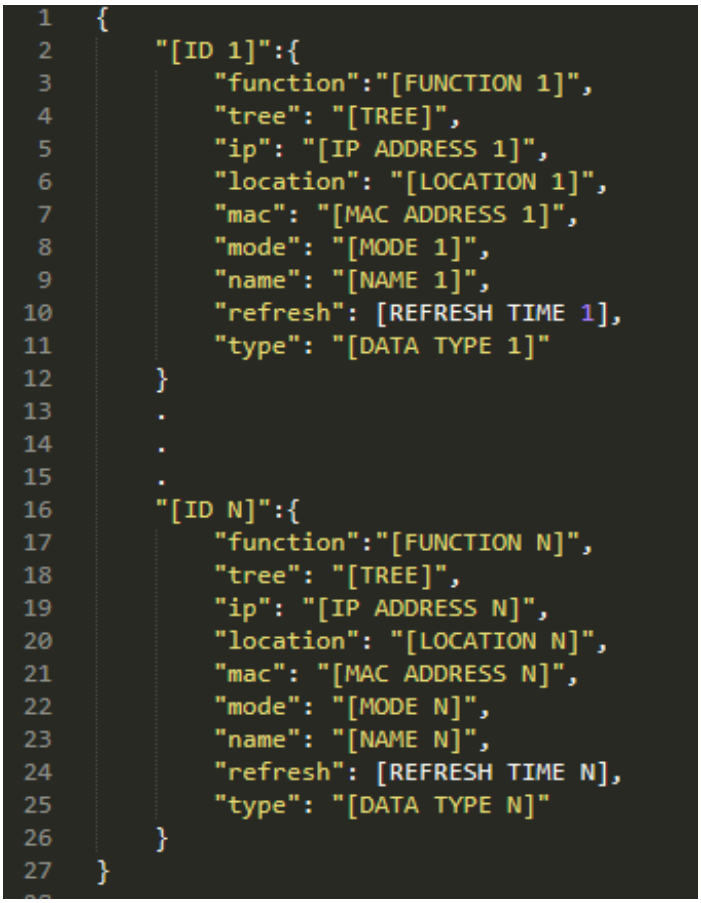
Figura 11. **Representación de metadatos con objeto JSON para una rama**



```
{
    "[ID]":{
        "function":"[FUNCTION]",
        "tree": "[TREE]",
        "ip": "[IP ADDRESS]",
        "location": "[LOCATION]",
        "mac": "[MAC ADDRESS]",
        "mode": "[MODE]",
        "name": "[NAME]",
        "refresh": [REFRESH TIME],
        "type": "[DATA TYPE]"
    }
}
```

Fuente: elaboración propia.

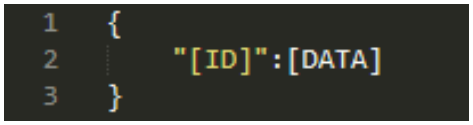
Figura 12. **Representación de metadatos con objeto JSON para múltiples ramas**



```
{
    "[ID 1]":{
        "function":"[FUNCTION 1]",
        "tree": "[TREE]",
        "ip": "[IP ADDRESS 1]",
        "location": "[LOCATION 1]",
        "mac": "[MAC ADDRESS 1]",
        "mode": "[MODE 1]",
        "name": "[NAME 1]",
        "refresh": [REFRESH TIME 1],
        "type": "[DATA TYPE 1]"
    }
    .
    .
    .
    "[ID N]":{
        "function":"[FUNCTION N]",
        "tree": "[TREE]",
        "ip": "[IP ADDRESS N]",
        "location": "[LOCATION N]",
        "mac": "[MAC ADDRESS N]",
        "mode": "[MODE N]",
        "name": "[NAME N]",
        "refresh": [REFRESH TIME N],
        "type": "[DATA TYPE N]"
    }
}
```

Fuente: elaboración propia.

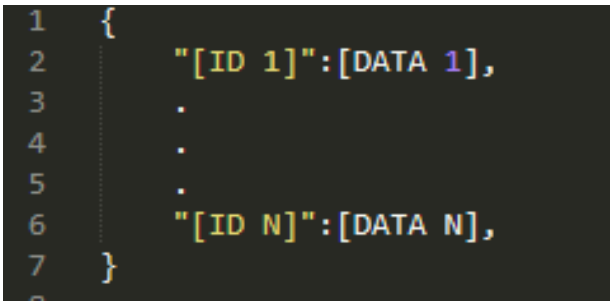
Figura 13. **Representación de estado con objeto JSON para para una rama**



```
{
    "[ID]":[DATA]
}
```

Fuente: elaboración propia.

Figura 14. **Representación de estado con objeto JSON para múltiples ramas**



Fuente: elaboración propia.

3.4.3.2. XML

A continuación, se presentan las tres representaciones con formato de objeto XML para una y múltiples ramas.

Figura 15. **Representación completa con objeto XML para una rama**

Fuente: elaboración propia.

Figura 16. **Representación completa con objeto XML para múltiples ramas**

```xml
<?xml version="1.0"?>
<devices>
    <[ID 1]>
        <data>[DATA 1]</data>
        <meta>
            <function>[FUNCTION 1]</function>
            <tree>[TREE]</tree>
            <ip>[IP ADDRESS 1]</ip>
            <location>[LOCATION 1]</location>
            <mac>[MAC ADDRESS 1]</mac>
            <mode>[MODE 1]</mode>
            <name>[NAME 1]</name>
            <refresh>[REFRESH TIME 1]</refresh>
            <type>[DATA TYPE 1]</type>
        </meta>
    </[ID 1]>
    .
    .
    .
    <[ID N]>
        <data>[DATA N]</data>
        <meta>
            <function>[FUNCTION N]</function>
            <tree>[TREE]</tree>
            <ip>[IP ADDRESS N]</ip>
            <location>[LOCATION N]</location>
            <mac>[MAC ADDRESS N]</mac>
            <mode>[MODE N]</mode>
            <name>[NAME N]</name>
            <refresh>[REFRESH TIME N]</refresh>
            <type>[DATA TYPE N]</type>
        </meta>
    </[ID N]>
</devices>
```

Fuente: elaboración propia.

Figura 17. **Representación de metadatos con objeto XML para una rama**

```xml
<?xml version="1.0"?>
<devices>
    <[ID]>
        <function>[FUNCTION]</function>
        <tree>[TREE]</tree>
        <ip>[IP ADDRESS]</ip>
        <location>[LOCATION]</location>
        <mac>[MAC ADDRESS]</mac>
        <mode>[MODE]</mode>
        <name>[NAME]</name>
        <refresh>[REFRESH TIME]</refresh>
        <type>[DATA TYPE]</type>
    </[ID]>
</devices>
```

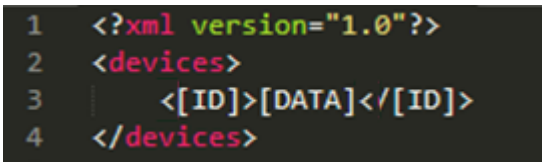
Fuente: elaboración propia.

Figura 18. **Representación de metadatos con objeto XML para múltiples ramas**

```xml
<?xml version="1.0"?>
<devices>
    <[ID 1]>
        <function>[FUNCTION 1]</function>
        <tree>[TREE]</tree>
        <ip>[IP ADDRESS 1]</ip>
        <location>[LOCATION 1]</location>
        <mac>[MAC ADDRESS 1]</mac>
        <mode>[MODE 1]</mode>
        <name>[NAME 1]</name>
        <refresh>[REFRESH TIME 1]</refresh>
        <type>[DATA TYPE 1]</type>
    </[ID 1]>
    .
    .
    .
    <[ID N]>
        <function>[FUNCTION N]</function>
        <tree>[TREE]</tree>
        <ip>[IP ADDRESS N]</ip>
        <location>[LOCATION N]</location>
        <mac>[MAC ADDRESS N]</mac>
        <mode>[MODE N]</mode>
        <name>[NAME N]</name>
        <refresh>[REFRESH TIME N]</refresh>
        <type>[DATA TYPE N]</type>
    </[ID N]>
</devices>
```

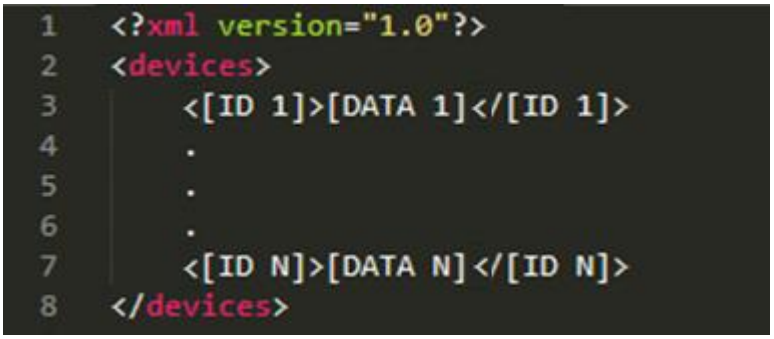
Fuente: elaboración propia.

Figura 19. **Representación de estado con objeto XML para una rama**



Fuente: elaboración propia.

Figura 20. **Representación de estado con objeto XML para múltiples ramas**



Fuente: elaboración propia.

3.4.3.3. Campos

En esta lista se detalla el significado de los datos modificables que se deben colocar en los campos de los objetos JSON y XML para la representación de una rama. Aunque la codificación de los datos para su presentación se realiza en formato ASCII, es preferible, aunque no obligatorio, que los datos contengan únicamente caracteres comunes al alfabeto español e inglés para mejorar su compatibilidad con aplicaciones multilenguaje.

- [ID]: es un identificador que le permitirá al sistema organizar las ramas de forma jerárquica. Este siempre se debe escribir entre comillas, aunque sea solamente un carácter numérico y el nivel jerárquico se separará con un guion medio. Ejemplos son: “1”, “2-3-5”, “5-1”.

- [DATA]: es el estado del dispositivo, ya sea el valor que obtiene por medio de un sensor o el valor que exterioriza por medio de un actuador o dispositivo de visualización. Si los datos que se envían son de tipo número o binario, se deben escribir sin comillas, pero si son de tipo texto, se deberán escribir entre comillas. Ejemplos son: *true*, “ABC” y 145.

- [FUNCTION]: es una corta descripción de lo que realiza el dispositivo o de lo que mide. Esta información siempre debe escribirse dentro de comillas. Ejemplos son: “Luces del exterior”, “Ventilador principal” e “Intensidad de luz”.

- [TREE]: es el nombre del árbol, es decir, del conjunto de dispositivos. Siempre debe escribirse dentro de comillas. La comprobación de este valor es únicamente útil cuando un servidor funciona como tronco o nudo de dos o más árboles a la vez.

- [IP ADDRESS]: esta es la dirección IP del dispositivo en la red. Esta información la obtiene al conectarse a una red local. Siempre debe escribirse entre comillas.

- [LOCATION]: este dato representa una descripción corta de la ubicación física del dentro de la edificación. Es útil para localizar a un dispositivo en caso de falla. Siempre se debe escribir entre comillas. Ejemplos son: “Sala”, “Dormitorio principal” y “Oficina 3 segundo nivel”.

- [MAC ADDRESS]: es la dirección MAC que posee el dispositivo. Esta información la puede obtener por medio de software o puede brindarla el fabricante del dispositivo. Se debe escribir siempre entre comillas.

- [MODE]: es el modo de operación de la rama, ya sea como dispositivo de entrada o de salida. Siempre se debe escribir entre comillas y en mayúscula y solamente puede tomar uno de dos valores: "*INPUT*" o "*OUTPUT*", dependiendo si es un dispositivo de entrada o de salida respectivamente.

- [NAME]: es un nombre que puede colocar el usuario para poder identificar los dispositivos con mayor facilidad. Se debe escribir siempre entre comillas. Ejemplos son: "Sensor de luz 3", "ESP8266" y "2Nivel_Sala_Lampara_3".

- [REFRESH TIME]: es el tiempo en segundos que esperará cada rama para pedir actualización de su estado. Este número siempre será entero, nunca se debe escribir entre comillas y no se acompaña de una dimensional. Ejemplos son: 2, 5 y 1.

- [DATA TYPE]: este es un indicador del tipo de datos que se envían en el estado de la rama. Es útil para identificar a nivel de software la información a manejar. Siempre se escribe entre comillas y en minúscula Este dato puede tomar uno de tres valores: "*number*" si es un número, "*bool*" si es un dato binario, y "*text*" si son caracteres alfanuméricos.

3.4.3.4. Identificación

Cada una de las ramas poseerá un identificador jerárquico único, que permitirá la transmisión agrupada de objetos entre los nudos y el tronco. Los nudos también se identificarán, pero su identificador no es enviado en ningún

objeto JSON o XML sino en la petición hacia el tronco. El único dispositivo que no posee identificador es el tronco, aunque, si por necesidades de programación se necesitara, se podría identificar como “0”.

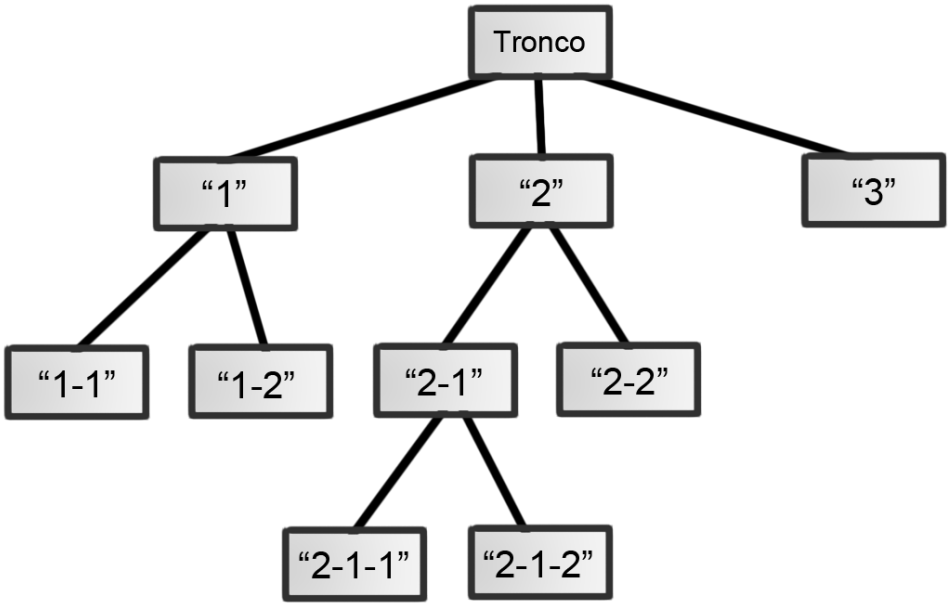
Este identificador está conformado por números naturales entre uno y doscientos cincuenta y cinco, para que ocupen únicamente un byte, que puede indicar el orden de adición al tronco o al nudo, aunque no es obligatorio, pues pueden ser elegidos por el administrador del sistema a conveniencia, y guiones medios que separan los niveles de jerarquía. Estos siempre deberán ser escritos entre comillas, aunque estén conformados únicamente por un número.

Los dispositivos, sean ramas o nudos que se agreguen al tronco poseerán los identificadores entre “1” y “255”. Además del límite numérico, no existe un límite práctico al número de dispositivos que se agreguen al tronco de forma directa, pero se debe considerar que un número elevado de ramas conectadas al tronco aumentará el número de peticiones por minuto que se realizarán al servidor central, por tanto, el tronco debe tener suficiente capacidad para atenderlas antes de sufrir una denegación de servicio.

Los dispositivos, sean nudos o ramas, que se agreguen en un nivel jerárquico inferior bajo algún nudo tendrán un identificador conformado por el identificador del nudo seguido de un guion y un número entre uno y doscientos cincuenta y cinco. Esta regla se aplica para todos los niveles inferiores.

En la siguiente figura se muestra un ejemplo de cómo se asignarían los identificadores hasta tres niveles jerárquicos inferiores.

Figura 21. **Identificación en jerarquía**



Fuente: elaboración propia.

3.4.4. Formato de peticiones

Las peticiones que se realicen al servidor central en el tronco llevan un formato de petición de HTTP. Esta petición debe contener encabezados, cada uno en una línea, seguido de una línea en blanco, y el cuerpo de la petición que contendrá la representación de una rama en formato de objeto JSON o XML.

Los encabezados que debe contener de forma obligatoria cada petición que se envíe al tronco, o hacia algún nudo, son los siguientes:

- *Host*: este encabezado es obligatorio en cada petición de HTTP a partir de la versión 1.1. Debe indicar la dirección IP o el nombre del dominio donde se encuentra alojado el servidor al que se realiza la petición.

- *Authorization*: este encabezado indica el árbol al que pertenece el dispositivo y la llave que utiliza para autenticarse como parte del árbol. Debe contener la palabra “Basic” seguida de un espacio e incluir las credenciales en formato “árbol: llave” codificados en sistema Base64.

- *Content-Type*: este encabezado solamente es obligatorio cuando se requiere enviar algún cuerpo en la petición. Indica el tipo de contenido que se enviará y podrá ser “*application/json*”, si se envía una representación de rama en formato de objeto JSON, o “*application/xml*”, si se usa el formato de objeto XML.

- *Content-Length*: este encabezado indica la longitud en bytes del contenido en el cuerpo de la petición. Es útil para indicarle al servidor el espacio que debe reservar para obtener todos los datos de la petición.

- *Accept*: este encabezado no es obligatorio, pero es útil cuando se desea especificar el formato de objeto que se desea recibir. Su valor puede ser “*application/json*”, si se necesita recibir una representación de rama en formato de objeto JSON, o “*application/xml*”, si se necesita el formato de objeto XML. Si este encabezado no aparece o tiene otro valor, el formato de objeto por defecto será JSON.

Cada respuesta que genere un servidor debe contener los siguientes encabezados:

- *Content-Length*: este encabezado indica la cantidad de bytes que contiene la respuesta.

- *WWW-Authenticate*: este se envía en la respuesta solamente cuando se realiza una petición sin enviar las credenciales de autenticación. Indica que es necesaria una autenticación para obtener el contenido solicitado y contiene.

- *Content-Type*: este encabezado solamente es útil cuando se requiere enviar algún cuerpo en la respuesta. Indica el tipo de contenido que se enviará y podrá ser "*application/json*", si se envía una representación de rama en formato de objeto JSON, o "*application/xml*", si se usa el formato de objeto XML. Si no se indica una de esas opciones, el cliente podrá asumir que la representación se envía en formato JSON.

3.4.5. Autenticación del protocolo de IoT

Un dispositivo debe identificarse como parte del árbol para tener acceso a la información del sistema. La autenticación permite que un dispositivo solamente pueda modificar la información de su árbol y no exista confusión cuando un servidor funciona como nudo o tronco de diferentes árboles. En redes privadas permite el control del sistema únicamente a quien posea las credenciales de autenticación.

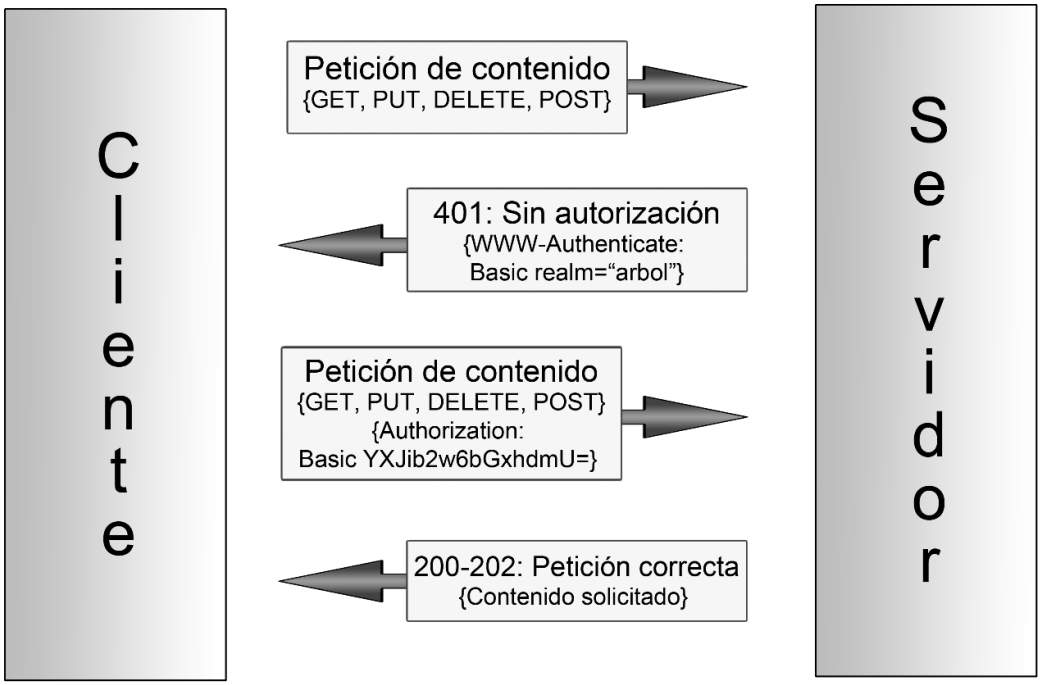
El administrador del árbol asignará un nombre al árbol y una llave, estas serán las credenciales de autenticación.

La autenticación en este protocolo no se realiza con el objetivo de brindar alta seguridad, por lo que no debe considerarse como una medida de seguridad suficiente cuando se implementa en redes públicas.

Para autenticarse, una rama, nudo o raíz deberá enviar sus credenciales en formato árbol: llave codificados en sistema Base64, en cada petición a cualquier servidor, con el encabezado "*Authorization*" seguido de la palabra "*Basic*", como indica la especificación de HTTP para el esquema de autenticación básica.

En caso de que no se envíen las credenciales correctas, el servidor deberá enviar una respuesta con código 401, indicando que se necesita autenticación para acceder a la información. Con el encabezado "*WWW-Authenticate*" mostrará la palabra Basic y entre comillas, el nombre del árbol al que se necesita tener acceso.

Figura 22. **Proceso de autenticación de los dispositivos**



Fuente: elaboración propia.

3.4.6. Procesos

A continuación, se detallarán los procedimientos que se llevarán a cabo en la comunicación de los dispositivos para la transmisión de la representación de las ramas. La comunicación entre los dispositivos, al funcionar con un modelo cliente/servidor, se realiza mediante peticiones de HTTP enviadas al tronco con una URI y contenido específicos del procedimiento. Cada petición debe incluir en el encabezado las credenciales de autenticación y el servidor debe verificarlas antes de entregar o modificar cualquier contenido sobre los dispositivos.

3.4.6.1. Agregar un dispositivo al conjunto

Para que una rama se pueda agregar al árbol, debe enviar una petición de HTTP hacia el nudo inmediatamente superior, o si no lo hubiera, hacia el tronco, con la representación completa de sí misma en el cuerpo de la petición, utilizando el método de acceso POST e indicando en un encabezado el formato de objeto utilizado para la representación.

La petición se realiza con el URI “http://[SERVER]/devices/”, donde [SERVER] es la dirección IP o nombre de dominio del servidor del nudo o tronco, según sea el caso. Esta petición también puede ser enviada por un administrador del árbol, de forma manual, sustituyendo el comportamiento de la rama si fuera necesario. Es obligatorio que se indique con el encabezado “*Content-Type*” el formato de objeto de la representación de la rama, utilizando “*application/json*” si se envía con el formato JSON o “*application/xml*” si se envía con el formato XML.

Se recomienda que esta petición únicamente se envíe al encender el dispositivo ya que solamente es necesario agregar la rama al árbol una vez.

Por el lado del servidor, independientemente si está en un nudo o el tronco, al recibir la petición con el método POST, debe almacenar las representaciones completas de las ramas que aún no pertenecen al árbol que se envíen en el cuerpo de la petición. Cada servidor debe permitir que una rama con un identificador especificado se pueda agregar una sola vez.

Si entre las representaciones de ramas que recibe en el cuerpo de la petición se encuentra alguna con el identificador de una rama que ya ha sido añadida al árbol, esta representación debe ser ignorada y la rama no se agregará

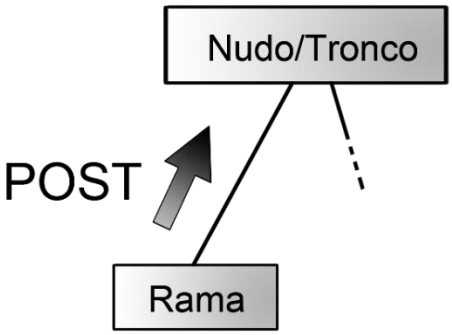
de nuevo ni se modificará. Esto permitirá que, aunque la petición sea enviada varias veces, las representaciones almacenadas se mantengan sin cambios.

Una rama puede agregarse de forma local, cuando un nudo guarda su representación, o de forma global, cuando su representación es almacenada en el tronco.

Si alguna rama es agregada al árbol con éxito, sea de forma local o global, el servidor que la agregó debe devolver una respuesta con código de estado 201, indicando que la petición fue aceptada y los elementos fueron agregados. Si ninguna rama es añadida al árbol, pero la petición fue realizada correctamente, el servidor debe devolver una respuesta con código de estado 202, indicando que la petición fue aceptada, pero las ramas no se duplicaron.

Los nudos, cuando agregan alguna rama al árbol de forma local, deben mandar de forma automática la petición a su nudo inmediatamente superior para que la rama pueda ser agregada y la petición reenviada hasta alcanzar el tronco y así lograr agregar la rama de forma global. De esta manera, todos los demás dispositivos pueden conocer de su pertenencia al árbol.

Figura 23. **Ejemplo de petición para agregar una rama al árbol**




```
POST /devices HTTP/1.1
Host: 192.168.0.10
Content-Length: 251
Content-Type: application/json
Authorization: Basic Z3JvdXA6a2V5

{
    "2": {
        "meta": {
            "function": "Ejemplificar",
            "mac": "80:7D:3A:3D:95:50",
            "location": "Laboratorio",
            "name": "Dispositivo de ejemplo",
            "type": "boolean",
            "ip": "192.168.0.104",
            "tree": "Arbol de ejemplo",
            "refresh": 2,
            "mode": "OUTPUT"
        },
        "data": false
    }
}
```

Fuente: elaboración propia.

3.4.6.2. Remover un dispositivo del conjunto

La remoción de una rama del árbol puede iniciarse de varias maneras, ya sea que la misma rama solicite ser retirada del árbol, otro dispositivo lo solicite o el administrador del árbol lo haga. En cualquiera de los casos, esta remoción iniciará con una petición de HTTP hacia el nudo inmediatamente superior, o si no lo hubiera, hacia el tronco, con el método de acceso DELETE y el cuerpo de la petición en blanco.

La petición se envía con el URI “http://[SERVER]/devices/[ID]”, donde [SERVER] es la dirección IP o nombre de dominio del servidor del nudo o tronco, según sea el caso, y [ID] es el identificador del dispositivo que será removido del

árbol. Este último puede ser el identificador de un nudo, caso que solicita la remoción de todos los dispositivos subordinados al nudo indicado.

Por el lado del servidor, independientemente de si está en un nudo o en el tronco, al recibir la petición con el método de acceso DELETE, este debe eliminar, de sus datos guardados, la representación completa de la rama que indica el identificador en el URI. Si el identificador pertenece a un nudo, el servidor debe remover todos los dispositivos que estén subordinados al nudo.

Una rama, o conjunto de ellas, pueden eliminarse de forma local, si solo se remueve de un nudo, o de forma global, si se retira del tronco.

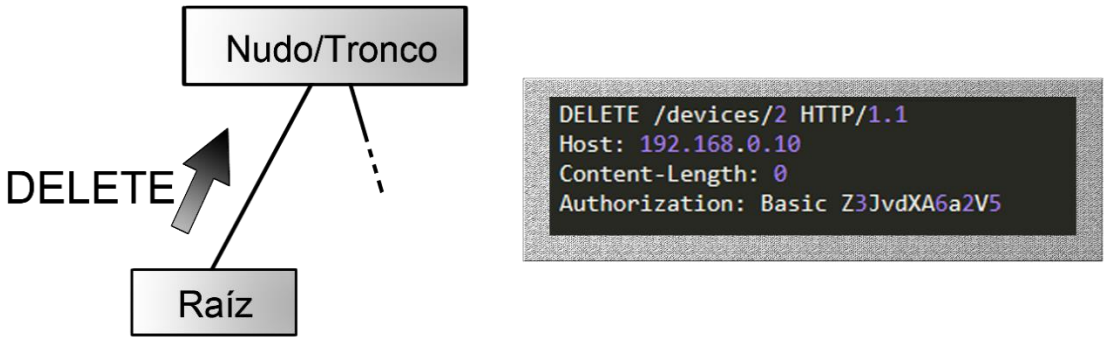
El servidor, al eliminar exitosamente una rama ya sea de forma local o global, debe responder con un código de estado 200, indicando que la petición se realizó de forma correcta y tuvo éxito. Si no se puede remover porque no existe la representación de la rama, debe responder con un código de estado 202, indicando que la petición se realizó correctamente pero no existe el contenido al que se hace referencia con el identificador.

Cuando una rama es eliminada de forma local, el nudo debe reenviar automáticamente la petición hacia su nudo inmediatamente superior para que realice la misma acción y continuar con el mismo procedimiento hasta alcanzar al tronco y lograr eliminar la rama de forma global. Sin embargo, esta acción elimina la rama solamente en los dispositivos superiores jerárquicamente al dispositivo donde se origina la petición.

Cuando un nudo solicita una representación de un dispositivo o conjunto de dispositivos y el servidor devuelve un elemento vacío o un conjunto con ausencia de cierto elemento, la representación faltante debe ser eliminada de su lista

guardada de representaciones. De esta forma se eliminan las ramas en dispositivos jerárquicamente inferiores.

Figura 24. **Ejemplo de petición para eliminar una rama del árbol**



Fuente: elaboración propia.

3.4.6.3. Obtener la información de un dispositivo

Un dispositivo o el usuario pueden solicitar toda la información sobre una o varias ramas. Esto se realizará enviado una petición HTTP con el método de acceso GET hacia el nudo jerárquicamente superior a las ramas en cuestión o hacia el tronco, el que esté más cerca de quien envía la petición.

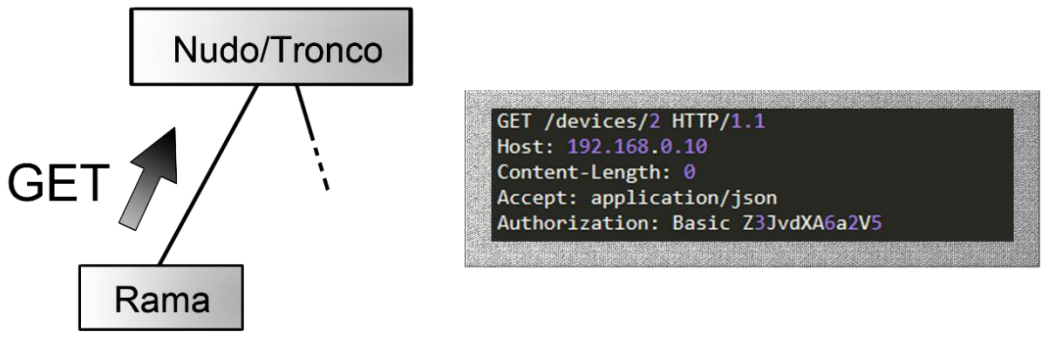
La petición se realiza con el URI “http://[SERVER]/devices/[ID]”, donde [SERVER] es la dirección IP o nombre de dominio del servidor del nudo o tronco, según sea el caso, y [ID] es el identificador del dispositivo sobre el que se solicita la información. El identificador puede pertenecer a una rama específica o a un nudo, caso que significará la solicitud de la información de todas sus ramas subordinadas.

El servidor, al recibir una petición con el método de acceso GET y el URI especificado, enviará una respuesta con el código de estado 200, indicando que la solicitud de realizó de forma correcta. En el cuerpo de la respuesta deberá incluir las representaciones completas de las ramas que indique el identificador en el URI. De no existir las representaciones de las ramas solicitadas deberá enviar un conjunto vacío.

El dispositivo que solicita la información podrá incluir el encabezado "*Accept*" indicando el formato de objeto que desea recibir, utilizando "application/json" si se solicita el formato JSON o "*application/xml*" si se solicita el formato XML. En caso de que no se especifique este dato, el servidor debe enviar la representación en formato JSON.

De esta manera se puede obtener toda la información de un dispositivo, sin embargo, este procedimiento no debe usarse para obtener solamente el estado de un dispositivo, pues la representación completa envía mucha más información a través del enlace.

Figura 25. **Ejemplo de petición de la información completa de una rama**



Fuente: elaboración propia.

Figura 26. **Ejemplo de respuesta a una petición de la información completa de una rama**

Fuente: elaboración propia.

3.4.6.4. Obtener el estado de un dispositivo

Un dispositivo o el usuario pueden solicitar el estado de una rama para su operación, ya sea que funcione como dispositivo de entrada o salida. Para realizar eso debe enviar una petición HTTP con el método de acceso GET hacia el nudo jerárquicamente superior a las ramas en cuestión o hacia el tronco, el que esté más cerca de quien envía la petición.

La petición tendrá el URI “http://[SERVER]/devices/[ID]/data”, donde [SERVER] es la dirección IP o nombre de dominio del servidor del nudo o tronco, según sea el caso, y [ID] es el identificador del dispositivo sobre el que se solicita la información. El identificador puede pertenecer a una rama específica o a un

nudo, caso que significará la solicitud de la información de todas sus ramas subordinadas.

El servidor, al recibir una petición con el método de acceso GET y el URI especificado, enviará una respuesta con el código de estado 200, indicando que la solicitud de realizó de forma correcta. En el cuerpo de la respuesta deberá incluir las representaciones de estado de las ramas que indique el identificador en el URI. De no existir las representaciones de las ramas solicitadas deberá enviar un conjunto vacío.

El dispositivo que solicita la información podrá incluir el encabezado "*Accept*" indicando el formato de objeto que desea recibir, utilizando "*application/json*" si se solicita el formato JSON o "*application/xml*" si se solicita el formato XML. En caso de que no se especifique este dato, el servidor debe enviar la representación en formato JSON.

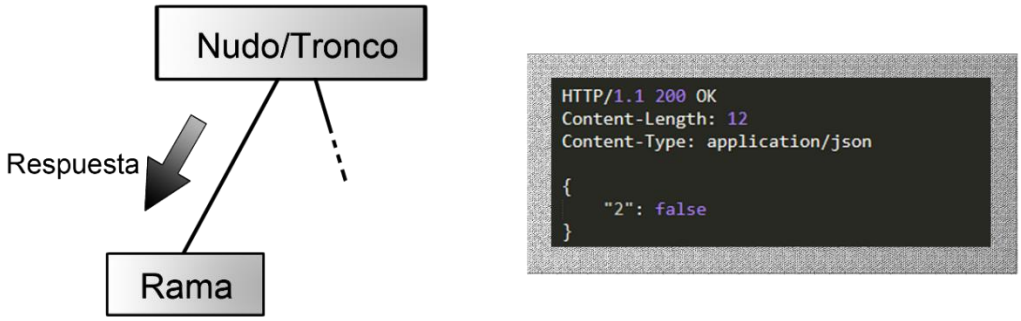
Cada dispositivo de salida debe enviar esta petición de forma periódica para obtener su estado, con un tiempo entre peticiones igual al que indica el campo "*refresh*" en sus metadatos.

Cada nudo debe enviar esta solicitud de forma periódica hacia su nudo jerárquicamente superior, o tronco, con un tiempo entre peticiones igual al menor tiempo que indiquen los campos "*refresh*" de sus dispositivos de salida subordinados. De esta forma se asegura que la información se propague hacia las ramas en el menor tiempo.

Figura 27. **Ejemplo de petición del estado de una rama**

Fuente: elaboración propia.

Figura 28. **Ejemplo de respuesta a petición del estado de una rama**



Fuente: elaboración propia.

3.4.6.5. Cambiar el estado de un dispositivo

Un dispositivo o el usuario pueden cambiar el estado de una rama. El estado a cambiar puede ser de la misma rama que lo solicita en el caso de los

dispositivos de entrada. Para realizar eso debe enviar una petición HTTP con el método de acceso PUT hacia el nudo jerárquicamente superior a las ramas en cuestión o hacia el tronco, el que esté más cerca de quien envía la petición.

La petición tendrá el URI “http://[SERVER]/devices/data”, donde [SERVER] es la dirección IP o nombre de dominio del servidor del nudo o tronco, según sea el caso. En el cuerpo de la petición se debe enviar las representaciones de estado de las ramas que se actualizarán, además es obligatorio que se indique con el encabezado “*Content-Type*” el formato de objeto de la representación de la rama, utilizando “*application/json*” si se envía con el formato JSON o “*application/xml*” si se envía con el formato XML.

Por el lado del servidor, independientemente si está en un nudo o el tronco, al recibir una petición con el método de acceso PUT y el URI especificado, deberá actualizar el estado de los dispositivos cuyas representaciones se encuentren en el cuerpo de la petición y que existan en el árbol. Si entre las representaciones de ramas que recibe en el cuerpo de la petición se encuentra alguna con el identificador de una rama que no ha sido añadida al árbol, esta representación debe ser ignorada y la rama no debe agregarse al árbol.

El estado de una rama puede actualizarse de forma local, cuando un nudo guarda el cambio de su estado, o de forma global, cuando su actualización almacenada en el tronco.

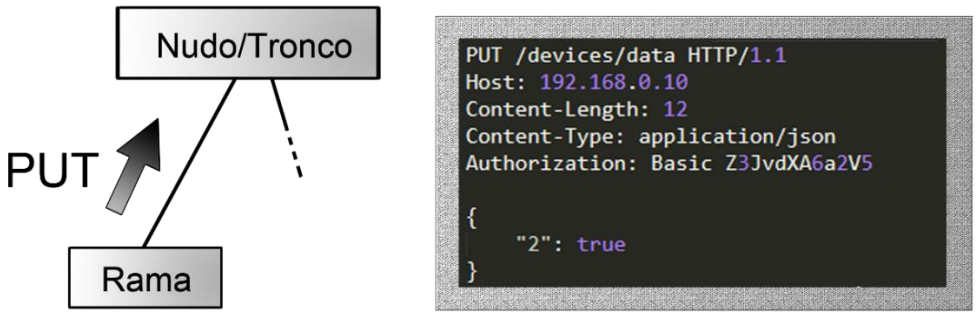
Si el estado de una rama es actualizado con éxito, sea de forma local o global, el servidor que guardo el cambio debe devolver una respuesta con código de estado 201, indicando que la petición fue aceptada y los estados fueron modificados. Si ningún estado es modificado porque no existen las ramas en el árbol, pero la petición fue realizada correctamente, el servidor debe devolver una

respuesta con código de estado 202, indicando que la petición fue aceptada, pero los cambios no se realizaron.

Cada dispositivo de entrada debe enviar esta petición de forma periódica para actualizar su estado, con un tiempo entre peticiones igual al que indica el campo “*refresh*” en sus metadatos.

Cada nudo debe enviar esta solicitud de forma periódica hacia su nudo jerárquicamente superior, o tronco, con un tiempo entre peticiones igual al menor tiempo que indiquen los campos “*refresh*” de sus dispositivos de entrada subordinados. De esta forma se asegura que la información se propague hacia el tronco en el menor tiempo.

Figura 29. **Ejemplo de petición para cambiar el estado de una rama**



Fuente: elaboración propia.

3.4.6.6. Obtener los metadatos de un dispositivo

Un dispositivo o el usuario pueden solicitar los metadatos de una rama. Para realizar eso debe enviar una petición HTTP con el método de acceso GET hacia

el nudo jerárquicamente superior a las ramas en cuestión o hacia el tronco, el que esté más cerca de quien envía la petición.

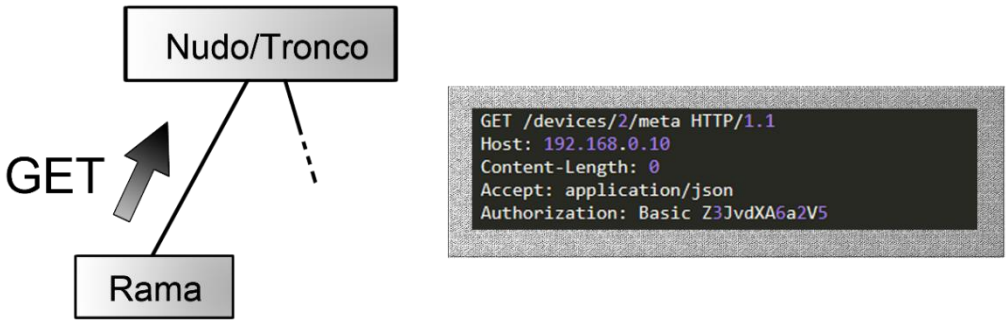
La petición tendrá el URI “http://[SERVER]/devices/[ID]/metadata”, donde [SERVER] es la dirección IP o nombre de dominio del servidor del nudo o tronco, según sea el caso, y [ID] es el identificador del dispositivo sobre el que se solicita la información. El identificador puede pertenecer a una rama específica o a un nudo, caso que significará la solicitud de la información de todas sus ramas subordinadas.

El servidor, al recibir una petición con el método de acceso GET y el URI especificado, enviará una respuesta con el código de estado 200, indicando que la solicitud de realizó de forma correcta. En el cuerpo de la respuesta deberá incluir las representaciones de metadatos de las ramas que indique el identificador en el URI. De no existir las representaciones de las ramas solicitadas deberá enviar un conjunto vacío.

El dispositivo que solicita la información podrá incluir el encabezado “*Accept*” indicando el formato de objeto que desea recibir, utilizando “*application/json*” si se solicita el formato JSON o “*application/xml*” si se solicita el formato XML. En caso de que no se especifique este dato, el servidor debe enviar la representación en formato JSON.

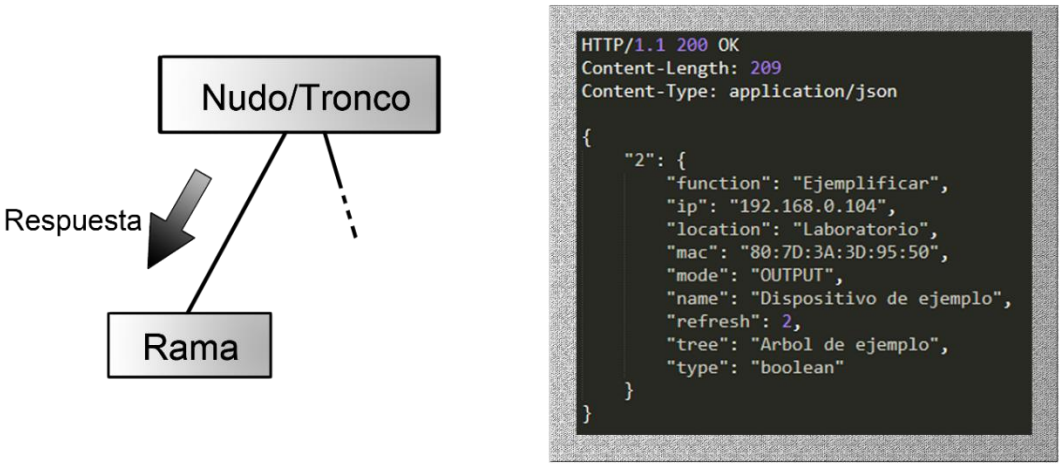
No se recomienda que esta petición se programe de forma periódica, pues no es frecuente que los metadatos de un dispositivo se modifiquen. Es recomendable que un dispositivo al encenderse solicite sus metadatos para modificar sus campos que han tenido un cambio. Esto obligaría a reiniciar un dispositivo cada vez que se solicita la actualización de sus metadatos. Sin embargo, esta recomendación no es obligatoria.

Figura 30. **Ejemplo de petición de los metadatos de una rama**



Fuente: elaboración propia.

Figura 31. **Ejemplo de respuesta a la petición de los metadatos de una rama**



Fuente: elaboración propia.

3.4.6.7. Editar los metadatos de un dispositivo

Un dispositivo o el usuario pueden cambiar los metadatos de una o varias ramas. Para realizar eso debe enviar una petición HTTP con el método de acceso PUT hacia el nudo jerárquicamente superior a las ramas en cuestión o hacia el tronco, el que esté más cerca de quien envía la petición.

La petición tendrá el URI “http://[SERVER]/devices/metadata”, donde [SERVER] es la dirección IP o nombre de dominio del servidor del nudo o tronco, según sea el caso. En el cuerpo de la petición se debe enviar las representaciones de metadatos de las ramas que se actualizarán, además es obligatorio que se indique con el encabezado “*Content-Type*” el formato de objeto de la representación de la rama, utilizando “*application/json*” si se envía con el formato JSON o “*application/xml*” si se envía con el formato XML.

Por el lado del servidor, independientemente si está en un nudo o el tronco, al recibir una petición con el método de acceso PUT y el URI especificado, deberá actualizar los metadatos de los dispositivos cuyas representaciones se encuentren en el cuerpo de la petición y que existan en el árbol. Si entre las representaciones de ramas que recibe en el cuerpo de la petición se encuentra alguna con el identificador de una rama que no ha sido añadida al árbol, esta representación debe ser ignorada y la rama no debe agregarse al árbol.

Los metadatos de una rama pueden actualizarse de forma local, cuando un nudo guarda el cambio, o de forma global, cuando su actualización almacenada en el tronco.

Si los metadatos de una rama son actualizados con éxito, sea de forma local o global, el servidor que guardo el cambio debe devolver una respuesta con

código de estado 201, indicando que la petición fue aceptada y los metadatos fueron modificados. Si los metadatos de ningún dispositivo son modificados porque no existen las ramas en el árbol, pero la petición fue realizada correctamente, el servidor debe devolver una respuesta con código de estado 202, indicando que la petición fue aceptada, pero los cambios no se realizaron.

Los nudos, cuando modifican los metadatos de alguna rama de forma local, deben mandar de forma automática la petición a su nudo inmediatamente superior para que la rama pueda ser modificada y la petición reenviada hasta alcanzar el tronco y así lograr actualizar los metadatos de la rama de forma global. De esta manera, todos los demás dispositivos pueden conocer la nueva información.

Figura 32. **Ejemplo de petición para editar los metadatos de una rama**

Fuente: elaboración propia.

3.4.7. Limitaciones

Según la especificación de este protocolo, no está diseñado para su implementación en:

- Aplicaciones que requieren un tiempo crítico de actualización, es decir que no permiten retrasos entre el envío de la instrucción y su ejecución.

- Dispositivos que envían gran cantidad de datos en tiempo real como transmisiones de video o de audio.

- Dispositivos que necesitan transferir archivos. Aunque el protocolo lo permite, no se recomienda, pues implica la transmisión de una gran cantidad de datos binarios que podrían incluir caracteres de cierre y alterar el formato de la representación de una rama.

- Aplicaciones que procesan datos a alta frecuencia. Pues el protocolo no permite el envío de datos en periodos menores a un segundo y se debe considerar posibles retrasos en la transmisión.

- Sistemas de control de rápida respuesta. Ya que en un sistema de esa naturaleza la frecuencia de actualización de datos necesita ser alta.

- Entornos donde la red es inestable. Pues los dispositivos necesitan de la comunicación en red para propagar la información.

Además, esta especificación no determina alguna forma de encriptación de los datos o medida de seguridad alguna, más allá de una autenticación básica para que un dispositivo se identifique como parte de un árbol. Por tanto, los datos

se transmiten en texto plano. Si se desea realizar una implementación con algún servidor en internet o en una red pública, es recomendable que se implemente una capa de encriptación o medidas de seguridad adicionales no definidas en esta especificación, según los requerimientos de la aplicación.

4. IMPLEMENTACIÓN DEL PROTOCOLO

4.1. Descripción de la implementación

Con la finalidad de probar el funcionamiento del protocolo diseñado, en este documento, se ha realizado un prototipo de dispositivo.

El prototipo consiste en un interruptor de un circuito de iluminación que es controlable por una interfaz web y por comandos de voz a través del asistente Alexa de Amazon.

Para demostrar su compatibilidad con otras aplicaciones existentes el protocolo se integra con el asistente Alexa, este recibe los comandos de voz y los traduce a peticiones HTTP hacia algún servidor en internet. Por tanto, el tronco está instalado en un servidor accesible desde internet.

Con el objeto de mantener el ejemplo de implementación lo más claro y simple posible no se han añadido medidas de seguridad adicionales para el acceso hacia internet. El ente que conozca el nombre del árbol y la llave puede controlar el dispositivo. No obstante, si se desea reproducir este ejemplo en una aplicación comercial, se recomienda agregar más medidas de seguridad.

El dispositivo prototipo, que funciona como la rama, se puede configurar al encenderlo. Con comandos, que se introducen con un pulsador, se puede solicitar al dispositivo que acceda al modo de configuración o que restablezca los valores de configuración a aquellos por defecto. En cualquiera de los casos se

enciende una red de área local inalámbrica a la que se puede conectar un usuario y acceder a una interfaz web de configuración.

El prototipo de rama es un módulo ESP-01S que controla un circuito de potencia, conectado en una red de área local inalámbrica a través de wifi con acceso a internet, en la banda de 2.4 GHz. El servidor del tronco está configurado en Python 2.7, alojado en internet. La raíz primaria, en este caso, la interfaz web de control, se encuentra en el mismo servidor del tronco y la rama secundaria, que en este caso es el asistente Alexa de Amazon, se encuentra en un servidor privado externos.

Cada sección de este ejemplo de implementación se detallará, para que cualquier persona con conocimientos básicos de redes de computadoras y programación en Python y C pueda replicar el mismo ejemplo o modificar características que considere.

4.2. Elementos utilizados

A continuación, se listan los elementos utilizados para la implementación del protocolo diseñado, tanto de hardware como de software.

- Elementos de hardware
  - Módulo ESP-01S
  - Circuito de potencia
  - Circuito de iluminación de 3 bombillas incandescentes de 100 W
  - Módulo Echo Dot de Amazon
  - Fuente de poder de 5 V y 3.3 V

- Elementos de software
    - Herramienta en línea PythonAnywhere
    - Entorno de desarrollo integrado de Arduino
    - Plug-in de ESP8266 para entorno de desarrollo integrado de Arduino
    - Herramienta de desarrollador de Amazon
    - Servicios Web de Amazon (AWS)

4.3. Procedimiento

En las subsecciones siguientes se detalla la construcción del prototipo de rama y su configuración, el tronco y las herramientas por integrar con el protocolo diseñado.

4.3.1. Construcción de un prototipo

El prototipo ha sido construido con el método artesanal de fabricación de circuitos impresos. A continuación, se detallan las fases de su fabricación.

4.3.1.1. Lista de componentes

El prototipo realizado consta de los siguientes componentes:

Tabla I. **Lista de componentes del prototipo**

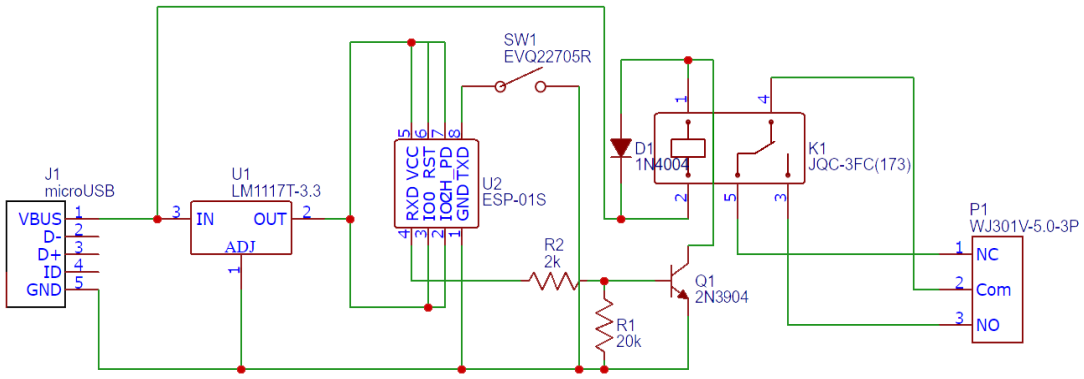
Código	**Nombre**	**Descripción**
D1	1N4004	Diodo de silicio
J1	Conector	Conector micro USB hembra
K1	JQC-3FC(173)	Relevador de 5 V a 120 V
P1	WJ301V-5.0-3P	Terminal de tres contactos
Q1	2N3904	Transistor NPN
R1	Resistencia	Resistencia de 20 kΩ para 0.25 W
R2	Resistencia	Resistencia de 2 kΩ para 0.25 W
SW1	EVQ22705R	Interruptor de pulso normalmente abierto
U1	LM1117T-3.3	Regulador de voltaje de 3.3 V
U2	ESP-01S	Módulo wifi con microprocesador ESP8266
	Pines	8 pines hembra para conexión de ESP-01S

Fuente: elaboración propia.

4.3.1.2. Esquemático

El esquemático es el diseño de las conexiones de los componentes que utiliza el circuito.

Figura 33. **Esquemático del prototipo**



Fuente: elaboración propia.

El circuito consta de tres partes principales:

- Alimentación de voltaje: esta sección está compuesta por un conector micro USB hembra que permite utilizar cualquier cargador de 5 V que provea al menos 300 mA como alimentación principal. Además, se añade un regulador de voltaje de 3,3 V, LM1117-3.3, que provee al circuito de alimentación para el módulo ESP-01S, pues este último admite alimentación entre 3 V y 3,7 V únicamente.

- Conmutación de potencia: el componente principal de esta etapa es un relevador que se activa con una señal de 5 V. Este componente hace la conmutación mecánica entre las terminales de salida. Sin embargo, el módulo ESP-01S únicamente emite señales de 3,3 V. Por lo tanto, se ha utilizado un transistor 2N3904, polarizado como conmutador, con R1 y R2, que permite convertir la señal del módulo ESP-01S a una señal de 5 V. Como elemento final de esta sección se agrega un diodo de silicio en

paralelo, con polarización inversa, a la bobina del relevador, esto se utiliza como protección del circuito ante la corriente residual de la bobina al desconectarse.

- Control y configuración: en esta sección está el módulo ESP-01S, el centro del prototipo. Este módulo es el que se encarga de conectarse a la red inalámbrica por medio de wifi, recibir las instrucciones del tronco y enviar las señales a la etapa de potencia. Otra función del módulo es la de permitir al usuario ingresar la configuración de la red inalámbrica y de la rama, esta función es activada al presionar el interruptor de pulso.

4.3.1.3. Valores de operación

El circuito se ha diseñado para operar con un circuito de iluminación de tres bombillas incandescentes de 100 W en paralelo. Por tanto, la parte de conmutación de potencia debe soportar 110 V y 2,73 A. El relevador seleccionado soporta hasta 10 A y 125 V.

La bobina del relevador consume 0,36 W a 5 V, por tanto, utiliza una corriente de 72 mA, mientras el módulo ESP-01S consume una corriente máxima de 170 mA, consecuentemente, la fuente de alimentación de bajo voltaje debe proporcionar al menos 242 mA a 5 V. Se ha seleccionado un cargador genérico de 5 V y 500 mA para su operación.

El prototipo se diseña para operar a temperatura ambiente, de 25 °C, y ya que el prototipo puede ser encapsulado dentro de un contenedor plástico, se espera que la temperatura de las pistas de cobre no aumente más de 10 °C sobre la temperatura de operación.

Tabla II. **Valores de operación del prototipo**

Parámetro	**Valor**
Voltaje de alimentación	5 V
Corriente de alimentación	242 mA
Voltaje de conmutación de potencia	110 V
Corriente de conmutación de potencia	2,73 A
Temperatura de operación	25 °C
Aumento de temperatura máximo	10 °C

Fuente: elaboración propia con base en las hojas de datos de los componentes.

4.3.1.4. Diseño del circuito impreso

Con base en el esquemático del circuito y el estándar de circuitos impresos, IPC-2221A,[20] de se ha diseñado la placa de circuito impreso.

Según el estándar mencionado, el ancho mínimo de las pistas conductoras es proporcional a la corriente máxima que transportan según la ecuación:

$$I = k\Delta T^{0.44} A^{0.725}$$

Donde:

- I es la corriente en amperios.

[20] IPC-2221 TASK GROUP. *IPC-2221A, Generic Standard on Printed Board Design.* http://www.ipc.org/TOC/IPC-2221A.pdf. Consulta: agosto de 2019.

- k es una constante que toma el valor de 0,048 para pistas en caras exteriores.

- ∆T es el cambio de temperatura permitido en grados centígrados.

- A es el área transversal de la pista dada en milésimas de pulgada cuadradas. Se puede descomponer en w, el ancho de la pista, y h, la altura de la pista, dadas en milésimas de pulgada (mil).

Despejando la ecuación para el valor de ancho de la pista (w), se obtiene:

$$w = \frac{\sqrt[0.725]{\frac{I}{k\Delta T^{0.44}}}}{h}$$

Según el estándar IPC-2221A[21] la altura de una pista de cobre (h) de 1 oz de cobre por pie cuadrado es de 35 µm, aproximadamente 1,38 mil. Ya que el prototipo se ha construido de forma artesanal, el procesado puede desgastar hasta un 50 % de esa altura, por tanto, se utilizará h como 0,69 mil.

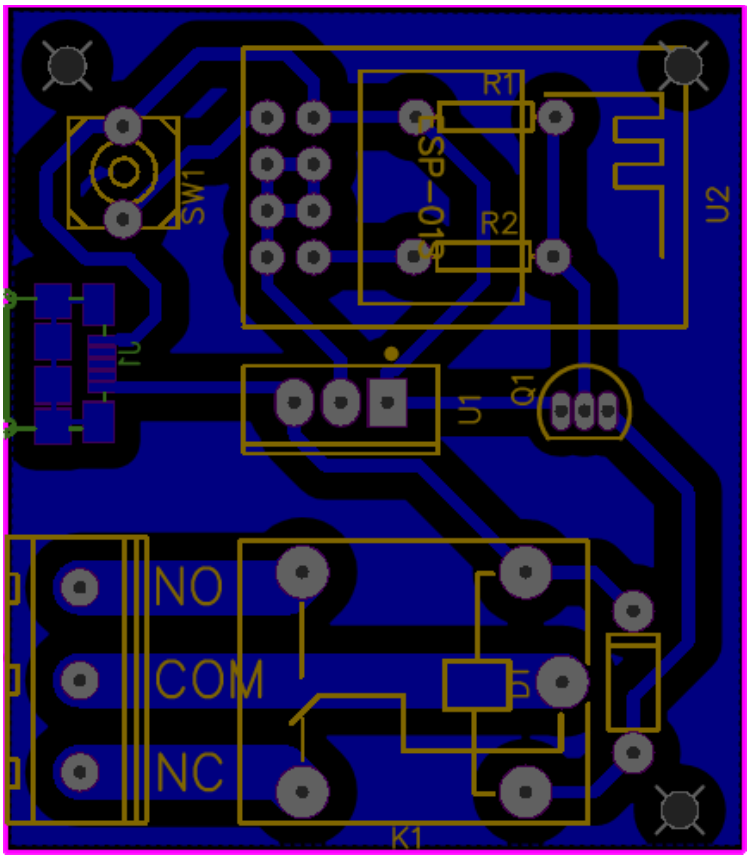
Sustituyendo valores en la ecuación se consigue el ancho de pista para la etapa de conmutación de potencia, 94,37 mil, aproximadamente 2,40 mm, este valor se redondeará a 3 mm para que una variación superior de la corriente no dañe el circuito impreso. Con el mismo procedimiento se consigue el ancho de la pista de la etapa de bajo voltaje, 3.34 mil, aproximadamente 0.08 mm. Sin embargo, con el método artesanal de fabricación, las pistas delgadas son susceptibles a defectos o fallas, por lo que se aumentará el ancho de la pista a 0.7 mm, un valor comúnmente usado en la práctica.

[21] IPC-2221 TASK GROUP. *IPC-2221A, Generic Standard on Printed Board Design.* http://www.ipc.org/TOC/IPC-2221A.pdf. Consulta: agosto de 2019..

Adicional a los requerimientos anteriores, es preferible que las pistas nunca formen ángulos rectos al cambiar de dirección, por tanto, las pistas conductoras se han diseñado para giren en ángulos de 45° durante su trayectoria.

El diseño se muestra en la siguiente figura:

Figura 34. **Diseño del circuito impreso del prototipo a color**



Fuente: elaboración propia empleando herramienta en línea EasyEDA.

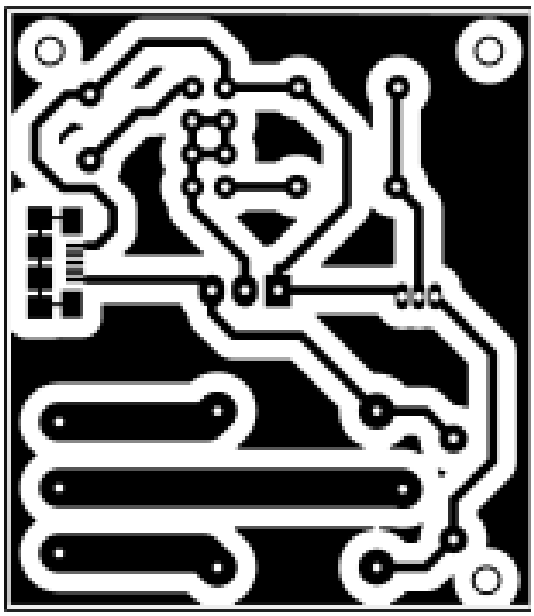
El área en azul representa el cobre en la cara posterior de la placa de circuito impreso, las líneas en amarillo representan la silueta de los componentes a del circuito montadas sobre la cara anterior, las líneas en verde muestran el

componente que se suelda en la cara posterior. Las áreas de soldadura de montaje de perforación se muestran en gris claro, mientras los agujeros aparecen en gris oscuro.

4.3.1.5. Fabricación de la placa de circuito

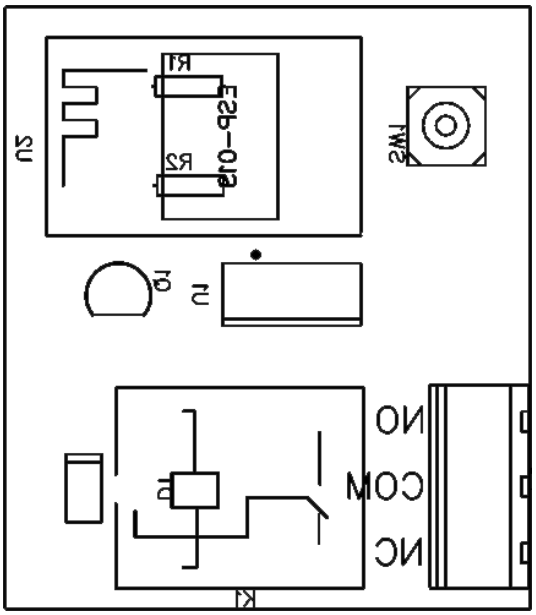
Para la fabricación de la placa del circuito, se imprimió el diseño para ambas caras de la placa, con dimensiones de 4,66 cm de alto y 4,06 cm de ancho.

Figura 35. **Diseño de cara posterior de la placa de circuito impreso**



Fuente: elaboración propia empleando herramienta en línea EasyEDA.

Figura 36. **Diseño de cara anterior de la placa de circuito impreso**



Fuente: elaboración propia empleando herramienta en línea EasyEDA.

El método de fabricación artesanal utilizado se detalla a continuación:

- Se imprime el diseño de ambas caras sobre papel termotransferible usando una impresora láser.

- Se pule la superficie de la placa algún material abrasivo para eliminar el componente antioxidante y limpiarla.

- Se coloca la impresión sobre la placa y se aplica calor, alrededor de 180 °C, al mismo tiempo que presión para que la impresión se transfiera a la placa. Se utilizó una plancha doméstica para este procedimiento.

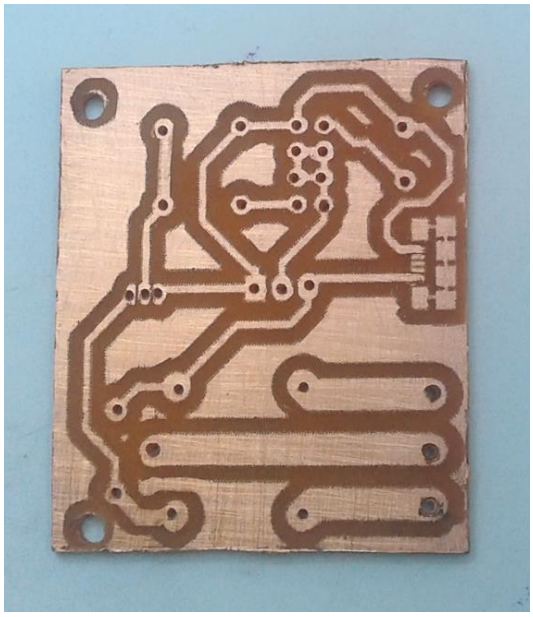
- Se sumerge la placa en agua y se retira el papel despacio para evitar dañar la impresión.

- Se sumerge la placa en una solución de cloruro férrico para que se disuelva el cobre que no fue cubierto por la impresión.

- Se perfora la placa con brocas adecuadas según el diámetro de los conectores de los componentes. Se utilizaron brocas de 1/32” y 1/16”.

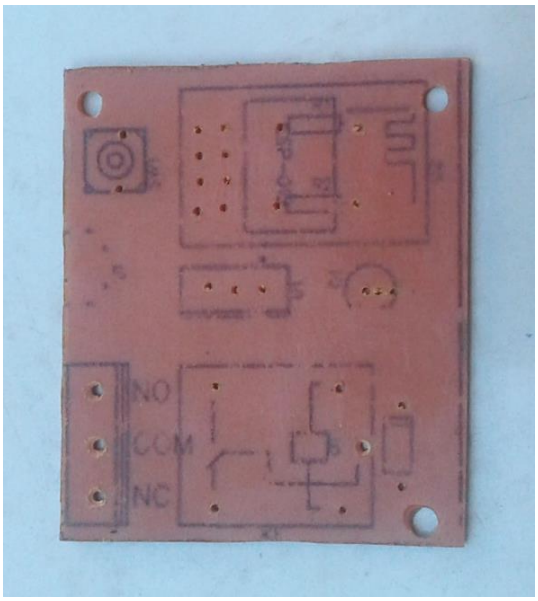
  El resultado de este método es el siguiente:

Figura 37. **Cara posterior de placa de circuito impreso sin componentes**



Fuente: elaboración propia.

Figura 38. **Cara anterior de placa de circuito impreso sin componentes**

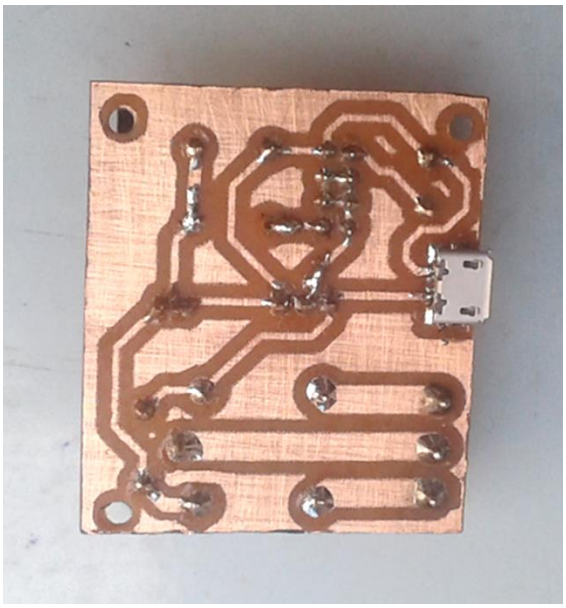


Fuente: elaboración propia.

4.3.1.6. Montaje de componentes

Todos los componentes, exceptuando J1, son dispositivos de paso por agujero (THD por sus siglas en inglés), mientras J1 es un dispositivo de montaje superficial (SMD por sus siglas en inglés). Los componentes THD se colocaron sobre la cara anterior y sus terminales se soldaron en la cara posterior de la placa y el componente SMD se montó y soldó sobre la cara posterior. Además, el componente U2, fue conectado sobre pines hembra que fueron soldados en su lugar.

Figura 39. **Cara posterior de placa de circuito impreso con componentes soldados**



Fuente: elaboración propia.

Figura 40. **Cara anterior de placa de circuito impreso con componentes soldados**

Fuente: elaboración propia.

4.3.2. Configuración del dispositivo central y de control de usuario

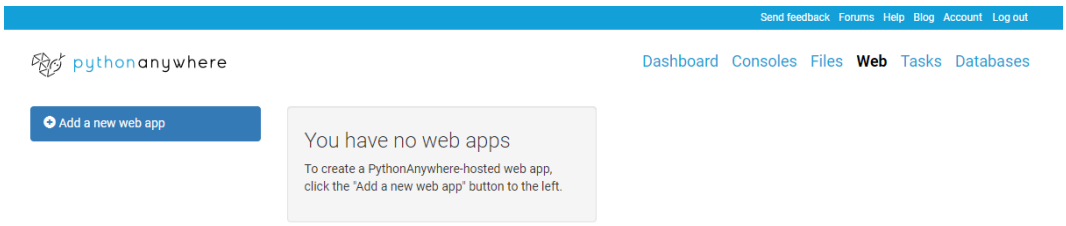
El dispositivo central, llamado tronco, y la raíz primaria fueron instalados en un servidor de HTTP, programado en Python 2.7, alojado en internet en el servidor PythonAnywhere.

4.3.2.1. Instalación de un servidor web

Se escogió la herramienta en línea PythonAnywhere, debido a que es un servidor gratuito, programable en Python y tiene suficiente capacidad para manejar un entorno de pruebas. Esta herramienta se encuentra disponible en https://www.pythonanywhere.com y para utilizarla es necesario crear una cuenta gratuita con una dirección de correo electrónico.

Al acceder a la cuenta creada es necesario añadir una nueva aplicación desde la sección web:

Figura 41. **Interfaz para agregar nueva aplicación en PythonAnywhere**



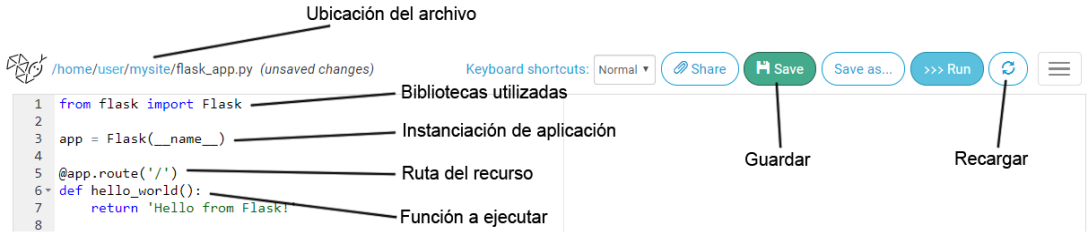
Fuente: elaboración propia, basado en sitio web de PythonAnywhere.

PythonAnywhere permite crear aplicaciones web eligiendo entre distintos marcos de trabajo (*frameworks*). Para la aplicación creada se ha elegido el marco de trabajo Flask programado en Python 2.7, este es un marco para la configuración de servidores de HTTP.

Cuando se ha elegido el marco de trabajo, la versión de Python y la ruta de los archivos se accede a la configuración de la aplicación. A partir de ahí se puede acceder desde la sección *Code* al directorio del código fuente.

El archivo flask_app.py contendrá el código principal de la aplicación. Al abrirlo mostrará la consola de Python:

Figura 42. **Consola de Python en PythonAnywhere**



Fuente: elaboración propia, basado en sitio web de PythonAnywhere.

En esta consola se puede programar todo el código fuente del servidor que manejará el tronco y la raíz primaria del árbol. Cada vez que se quiere probar el resultado se debe guardar el código y recargar el servidor para que tomen efecto los cambios.

4.3.2.2. Diagrama de flujo del servidor

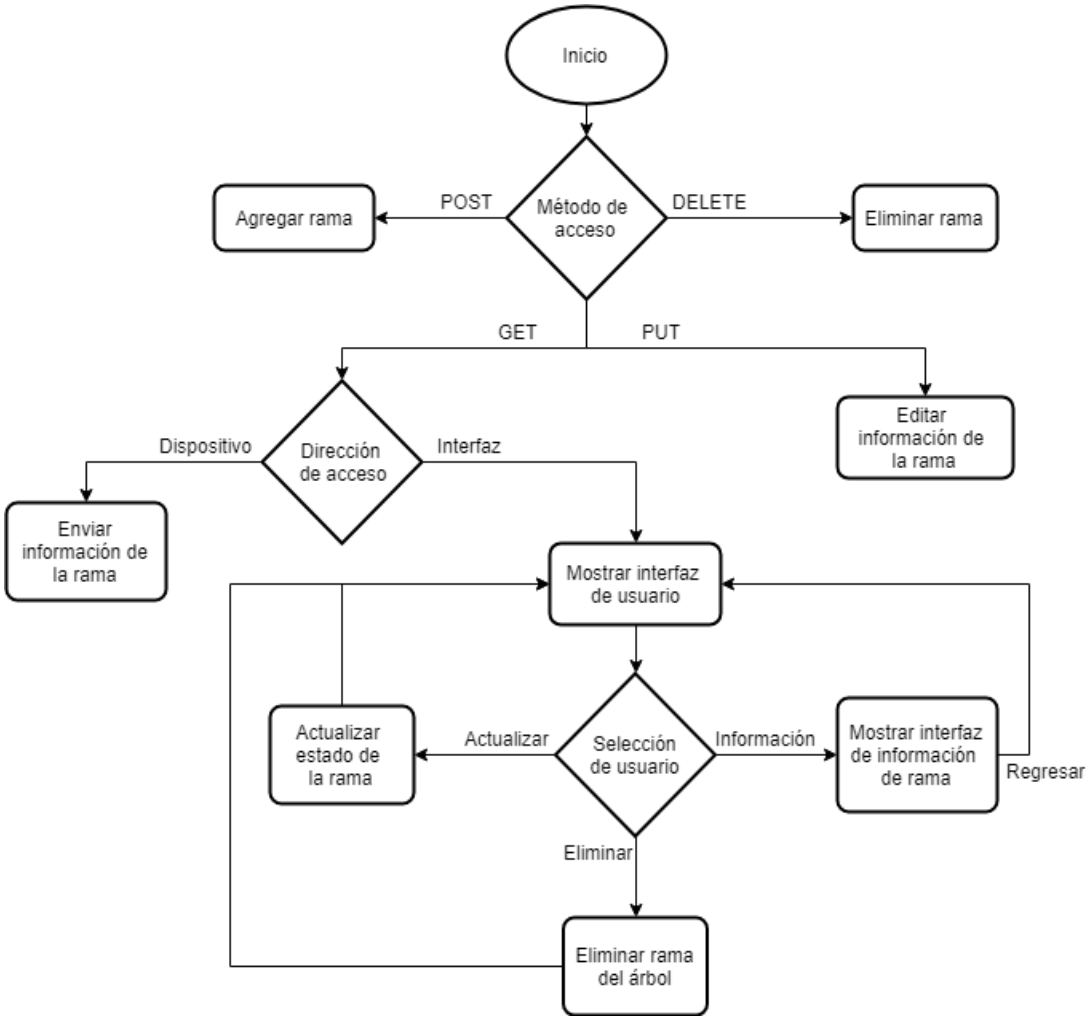
HTTP, es un protocolo sin estado y asíncrono, es decir que puede recibir peticiones en cualquier momento y no guarda información de peticiones anteriores. Por tanto, el método de acceso será el que determine la función a ejecutar por el servidor:

- Si se usa el método POST se ejecutará una función que agregue una rama.
- Si es DELETE, se ejecutará una función que elimine una rama.
- Si es PUT, se ejecutará una función que edite la información de la rama.
- Si es GET, podrá enviar información de una rama o mostrar la interfaz de usuario de la raíz primaria.

La interfaz de usuario contendrá, para cada rama, una opción para actualizar su estado, una para eliminarla del árbol y una para ver su información.

El siguiente diagrama de flujo resume la configuración del servidor:

Figura 43. **Diagrama de flujo de la configuración del servidor principal**



Fuente: elaboración propia.

4.3.2.3. Configuración del servidor web

Los nombres de objetos, funciones y variables se han asignado en inglés para mantener una homogeneidad de lengua en la programación, ya que las funciones propias del lenguaje de programación se encuentran en idioma inglés.

A continuación, se detalla el código fuente del servidor web. Este se puede encontrar completo en el apéndice 1.

La primera parte del código fuente es la importación de bibliotecas por utilizar:

Figura 44. **Código fuente del servidor, segmento 1**

```
1  #Bibliotecas utilizadas
2  from flask import Flask, request, render_template, jsonify, Response
3  import json
```

Fuente: elaboración propia.

La biblioteca *Flask* es la que contiene las herramientas propias del marco de trabajo, *Response* y *request*, contienen herramientas para personalizar respuestas y peticiones, respectivamente, *render_template* se utiliza para mostrar páginas web escritas con HTML y *json* y *jsonify* manejan y convierten objetos JSON.

Posteriormente, se agregan los objetos y variables que son utilizados por varias funciones del código, conocidas como objetos y variables globales:

Figura 45. **Código fuente del servidor, segmento 2**

```
5  #Objetos y variables globales
6  app = Flask(__name__)
7  tree = "tree"
8  password = "key"
9  devices = ""
```

Fuente: elaboración propia.

App es el objeto que inicializa la aplicación, *tree* y *password* son el nombre del árbol y su llave, respectivamente y *devices* es la variable que almacenará de forma temporal la lista de ramas agregadas.

Luego se necesita que al iniciar la aplicación se abra un archivo donde se almacenaron de forma permanente las ramas agregadas al árbol y se copie la información a la variable *devices*. Esta última servirá para tener acceso a la información de las ramas del árbol sin tener que realizar operaciones de lectura de archivos, de esa forma se reduce el tiempo de respuesta.

Figura 46. **Código fuente del servidor, segmento 3**

```
11  #Obtiene ramas de un archivo de texto
12  f = open("mysite/devices.txt","rb")
13  devices = json.loads(f.read())
14  f.close()
```

Fuente: elaboración propia.

La instrucción *json.loads*() toma una cadena de texto del archivo *devices.txt* y la convierte a un objeto JSON.

Figura 47. **Código fuente del servidor, segmento 4**

```
16  #Maneja el metodo POST para agregar ramas al arbol
17  @app.route('/devices', methods=['POST'])
18  def addDevices():
19      auth = request.authorization
20      if auth and auth.username == tree and auth.password == password:
21          objects = request.json
22          created = False
23          for key in objects:
24              if not devices.has_key(key):
25                  devices[key] = objects[key]
26                  created = True
27          f = open("mysite/devices.txt","wb")
28          f.write(json.dumps(devices))
29          f.close()
30          if created:
31              return Response(
32              '<h1>Created</h1> All or some of the devices were created', 201)
33          else:
34              return Response(
35              '<h1>Accepted</h1> The devices were not created', 202)
36      else:
37          return Response(
38          '<h1>Unauthorized</h1>', 401,
39          {'WWW-Authenticate': 'Basic realm="Devices"'})
```

Fuente: elaboración propia.

La función *addDevices*() ser la encargada de manejar una petición que utiliza el método de acceso POST con la URN */devices*. En primer lugar, debe comprobar que las credenciales de acceso, el árbol y la llave, son las correctas, en caso contrario envía una respuesta con código 401, indicando que las credenciales son incorrectas. Luego, entre el cuerpo de la petición obtiene las ramas en formato JSON y verifica que no pertenezcan de antemano al árbol. Si no se han agregado antes las añade. Guarda los nuevos datos en el archivo permanente *devices.txt* y envía una respuesta con el código 201, indicando que alguna rama se agregó, o con el código 202, indicando que las ramas ya habían sido agregadas antes y no se añadieron de nuevo.

Figura 48. **Código fuente del servidor, segmento 5**

```
41  #Maneja el metodo PUT para modificar informacion de las ramas
42  @app.route('/devices/<subset>', methods=['PUT'])
43  @app.route('/devices', methods=['PUT'])
44  def updateDevices(subset=''):
45      auth = request.authorization
46      updated = False
47      if auth and auth.username == tree and auth.password == password:
48          objects = request.json
49          for key in objects:
50              if devices.has_key(key):
51                  if subset == '':
52                      devices[key] = objects[key]
53                  else:
54                      devices[key][str(subset)] = objects[key]
55                  updated = True
56          f = open("mysite/devices.txt","wb")
57          f.write(json.dumps(devices))
58          f.close()
59          if updated:
60              return Response(
61              '<h1>Created</h1> All or some of the devices were updated', 201)
62          else:
63              return Response(
64              '<h1>Accepted</h1> The devices were not updated', 202)
65      else:
66          return Response(
67          '<h1>Unauthorized</h1>', 401,
68          {'WWW-Authenticate': 'Basic realm="Devices"'})
```

Fuente: elaboración propia.

La función *updateDevices*() maneja las peticiones que usan el método de acceso PUT cuando se utiliza el URN */devices*, */devices/meta* o */devices/data* y sigue el mismo procedimiento de autenticación de la función anterior. Verifica que la rama exista en el árbol y modifica los datos que se reciben en el cuerpo de la petición, los mismos son guardados en el archivo permanente devices.txt. Por último, envía una respuesta con código 201, indicando que alguna rama fue editada, o con código 202, indicando que la información no se editó porque la rama no se encontró.

Figura 49. **Código fuente del servidor, segmento 6**

```
70  #Maneja el metodo DELETE para eliminar ramas del arbol
71  @app.route('/devices/<id>', methods=['DELETE'])
72  def deleteDevices(id):
73      auth = request.authorization
74      if auth and auth.username == tree and auth.password == password:
75          deleted = False
76          tmp = devices.copy()
77          for key in tmp:
78              if key.find(id)==0:
79                  deleted = True
80                  del(devices[key])
81              else:
82                  continue
83          f = open("mysite/devices.txt","wb")
84          f.write(json.dumps(devices))
85          f.close()
86          if deleted:
87              return Response(
88              '<h1>Created</h1> All or some of the devices were deleted', 201)
89          else:
90              return Response(
91              '<h1>Accepted</h1> The devices did not exist', 202)
92      else:
93          return Response(
94          '<h1>Unauthorized</h1>', 401,
95          {'WWW-Authenticate': 'Basic realm="Devices"'})
```

Fuente: elaboración propia.

La función *deleteDevices*() maneja las peticiones que se reciben con el método de acceso DELETE y el URN /*devices*/<id>, donde <id> es el identificador de la rama a eliminar. Evalúa las credenciales al igual que las funciones anteriores. Luego busca todas las ramas que coincidan con el identificador o sean jerárquicamente inferiores y las elimina del árbol. Guarda las ramas restantes en el archivo permanente *devices.txt* y envía una respuesta con código 201, si alguna rama fue eliminada, o con código 202, si ninguna rama se pudo eliminar porque no se encontró.

Figura 50. **Código fuente del servidor, segmento 7**

```
 97  #Maneja el metodo GET para enviar informacion de las ramas
 98  @app.route('/devices/<id>/<subset>', methods=['GET'])
 99  @app.route('/devices/<id>', methods=['GET'])
100  def getDevices(id, subset=''):
101      auth = request.authorization
102      if auth and auth.username == tree and auth.password == password:
103          body = {}
104          for key in devices:
105              if key.find(id) == 0:
106                  if subset == '':
107                      body[key] = devices[key]
108                  else:
109                      body[key] = devices[key][subset]
110          return jsonify(body)
111      else:
112          return Response(
113          '<h1>Unauthorized</h1>', 401,
114          {'WWW-Authenticate': 'Basic realm="Devices"'})
```

Fuente: elaboración propia.

La función *getDevices*() responde a las peticiones que utilizan el método de acceso GET y el URN */devices/*<id>, */devices/*<id>*/meta* y */devices/*<id>*/data*, siendo <id> el identificador de la rama de la que se busca información. Maneja la autenticación de la misma manera que las funciones anteriores. Busca entre las ramas del árbol aquella que coincida con el identificador y las que sean jerárquicamente inferiores a ella. Los datos son enviados en formato de objeto JSON en la respuesta. De no encontrar datos envía un objeto JSON vacío.

El servidor web también contiene la raíz primaria, que es representada por una interfaz gráfica que le permite el control al usuario de cada dispositivo del árbol. Esta interfaz contiene tres botones por cada dispositivo, uno para actualizar su estado, uno para eliminar la rama del árbol y otro para ver su información completa.

Figura 51. **Ejemplo de interfaz de usuario para control del sistema**

Fuente: elaboración propia.

La página web se muestra según la codificación HTML siguiente en el archivo *interface.html*:

Figura 52. **Código HTML de la interfaz de control de usuario**

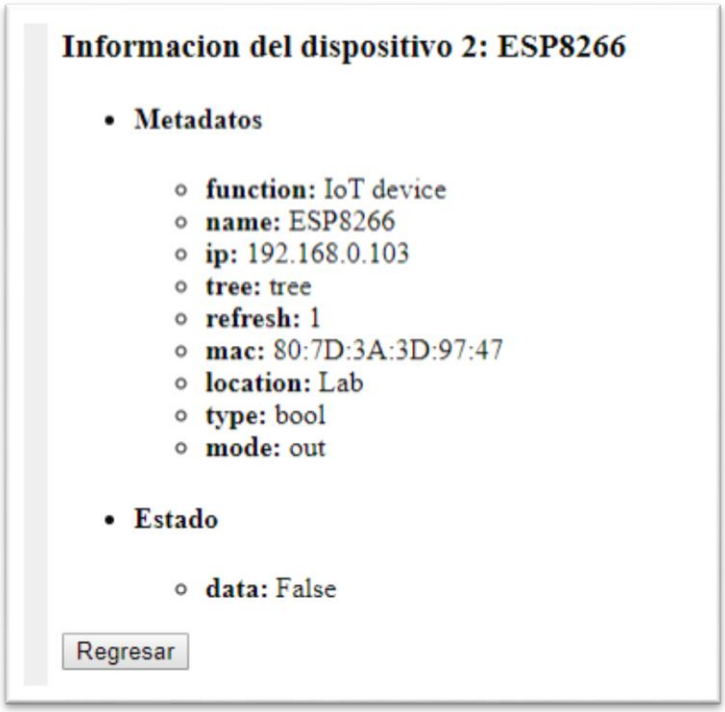
```
<!DOCTYPE html>
<html>
<head>
    <title>Interfaz de usuario</title>
</head>
<body>
    <h1>Interfaz de usuario</h1>
    {%for id in objects%}
        <form action = "http://user.pythonanywhere.com/interface" method = "POST">
            <p><h3><label>{{id}}: {{objects[id]['meta']['name']}}</label></h3></p>
            {%if objects[id]['meta']['type'] == 'bool'%}
                <p><input type="hidden" name={{id}}></p>
                <p><input type="submit" value ="Conmutar">
            {%else%}
                <p><input type="text" name={{id}}></p>
                <p><input type="submit" value ="Actualizar">
            {%endif%}
        </form>
        <form action = "http://user.pythonanywhere.com/interface" method = "POST">
            <input type="hidden" name={{id}}>
            <input type="hidden" name="DELETE">
            <input type="submit" value ="Eliminar"></p>
            <p><a href={{"http://user.pythonanywhere.com/interface/info/"+id}}>
                <input type="button" value ="Informacion"></a></p>
        </form>
    {%endfor%}
</body>
</html>
```

Fuente: elaboración propia.

El ciclo iterativo *for* recorre un diccionario que contiene las ramas del árbol y coloca un botón llamado conmutar, si el tipo de la rama es binario, o uno llamado Actualizar en conjunto con un campo de texto, si el tipo de la rama es otro. Ambos botones envían un formulario al presionarse. Además, se agrega otro formulario con un botón para eliminar la rama y un botón con un enlace a una interfaz de información.

La interfaz de información del dispositivo muestra el identificador de la rama en el título de la página. Posteriormente, una lista con los metadatos y otra con el estado de la rama. Por último, un botón para regresar a la interfaz de control.

Figura 53. **Ejemplo de interfaz de información de un dispositivo**



Fuente: elaboración propia.

Su codificación es la siguiente y se encuentra en el archivo *info.html*:

Figura 54. **Código HTML de la interfaz de información de un dispositivo**

```
<!DOCTYPE html>
<html>
<head>
    <title>Informacion del dispositivo {{id}}</title>
</head>

<body>
    <h3>Informacion del dispositivo {{id}}: {{device['meta']['name']}}</h3>
    <ul>
        <li><h4>Metadatos</h4></li>
        <ul>
            {%for field in device['meta']%}
                <li><b>{{field}}: </b>{{device['meta'][field]}}</li>
            {%endfor%}
        </ul>
        <li><h4>Estado</h4></li>
        <ul>
            <li><b>data: </b>{{device['data']}}</li>
        </ul>
    </ul>

    <a href="http://user.pythonanywhere.com/interface"><input type="button" value="Regresar"></a>
</body>
</html>
```

Fuente: elaboración propia.

La codificación contiene un ciclo iterativo *for* que recorre un diccionario con los metadatos del dispositivo y muestra en una lista cada uno de ellos. Por último, muestra el estado de la rama y agrega un botón con un enlace hacia la interfaz de control.

En el mismo archivo donde se encuentra el código fuente del tronco se ha agregado el código fuente del servidor que maneja la interfaz de usuario y es el siguiente:

Figura 55. **Código fuente del servidor, segmento 8**

```
116 #Maneja la interfaz de usuario
117 @app.route('/interface', methods=['GET','POST'])
118 def showInterface():
119     auth = request.authorization
120     if auth and auth.username == tree and auth.password == password:
121         if request.method == 'POST':
122             data = request.form
123             key = data.keys()[0]
124             if len(data)>1:
125                 if data.keys()[1] == "DELETE" and devices.has_key(key):
126                     del(devices[key])
127             else:
128                 if devices[key]['meta']['type']=='bool':
129                     state = devices[key]['data']
130                     devices[key]['data'] = not(state)
131                 else:
132                     devices[key]['data'] = data.values()[0]
133             f = open("mysite/devices.txt","wb")
134             f.write(json.dumps(devices))
135             f.close()
136             return render_template('interface.html',objects=devices)
137         else:
138             return render_template('interface.html',objects=devices)
139     else:
140         return Response(
141         '<h1>Unauthorized</h1>', 401,
142         {'WWW-Authenticate': 'Basic realm="Devices"'})
```

Fuente: elaboración propia.

La función *showInterface*() muestra la interfaz de usuario cuando la petición utiliza el URN */interface* y los métodos de acceso GET o POST. Cuando el usuario accede a la página web desde un navegador, este por defecto envía una petición con el método de acceso GET, esto significa que aún no se ha enviado ningún dato en el formulario. Cuando el navegador utiliza el método POST significa que se ha utilizado el botón de envío de formulario en la página web.

Por lo tanto, cuando la petición contiene el método de acceso POST, el servidor debe recolectar los datos del formulario enviados y modificar el estado de la rama o eliminarla según el usuario elija. La función *render_template*()

muestra la página web que se detalla en el archivo interface.html enviándole un diccionario que contiene las ramas del árbol.

4.3.3. Configuración de un dispositivo actuador

La rama que controla el circuito de iluminación utilizado en este ejemplo debe programarse sobre un dispositivo que tenga capacidad para conectarse a una red LAN. Para esto se ha seleccionado el módulo ESP-01S que utiliza como base el microcontrolador ESP8266. Este es un microcontrolador, creado por la empresa Espressif, que tiene conectividad inalámbrica por medio de wifi en la banda de 2.4 GHz, puede funcionar como servidor o cliente de wifi y tiene pines de entrada y salida digitales.

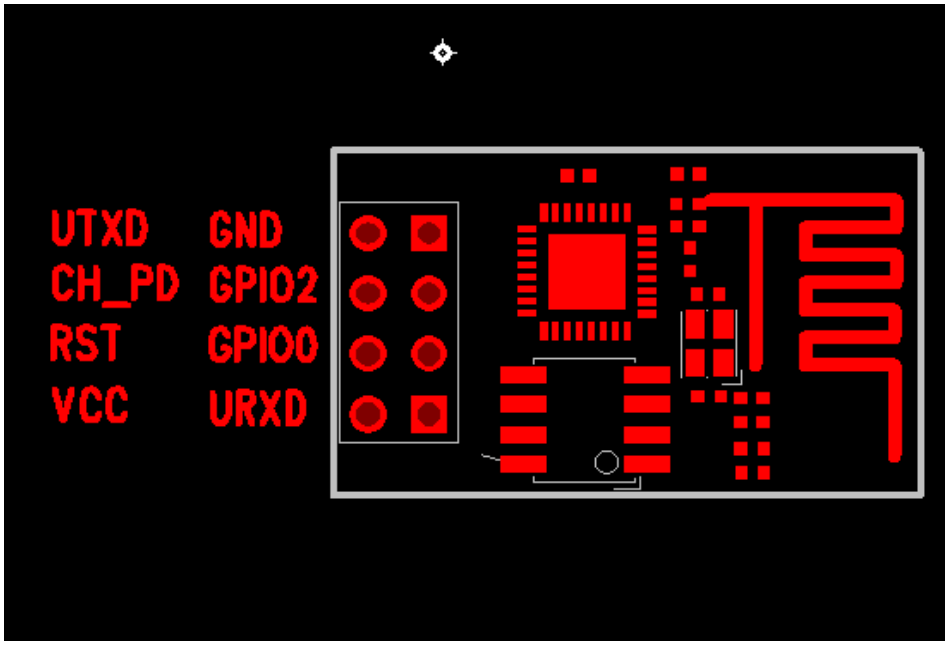
Figura 56. **Módulo ESP-01S**

Fuente: PROMETEC. ESP-8266 01 WIFI. store.prometec.net. Consulta: 6 de septiembre de 2019.

El módulo ESP-01S contiene dos pines de entrada y salida de propósito general (GPIO0 y GPIO2), dos de comunicación UART (UTXD y URXD), que

pueden utilizarse como entradas y salidas de propósito general, un pin de alimentación de 3.3 V (VCC), uno de conexión a referencia (GND), un pin de selección (CH_PD) y uno de reinicio externo (RST).

Figura 57. **Mapa de pines del módulo ESP-01S**



Fuente: PROMETEC. Arduino y Wifi ESP8266. www.prometec.net. Consulta: 5 de septiembre de 2019.

La programación se ha realizado utilizando el entorno de desarrollo integrado de Arduino y se detalla en las subsecciones siguientes.

4.3.3.1. Configuración del entorno de desarrollo

El microcontrolador ESP8266 se puede programar mediante comandos AT, código de Arduino, que es una variante de C, y con intérpretes de Python, Javascript o Basic. Se ha seleccionado el código de Arduino para su

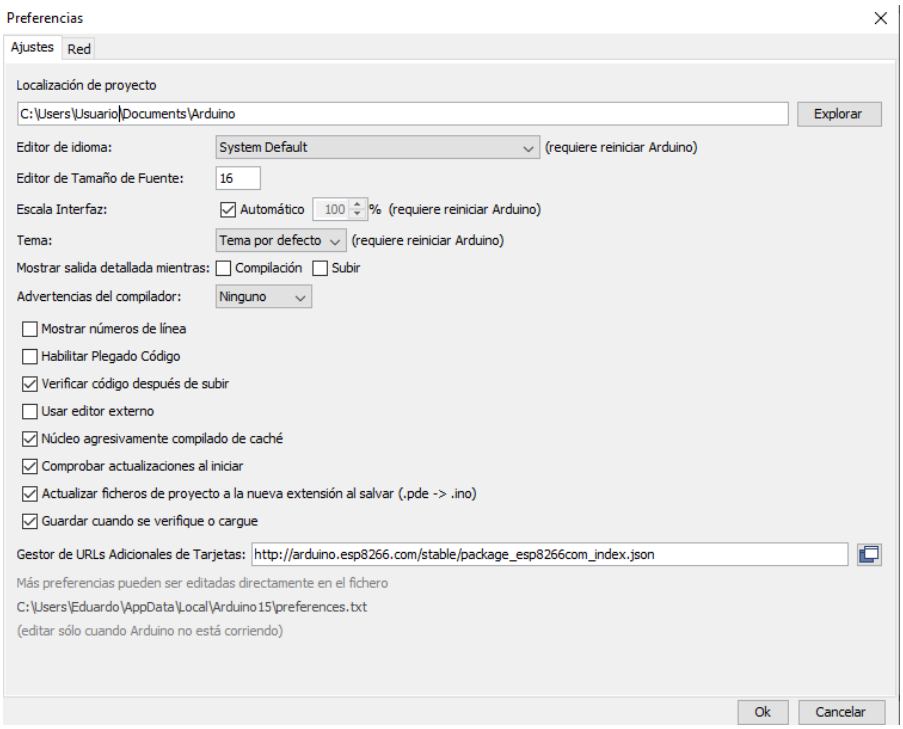
programación por ser el código para el que existe mayor documentación y el más sencillo de interpretar por una persona que lea este documento.

Antes de comenzar a programar el código fuente de la rama, es necesario configurar el entorno de desarrollo integrado de Arduino, pues originalmente este no ha sido diseñado para la configuración del microcontrolador ESP8266. Para esto es necesario contar con la versión 1.6.4 o superior del IDE, este se puede descargar desde la página oficial de Arduino https://www.arduino.cc.

Es necesario instalar un complemento en el IDE de Arduino para que reconozca los módulos basados en el microcontrolador ESP8266. Esto se realiza desde la configuración de preferencias, a la que se accede desde el menú de archivo. En el cuadro de texto que tiene por etiqueta Gestor de URLs adicionales de tarjetas, se debe agregar la dirección http://arduino.esp8266.com/stable/package_esp8266com_index.json.

Si ya se encuentra alguna otra dirección en el cuadro se puede agregar al final una coma seguida de la nueva dirección. Por último, el botón Ok guardará los cambios realizados.

Figura 58. **Ventana de configuración de preferencias del IDE de Arduino**



Fuente: elaboración propia, basado en entorno de desarrollo integrado de Arduino.

Posteriormente, en el menú de herramientas, en la selección de la placa, se selecciona Gestor de tarjetas e ingresar en el buscador ESP8266. Es necesario instalar la tarjeta que tiene por título esp8266 *by* ESP8266 *Community*.

Figura 59. **Gestor de tarjetas del IDE de Arduino**

Fuente: elaboración propia, basado en entorno de desarrollo integrado de Arduino.

Después de la instalación de la tarjeta ESP8266, se pueden configurar los demás parámetros. Desde el menú de herramientas, en la sección Placa, se selecciona *Generic* ESP8266 *Module* y en la sección *Upload Speed*, 115200.

Por otra parte, el dispositivo, deberá guardar datos, la configuración de su red de área local y sus metadatos, que podrán ser modificados por el usuario y deberán permanecer almacenados aun cuando se pierda la alimentación del circuito. Por tanto, adicional a la instalación del complemento de tarjetas, será necesario agregar otro complemento para la administración de un sistema de archivos dentro de la memoria *flash* de la tarjeta.

Esta herramienta se puede descargar desde la dirección https://github.com/esp8266/arduino-esp8266fs-

plugin/releases/download/0.3.0/ESP8266FS-0.3.0.zip y se descomprime en la carpeta *tools*, dentro de la ruta de instalación del IDE de Arduino.

Al reiniciar el IDE de Arduino se habrá agregado, en el menú herramientas, la opción ESP8266 *Sketch Data Upload*, y será esta la que permita subir archivos a la memoria *flash* del módulo ESP-01S.

Habiendo realizado las configuraciones mencionadas en esta sección, se puede iniciar a programar el dispositivo.

4.3.3.2. Diagrama de flujo del dispositivo actuador

El dispositivo deberá contener un pin de entrada que se conectará a un pulsador y un pin de salida que controlará el circuito de iluminación.

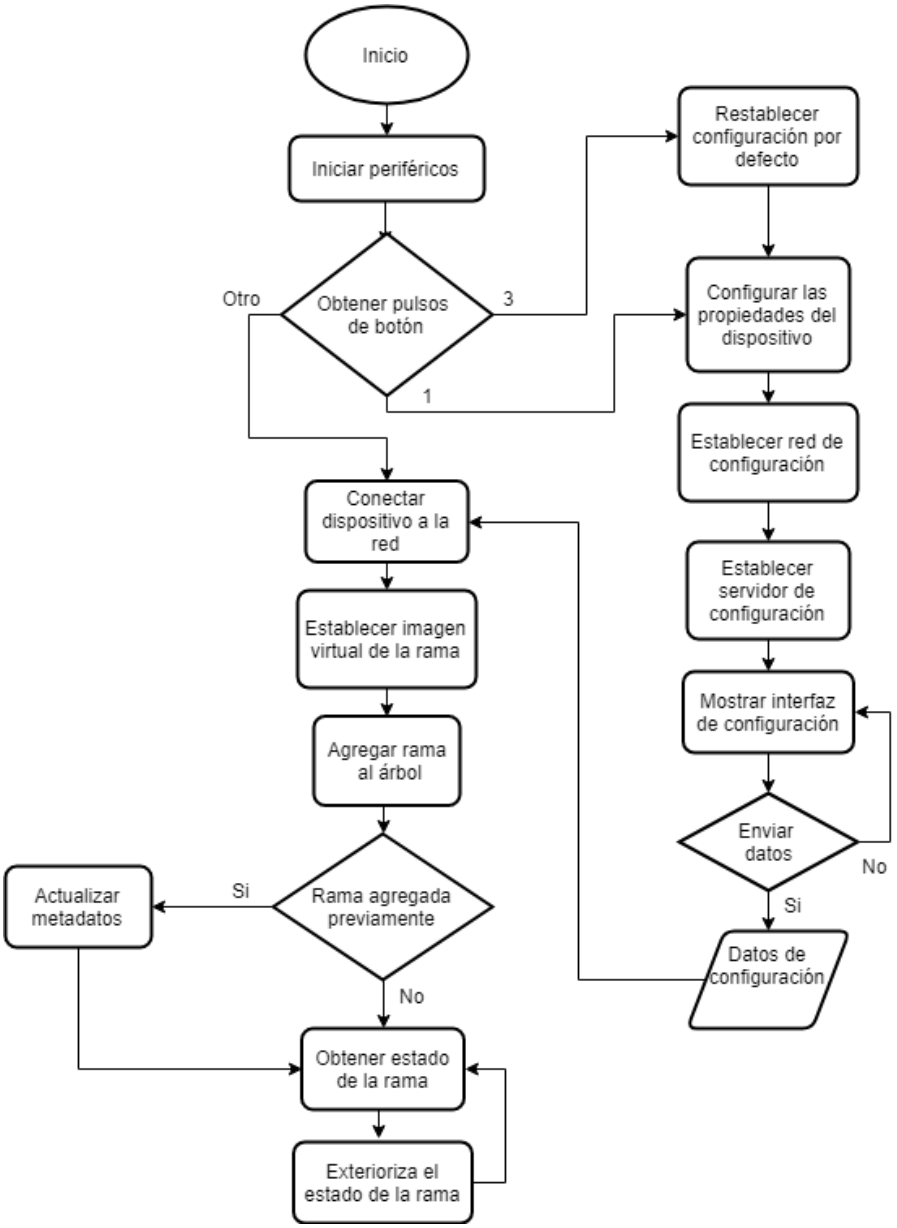
Al iniciar el dispositivo, este debe permitir un lapso para que el usuario pueda presionar el pulsador si desea configurar los datos de red y metadatos o desea reiniciar los parámetros a los valores por defecto.

Si el usuario presiona el botón, el dispositivo debe levantar un punto de acceso inalámbrico y un servidor web con una página para que el usuario pueda ingresar los parámetros que desea.

Después de haber pasado el tiempo de selección de configuración o después de haber configurado el dispositivo, este debe conectarse a la red inalámbrica de área local que ha configurado el usuario. Luego inicia el dispositivo y la rama y agrega la rama al árbol. Una vez agregada o actualizada exitosamente, el dispositivo consultará periódicamente el estado de la rama para activar o desactivar el circuito de iluminación.

El diagrama de flujo siguiente resume la programación del dispositivo actuador:

Figura 60. **Diagrama de flujo del dispositivo actuador**



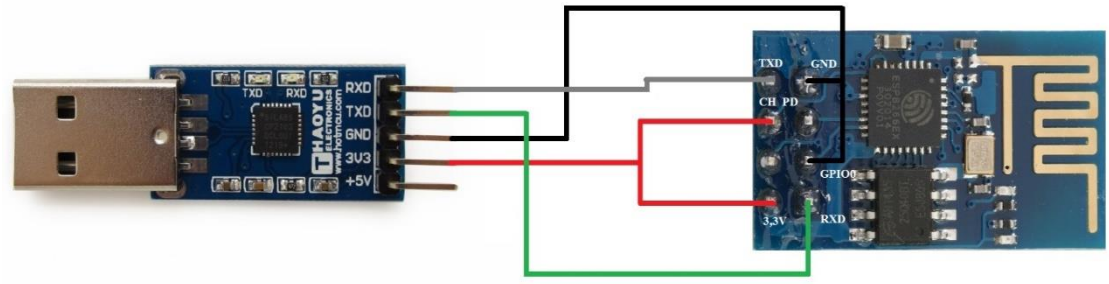
Fuente: elaboración propia.

4.3.3.3. Programación del dispositivo

Para programar el dispositivo con una computadora es necesario conectarlo a un adaptador de USB a UART. En el mercado hay varios modelos, pero funciona cualquiera que maneje valores de voltaje TTL y que tenga una línea de alimentación de 3.3 V.

Se debe conectar como muestra la figura siguiente:

Figura 61. **Diagrama de conexión de adaptador USB a UART a ESP-01S**



Fuente: PROMETEC. ESP8266 Plugin para Arduino IDE. www.prometec.net. Consulta: 6 de septiembre de 2019.

Una vez conectado se puede conectar a la computadora y utilizar el IDE de Arduino como si de un dispositivo Arduino se tratará. A continuación, se detallará el código fuente que se ha utilizado para programar el dispositivo que funcionará como rama del árbol. Este código se puede encontrar completo en el apéndice 2.

En primer lugar, es necesario agregar las bibliotecas que se utilizarán.

Figura 62. **Código fuente del actuador, segmento 1**

```
//Bibliotecas utilizadas
#include <ESP8266WiFi.h>
#include <ArduinoJson.h>
#include "FS.h"
#include <ESP8266WebServer.h>
#include <WiFiClient.h>
#include <ESP8266HTTPClient.h>
```

Fuente: elaboración propia

La biblioteca *ESP8266WiFi* agrega las funciones específicas del microcontrolador ESP8266, debe ser incluida cada vez que se programa un dispositivo basado en ese microcontrolador. ArduinoJson permite manejar objetos JSON, mientras FS es la encargada de manejar el sistema de archivos de la memoria *flash*. *ESP8266WebServer*, *WiFiClient* y *ESP8266HTTPClient* son bibliotecas que manejan la conexión a la red y la administración de servidores y clientes de HTTP.

Para mantener el código fuente principal limpio, se ha agregado un archivo adicional de encabezados que contiene la codificación en HTML de la interfaz de configuración, que será una interfaz gráfica donde el usuario podrá ingresar los parámetros de configuración del dispositivo. Este archivo debe ser incluido al inicio:

Figura 63. **Código fuente del actuador, segmento 2**

```
//Interfaz de configuración
#include "interface.h"
```

Fuente: elaboración propia.

El archivo *interface.h* contiene la siguiente codificación HTML. Disponible completa en el apéndice 3.

Figura 64. **Codificación de la interfaz de configuración del dispositivo, segmento 1**

```
const char INTERFACE[] PROGMEM = R"=====(<!DOCTYPE html>
<html lang="en">
<head>
  <title>Configure device</title>
</head>
<body>
  <h1>Device configuration</h1>
  <form method="POST">
    <h3>Device data</h3>
    <p>
      ID:
      <input name="id" type="text">
    </p>
    <p>
      Name:
      <input name="name" type="text">
    </p>
    <p>
      Function:
      <input name="function" type="text">
    </p>
```

Fuente: elaboración propia.

Figura 65. **Codificación de la interfaz de configuración del dispositivo, segmento 2**

```
<p>
  Location:
  <input name="location" type="text">
</p>
<p>
  Mode:
  <select name ="mode">
    <option value="">No change</option>
    <option value="INPUT">Input</option>
    <option value="OUTPUT">Output</option>
  </select>
</p>
<p>
  Type:
  <select name ="type">
    <option value="">No change</option>
    <option value="bool">boolean</option>
    <option value="number">number</option>
    <option value="text">string</option>
  </select>
</p>
```

Fuente: elaboración propia.

Figura 66. **Codificación de la interfaz de configuración del dispositivo, segmento 3**

```
<p>
  Refresh:
  <input type="number" name="refresh">
</p>
<h3>Network data</h3>
<p>
  Network SSID:
  <input type="text" name="ssid">
</p>
<p>
  Password:
  <input type="password" name="password">
</p>
<p>
  Tree:
  <input type="text" name="tree">
</p>
<p>
  Key:
  <input type="password" name="key">
</p>
```

Fuente: elaboración propia.

Figura 67. **Codificación de la interfaz de configuración del dispositivo, segmento 4**

```
    <p>
      Server:
      <input type="text" name="server">
    </p>
    <p>
      Port:
      <input type="number" name="port">
    </p>
    <input type="submit" name="submit" value="Update">
  </form>
</body>
</html>
)=====";
```

Fuente: elaboración propia.

Lo primero que se observa es la declaración de una cadena de caracteres llamada *INTERFACE*, esta será la que se llame desde el código fuente principal. A continuación, aparece la palabra PROGMEM que indica que la variable se deberá guardar en la memoria *flash* y no en la RAM. Esto se realiza para que la memoria RAM no se agote por almacenar cadenas de texto de larga longitud. Luego, aparece una literal de texto en bruto indicada con los caracteres R"=====()=====", esta función de C++ permite agregar una cadena de texto con caracteres especiales que no deben ser procesados por el compilador, esta función evita que el compilador interprete los símbolos de mayor y menor que y las comillas.

Posterior a esto, aparece la codificación HTML de la interfaz de configuración. Lo primero por notar es que aparece un formulario que utiliza el método de acceso POST para enviar los datos que ingresa el usuario. Luego, aparece cada parámetro de configuración que puede modificar el usuario:

identificador (*id*), nombre (*name*), función (*function*), ubicación (*location*), modo (*mode*), tipo (*type*), tiempo de actualización (*refresh*), nombre de la red (*network* SSID), contraseña de la red (*password*), árbol (*tree*) y llave (*key*). Por último, se agrega un botón que envía el formulario.

Figura 68. **Código fuente del actuador, segmento 3**

```
//Definiciones
#define OUTPUT_PIN 3
#define CONFIG_PIN 1
#define CONFIG_TIME 5
#define CONFIG_FIELDS {"id","name","function","location","mode",
"type","refresh","ssid","password","tree","key","server","port"}

//Objetos globales
ESP8266WebServer configServer(80);
HTTPClient http;
WiFiClient client;

//Variables globales
String server = "";
String tree = "";
String key = "";
String id = "";
String device = "";
int refresh = 5;
bool deviceState = false;
bool communicationError = false;
```

Fuente: elaboración propia.

Regresando al archivo del código fuente principal, después de la inclusión de bibliotecas y archivos adicionales, se colocan las definiciones y objetos y variables globales. Las definiciones serán valores que no cambiarán en el tiempo y se traducen al momento de la compilación, no se guardan como variables. La definición *OUTPUT_PIN*, representa el pin de salida, que será el que active el

circuito de control de iluminación, en este caso el pin 3. *CONFIG_PIN* será el pin que leerá el botón de configuración, *CONFIG_TIME* será el tiempo que se dará para que el usuario presione el botón de configuración y *CONFIG_FIELDS* serán los campos que el usuario podrá configurar en la interfaz de configuración.

Los objetos y variables globales son entidades a las que puede tener acceso cualquier función del código fuente. Los objetos globales ESP8266WebServer y *HTTPClient*, manejan un servidor y un cliente de HTTP respectivamente, mientras *WiFiClient*, permite conectarse a una red inalámbrica de área local. Se han agregado las variables *server*, *tree*, *key*, *id*, *device* y *refresh* que serán los parámetros configurables que se usarán en varias funciones.

La variable *deviceState* mantiene el estado de la rama, para este ejemplo será de tipo binario (*bool*) y la variable *communicationError* funciona como una bandera indicando que hubo un error al agregar la rama al árbol.

Figura 69. **Código fuente del actuador, segmento 4**

```
//Estado de configuración
void setup() {
  initPeripherals();
  int option = getButtonTimes(CONFIG_PIN,CONFIG_TIME);
  if(option == 1){
    configureDevice();
  }else if(option == 3){
    restoreDevice();
    configureDevice();
  }
  connectDevice();
  initDevice();
  int code = postDevice();
  if (code == 202){
    int code = putMetadata();
  }
  if(code/100 != 2){
    communicationError = true;
  }
}
```

Fuente: elaboración propia.

Cualquier programación en el IDE de Arduino se compone de dos funciones principales, una función de configuración (*setup*) y otra de ejecución periódica (*loop*). La primera solamente se ejecuta una vez al inicio y la segunda se ejecuta de forma cíclica indeterminadamente. Dentro de la función de configuración se ha agregado la parte de configuración inicial que debe realizar el dispositivo.

En esta función se agrega la función *initPeripherals*() que inicializa los pines del dispositivo, la función *getButtonTimes*() que detecta las veces que se pulsó el botón de configuración y dependiendo si se presionó una o tres veces se

ejecuta la función *configureDevice*(), que permite configurar el dispositivo, o la función *restoreDevice*() que restablece los valores por defecto.

Finalizada la configuración del dispositivo se ejecutan las funciones *connectDevice*(), que conecta el dispositivo a la red inalámbrica de área local, y la función *initDevice*(), que inicializa la representación de la rama en formato de objeto JSON. Posteriormente, las funciones *postDevice*() y *putMetadata*() se encargan de agregar la rama al árbol y actualizar sus metadatos si ya fue agregada con anterioridad respectivamente. Si la respuesta a las peticiones de HTTP no tiene un código cuya centena es 2 levanta la bandera *communicationError* indicando que hubo un error al agregar la rama o actualizar sus metadatos.

Figura 70. **Código fuente del actuador, segmento 5**

```
//Estado de ejecución
void loop() {
  if(!communicationError){
    String command = getData();
    executeCommand(command);
    delay(refresh*1000);
  }
}
```

Fuente: elaboración propia.

Dentro de la función de ejecución aparece la función *getData*(), que obtiene el estado de la rama, y *executeCommand*(), que envía la señal que indica el estado de la rama hacia el pin de salida. Ambas funciones se ejecutan de forma

periódica, con una separación en segundos que indica la variable global *refresh*, si no existe un error al agregar la rama al árbol.

Figura 71. **Código fuente del actuador, segmento 6**

```
//Inicialización de puertos de entrada y salida
void initPeripherals(){
  SPIFFS.begin();
  pinMode(CONFIG_PIN, INPUT_PULLUP);
  pinMode(OUTPUT_PIN, OUTPUT);
  digitalWrite(OUTPUT_PIN,LOW);
}
```

Fuente: elaboración propia.

La función *initPeripherals*(), inicializa el sistema de archivos (SPIFFS), establece el modo de los pines, ya sea de entrada o salida, y escribe un valor inicial para el pin de salida.

Figura 72. **Código fuente del actuador, segmento 7**

```
//Obtiene la cantidad de pulsaciones de un botón
int getButtonTimes(int button, int seconds){
  int t = millis();
  int times = 0;
  int actual = 0;
  int previous = 1;
  while((millis() - t) < seconds*1000){
    ESP.wdtFeed();
    for(int i = 0; i<1000; i++){
      actual = actual + digitalRead(button);
    }
    actual = actual/1000;
    if (actual - previous == -1){
      times = times + 1;
    }
    previous = actual;
  }
  return times;
}
```

Fuente: elaboración propia.

La función *getButtonTimes*() toma como parámetros el pin de donde se lee el valor del botón y el tiempo en el que se lee el pin. Posteriormente, establece la variable t como el tiempo inicial con la función *millis*() y mientras la diferencia de tiempo, actual menos t, no sobrepase el tiempo de lectura realiza un promedio de cada mil muestras y detecta los flancos de bajada, por ser el pin de lectura una entrada con una resistencia *pull-up*, es decir, que lee un valor alto mientras no se pulse el botón y uno bajo cuando se pulsa.

Es necesario agregar la función ESP.*wdtFeed*() dentro del ciclo *while* para reiniciar el temporizador de perro guardián y evitar que el microcontrolador active una interrupción, relacionada a ese temporizador, que detenga permanentemente la ejecución del programa.

Figura 73. **Código fuente del actuador, segmento 8**

```
//Restablece la configuración por defecto
void restoreDevice(){
  StaticJsonDocument<1024> json;
  getJsonFromFile(&json,"default.txt");
  saveJsonToFile(&json,"config.txt");
}

//Configuración de datos
void configureDevice(){
  setConfigNetwork();
  setConfigServer();
}
```

Fuente: elaboración propia.

La función *restoreDevice*() copia los datos que se encuentran en el archivo default.txt hacia el archivo *config.txt*, siendo el primero donde se encuentran los valores de configuración por defecto y el segundo donde se almacenan los valores de configuración que ha ingresado el usuario. Con esto se logra restablecer los valores de configuración a los valores por defecto.

Por otra parte, la función *configureDevice*() hace llamadas a las funciones *setConfigureNetwork*() y *setConfigServer*().

Figura 74. **Código fuente del actuador, segmento 9**

```
//Levanta la red de configuración
void setConfigNetwork(){
  String ssid = "IoTDevice";
  String pass = "1234567890";
  WiFi.mode(WIFI_AP);
  while(!WiFi.softAP(ssid,pass)){
    delay(500);
  }
}

//Levanta el servidor que muestra la interfaz en la dirección 192.168.4.1
void setConfigServer(){
  configServer.on("/",showConfigInterface);
  configServer.begin();
  while(configServer.arg("submit")!="Update"){
    configServer.handleClient();
  }
}

//Muestra la interfaz de configuración
void showConfigInterface(){
  String webpage = INTERFACE;
  configServer.send(200,"text/html",webpage);
  saveFormData();
}
```

Fuente: elaboración propia.

La función *setConfigNetwork*(), configura el microcontrolador como un punto de acceso inalámbrico y levanta una red de área local para que el usuario se pueda conectar a ella y acceder a la interfaz de configuración del dispositivo. Mientras, la función *setConfigServer*() activa un servidor web en la dirección 192.168.4.1 que será el encargado de manejar la interfaz de configuración.

La función *showConfigInterface*() será la que muestra el código HTML que se encuentra en el archivo *interface.h* cada vez que se envía una petición hacia el servidor en el microcontrolador, además, hace una llamada hacia la función *saveFormData*() para almacenar los datos recibidos.

Figura 75. **Código fuente del actuador, segmento 10**

```
//Almacena los datos obtenidos por la interfaz de configuración
void saveFormData(){
  String configFields[] = CONFIG_FIELDS;
  bool hasChanged = false;
  StaticJsonDocument<1024> json;
  getJsonFromFile(&json, "config.txt");
  for(int i=0; i<13; i++){
    String value = configServer.arg(configFields[i]);
    value.trim();
    if(value!=""){
      if(configFields[i]=="refresh" or configFields[i]=="port"){
        json[configFields[i]] = value.toInt();
      }else{
        json[configFields[i]] = value;
      }
      hasChanged = true;
    }
  }
  if (hasChanged){
    saveJsonToFile(&json,"config.txt");
  }
}
```

Fuente: elaboración propia.

La función *saveFormData*(), recorre la lista de parámetros de configuración obteniendo el valor que el usuario ha ingresado en la interfaz de configuración para cada uno. Si el usuario no ha ingresado ningún valor no se levanta la bandera *hasChanged*, que indica que hubo algún cambio, esto se realiza para evitar escrituras a la memoria *flash* innecesarias. De haber algún cambio, se guardan los datos en el archivo *config.txt*.

Figura 76. **Código fuente del actuador, segmento 11**

```
//Conecta el dispositivo a la red
void connectDevice(){
  StaticJsonDocument<1024> json;
  getJsonFromFile(&json,"config.txt");
  WiFi.mode(WIFI_STA);
  WiFi.begin(json["ssid"].as<String>(),json["password"].as<String>());
  while(WiFi.status() != WL_CONNECTED){
    delay(500);
  }
}

//Inicializa la imágen virtual de la rama
void initDevice(){
  if(WiFi.status() != WL_CONNECTED){
    connectDevice();
  }
  StaticJsonDocument<1024> json;
  getJsonFromFile(&json,"config.txt");
  server = json["server"].as<String>();
  id = json["id"].as<String>();
  refresh = json["refresh"];
  tree = json["tree"].as<String>();
  key = json["key"].as<String>();
  device = "{\""+id+"\":{\"meta\":{\"ip\":\""+WiFi.localIP().toString()+
  "\",\"mac\":\""+WiFi.macAddress()+"\",\"tree\":\""+tree+"\",\"name\":\""+
  json["name"].as<String>()+"\",\"function\":\""+json["function"].as<String>()+
  "\",\"location\":\""+json["location"].as<String>()+"\",\"type\":\""+
  json["type"].as<String>()+"\",\"mode\":\""+json["mode"].as<String>()+
  "\",\"refresh\":"+refresh+"},\"data\":"+deviceState+"}}";
}
```

Fuente: elaboración propia.

Una vez el usuario ha terminado de configurar el dispositivo, este se intenta conectar a la red inalámbrica de área local que el usuario le ha indicado. Esto lo realiza la función *connectDevice*(), que toma el parámetro *ssid* y *password* del archivo donde se ha guardado la configuración e intenta establecer la conexión hasta que es exitosa.

A continuación la función *initDevice*() forma una cadena de texto, llamada *device*, con los datos guardados en el archivo *config.txt*, que será la

representación de la rama que se enviará al tronco. El resultado de esta cadena se muestra en la figura a continuación.

Figura 77. **Representación completa del dispositivo actuador de ejemplo**

Fuente: elaboración propia.

Figura 78. **Código fuente del actuador, segmento 12**

```
//Agrega la rama al árbol
int postDevice(){
  if(WiFi.status() != WL_CONNECTED){
    connectDevice();
  }
  http.begin(client, server + "/devices");
  http.addHeader("Content-Type", "application/json");
  http.setAuthorization(tree.c_str(), key.c_str());
  int code = http.POST(device);
  http.end();
  return code;
}

//Modifica los metadatos de la rama
int putMetadata(){
  if(WiFi.status() != WL_CONNECTED){
    connectDevice();
  }
  String meta = "";
  StaticJsonDocument<1024> json;
  deserializeJson(json,device);
  json = json["meta"];
  serializeJson(json,meta);
  http.begin(client, server + "/devices/meta");
  http.addHeader("Content-Type", "application/json");
  http.setAuthorization(tree.c_str(), key.c_str());
  int code = http.PUT(meta);
  http.end();
  return code;
}
```

Fuente: elaboración propia.

Una vez el dispositivo ha sido inicializado, las funciones *postDevice*() y *putMetadata*() se encargan de agregar el dispositivo a la rama y actualizar sus metadatos respectivamente. Como toda petición, deben contener las credenciales de autenticación *tree* y *key*.

Figura 79. **Código fuente del actuador, segmento 13**

```
//Obtiene el estado de la rama
String getData(){
  if(WiFi.status() != WL_CONNECTED){
    connectDevice();
  }
  http.begin(client, server + "/devices/"+id+"/data");
  http.setAuthorization(tree.c_str(), key.c_str());
  int code = http.GET();
  if(code == 200){
    String text = http.getString();
    http.end();
    StaticJsonDocument<1024> json;
    deserializeJson(json,text);
    text = json[id].as<String>();
    if(text == ""){
      communicationError = true;
    }
    return text;
  }else{
    http.end();
    return "";
  }
}

//Exterioriza el estado de la rama
void executeCommand(String command){
  if (command!=""){
    if(command == "true"){
      deviceState = true;
      digitalWrite(OUTPUT_PIN, HIGH);
    }else{
      digitalWrite(OUTPUT_PIN, LOW);
      deviceState = false;
    }
  }
}
```

Fuente: elaboración propia.

La función *getData*() se encarga de enviar una petición con el método de acceso GET hacia el tronco para obtener el estado de la rama. Si se obtiene una cadena de texto vacía, se levanta la bandera *communicationError* indicando que hubo un error en la comunicación porque la rama ha sido eliminada del árbol.

Por otra parte, la función *executeCommand*() recibe como parámetro el estado de la rama y se encarga de enviar una señal con valor de voltaje alto hacia el circuito de iluminación, si el estado es *true* y una señal con valor de voltaje bajo si el estado es *false*.

Figura 80. **Código fuente del actuador, segmento 14**

```
//Guarda un objeto json a un archivo de texto
void saveJsonToFile(StaticJsonDocument<1024> *json, String file){
  String text = "";
  serializeJsonPretty(*json,text);
  if (text != ""){
    File f = SPIFFS.open("/"+file,"w+");
    f.print(text);
    f.close();
  }
}

//Obtiene un objeto json de un archivo de texto
void getJsonFromFile(StaticJsonDocument<1024> *json, String file){
  String text = "";
  File f = SPIFFS.open("/"+file,"r");
  text = f.readString();
  deserializeJson(*json,text);
  f.close();
}
```

Fuente: elaboración propia.

Las funciones *saveJsonToFile*() y *getJsonFromFile*() son funciones auxiliares que toman como parámetros un objeto JSON y el nombre de un archivo. La primera guarda un objeto JSON hacia un archivo y la segunda obtiene un objeto JSON desde un archivo. Estas funciones son utilizadas cuando se necesita acceder a la información guardada en los archivos *config.txt* y *default.txt*.

Antes de cargar el código hacia el microcontrolador, es necesario cargar los archivos *default.txt* y *config.txt*, que será donde se encuentre la configuración por

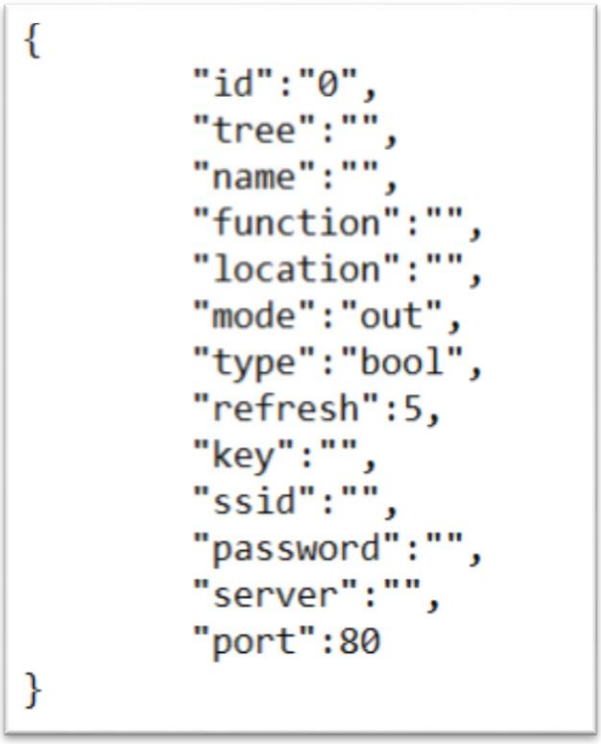
defecto y la configuración del usuario, respectivamente. Estos archivos se deben agregar en una carpeta llamada data dentro de la carpeta del proyecto.

Figura 81. **Carpeta de archivos de configuración**

Fuente: elaboración propia, basada en explorador de archivos de Windows 10.

El archivo *config.txt* originalmente contiene una copia del archivo *default.txt*. La información que contienen es la siguiente:

Figura 82. **Archivo de configuración por defecto**



Fuente: elaboración propia.

Antes de subir los archivos de configuración, desde el menú Herramientas, en la opción *Flash Size*, se debe seleccionar una que corresponda al tamaño de la memoria *flash* del módulo utilizado y que contenga entre paréntesis un tamaño para el sistema de archivos, SPIFFS. Para este ejemplo se utilizó la opción 512K (32K SPIFSS). Esta opción reserva 32 KB memoria *flash* para montar el sistema de archivos.

Para subir los archivos de configuración al sistema de archivos del módulo ESP-01S, desde el menú Herramientas del IDE de Arduino se selecciona la opción ESP8266 *Sketch Data Upload*. Esto toma los archivos en la carpeta data

del proyecto y los guarda en el sistema de archivos instalado en la memoria *flash* del módulo.

Por último, al presionar el botón Verificar se compilará el código y el botón Subir lo subirá al a memoria *flash* del microcontrolador. Al subir el código se ha terminado de programar el dispositivo actuador.

4.3.4. Configuración de interfaz de voz

En este ejemplo de implementación del protocolo se permitirá que un usuario pueda dictar comandos en lenguaje natural a través del asistente Alexa de Amazon utilizando un módulo Echo Dot.

Figura 83. **Amazon Echo Dot (segunda generación)**

Fuente: iTechDeals. Amazon Echo Dot 2nd Gen. www.itechdeals.com. Consulta: 23 de septiembre de 2019.

Para que el asistente Alexa reconozca los comandos de voz y envíe las peticiones correctas será necesario crear una habilidad (*skill*) de Alexa. Una

habilidad de Alexa es un servicio que interpreta los comandos que han sido dictados por voz y puede presentar una respuesta o ejecutar una acción según se le indique. Las acciones por ejecutar son definidas en una función que se aloja en un servidor en internet. En este ejemplo se utilizan los servicios web de Amazon (*Amazon Web Services*).

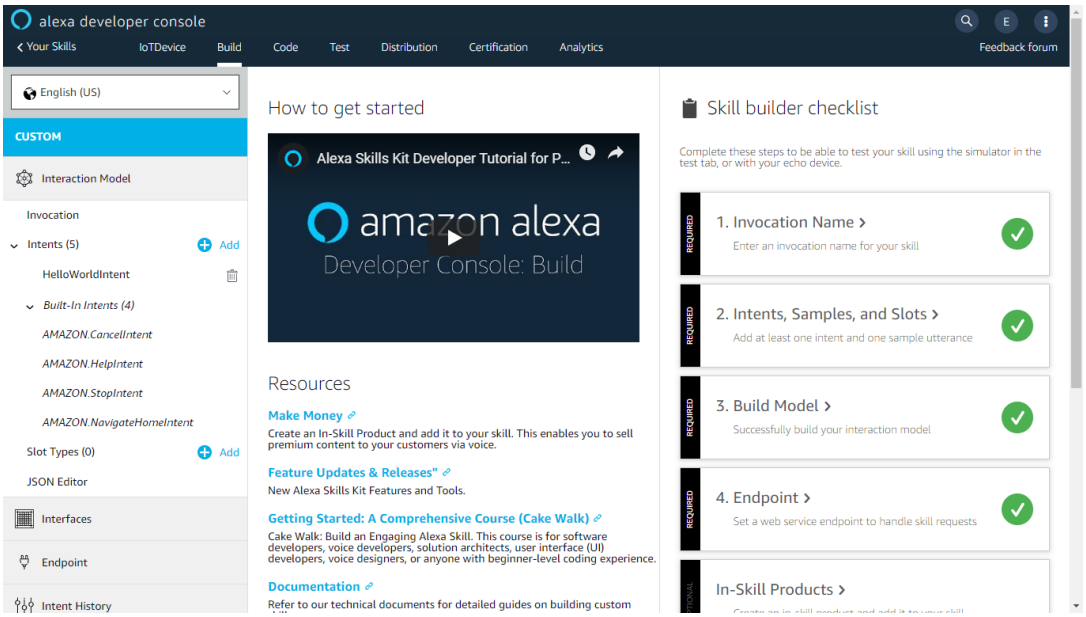
4.3.4.1. Configuración de intérprete de comandos

La habilidad de Alexa es la encargada de interpretar los comandos dictados por voz y reconocer la acción que el usuario desea ejecutar. Esta debe ser creada para cada aplicación de forma específica. Para esto es necesario crear una cuenta de desarrollador de Amazon en la página web https://developer.amazon.com.

Una vez se ha accedido a la cuenta se debe seleccionar Amazon Alexa entre los Servicios y Tecnologías de Desarrollador de Amazon (Amazon Developer Services and Technologies). Luego seleccionar la opción de Crear Habilidades de Alexa (Create Alexa Skills) y seguido, la opción Empezar una habilidad.

Esto llevará a una interfaz donde se podrá nombrar la habilidad y crearla. Entre las opciones que en esa interfaz se ofrecen se debe seleccionar la opción de modelo personalizado y el método para alojar los servicios *backend* como Alexa-Hosted (Python). Al momento de presionar el botón de Crear Habilidad se abrirá una interfaz de configuración de la habilidad.

Figura 84. **Interfaz de configuración de la habilidad**



Fuente: elaboración propia, basado en interfaz de desarrollador de Amazon.

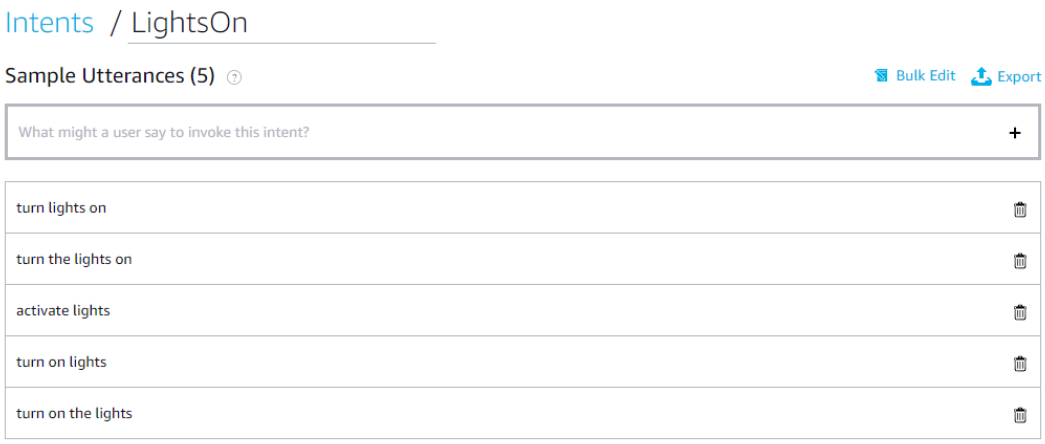
Una habilidad está compuesta por una invocación (*invocation*) y uno o varios intentos (*intents*). A su vez, cada intento está compuesto por varios enunciados (*utterances*).

La invocación será el nombre por el que el asistente Alexa reconocerá la habilidad y la abrirá. Este nombre puede ser cualquiera que sea significativo para la aplicación que se está realizando, pero debe contener más de una palabra. En este ejemplo se ha utilizado el nombre árbol de dispositivos (*tree of devices*). Los intentos serán las representaciones de las acciones que desea realizar el usuario, mientras los enunciados serán las frases con las que se reconocerá un intento.

En el panel izquierdo de la interfaz de configuración, bajo el menú Modelo de Interacción (*Interaction Model*), se puede configurar la invocación, añadir intentos y configurar sus enunciados.

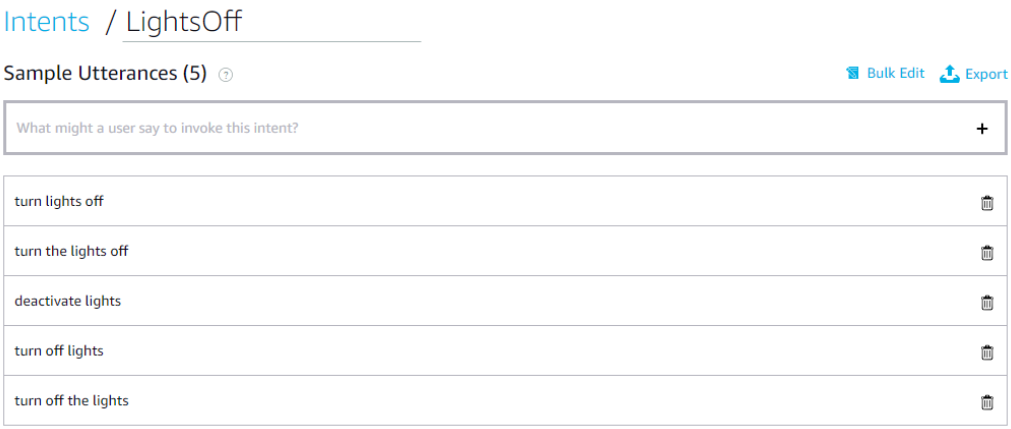
En este ejemplo se han creado dos intentos para activar o desactivar el circuito de iluminación, llamados *LightsOn* y *LightsOff* respectivamente. Cada uno con diferentes frases que el usuario podría utilizar para activarlo. Se ha utilizado el inglés debido a que el módulo Echo Dot no tiene soporte para el idioma español.

Figura 85. **Intento de activación del circuito de iluminación**



Fuente: elaboración propia, basado en interfaz de desarrollador de Amazon.

Figura 86. **Intento de desactivación del circuito de iluminación**



Fuente: elaboración propia, basado en interfaz de desarrollador de Amazon

Una vez creados los intentos, se regresa a la interfaz de configuración con la opción personalizar (*custom*) y se debe construir el modelo con la opción en el panel derecho llamada Construir modelo (*Build model*).

En la sección Terminación (*Endpoint*) se enlazará la habilidad con una función donde se describirá la acción a ejecutar por cada intento, está es llamada función lambda. En el recuadro de Región por defecto se copia el identificador (ARN) de la función lambda. A continuación, se crea esta función.

4.3.4.2. Configuración de una función de ejecución de comandos

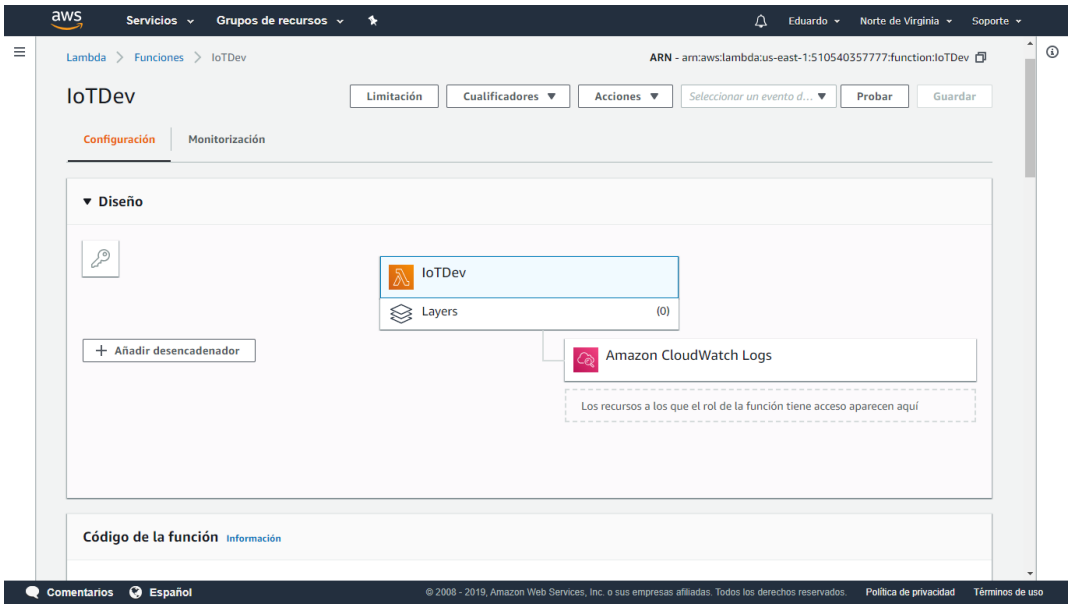
Una vez creada la habilidad de Alexa, el módulo Echo Dot puede reconocer los intentos, más aún no puede ejecutar acciones. Estas acciones por ejecutar son descritas en un archivo de Python alojado en un servidor en internet. Este

archivo que contiene la funcionalidad de la habilidad se le conoce como función lambda.

Para crear una función lambda será necesario crear una cuenta gratuita en los Servicios Web de Amazon, disponible a través del enlace http://aws.amazon.com. Una vez accedido, es necesario seleccionar como región el Norte de Virginia, pues las habilidades de Alexa solo tienen soporte en esa región.

En la Consola de administración de AWS se busca el servicio Lambda y se selecciona la opción Crear una función. Esto lleva a una interfaz de creación donde se selecciona la opción Crear desde cero, se asigna un nombre a la función y se selecciona Python 2.7 entre Tiempo de ejecución. Al crear la función se accede a una interfaz de configuración de la función.

Figura 87. **Interfaz de configuración de función lambda**



Fuente: elaboración propia, basado en sitio de Amazon Web Services.

El ARN que se presenta en la sección superior se debe añadir en la sección *Endpoint* de la interfaz de configuración de la habilidad. Luego con el botón Añadir desencadenador se debe agregar *Alexa Skills Kit* para permitir que la habilidad de Alexa utilice la función lambda, esto necesitará el identificador de habilidad que se encuentra también en la sección *Endpoint* de la interfaz de configuración de la habilidad.

Una vez agregado el desencadenador, se puede agregar el código en Python que describe la acción a ejecutar en la sección Código de la función.

La habilidad de Alexa y la función lambda se comunican a través de objetos JSON. La habilidad envía un objeto con datos de sesión, usuario, región y petición. Utiliza tres tipos de peticiones principalmente, de lanzamiento (*LaunchRequest*), de intento (*IntentRequest*) y de finalización de sesión (*SessionEndedRequest*). La función lambda deberá manejar estas peticiones.

Figura 88. **Objeto JSON enviado por la habilidad de Alexa**

Fuente: elaboración propia.

La función lambda devolverá un objeto JSON para cada petición de la habilidad. Este contendrá tres elementos principales, la respuesta en voz, la respuesta escrita y la opción de terminar la sesión.

Figura 89. **Objeto JSON enviado por la función lambda**

Fuente: elaboración propia.

A continuación, se detalla el código a colocar en la función lambda. Este se encuentra completo en el apéndice 4.

Figura 90. **Código fuente de la función lambda, segmento 1**

```
from botocore.vendored import requests
import json

def lambda_handler(event, context):
    if event["request"]["type"] == "LaunchRequest":
        return on_launch()
    elif event["request"]["type"] == "IntentRequest":
        return on_intent(event["request"])
    elif event["request"]["type"] == "SessionEndedRequest":
        return on_end()
```

Fuente: elaboración propia.

Lo primero que se debe hacer es importar las bibliotecas necesarias y a continuación declarar una función con el nombre *lambda_handler* que será la que se ejecutará en primer lugar. Esta detecta el tipo de petición y llama a otra función que ejecute una funcionalidad para cada petición.

Figura 91. **Código fuente de la función lambda, segmento 2**

```
def on_launch():
    session_attributes = {}
    card_title = "Welcome"
    speech_output = "Welcome to the voice interface for your devices "
    reprompt_text = ""
    should_end_session = False
    return build_response(session_attributes, build_speechlet(
        card_title, speech_output, reprompt_text, should_end_session))

def on_intent(intent_request):
    intent = intent_request["intent"]
    intent_name = intent_request["intent"]["name"]
    if intent_name == "LightsOn":
        return turn_on_lights()
    elif intent_name == "LightsOff":
        return turn_off_lights()

def on_end():
    session_attributes = {}
    card_title = "Goodbye"
    speech_output = ""
    reprompt_text = "Have a good day"
    should_end_session = True
    return build_response(session_attributes, build_speechlet(
        card_title, speech_output, reprompt_text, should_end_session))
```

Fuente: elaboración propia.

La función *on_launch*(), que se ejecuta cuando se invoca a la habilidad, envía un mensaje de bienvenida, indicando que ha iniciado la ejecución de la habilidad. La función *on_intent*() toma como parámetro el intento y llama a otra función dependiendo del nombre del mismo. Luego, la función *on_end*() se ejecuta cuando se pide una finalización de sesión.

Figura 92. **Código fuente de la función lambda, segmento 3**

```
def turn_on_lights():
    session_attributes = {}
    card_title = "LightsOn"
    reprompt_text = ""
    should_end_session = False
    speech_output = "Turning on the lights"
    r = requests.put('http://user.pythonanywhere.com/devices/data', auth=('tree', 'key'),
                     json={"2":True})
    return build_response(session_attributes, build_speechlet(
        card_title, speech_output, reprompt_text, should_end_session))

def turn_off_lights():
    session_attributes = {}
    card_title = "LightsOff"
    reprompt_text = ""
    should_end_session = False
    speech_output = "Turning off the lights"
    r = requests.put('http://user.pythonanywhere.com/devices/data', auth=('tree', 'key'),
                     json={"2":False})
    return build_response(session_attributes, build_speechlet(
        card_title, speech_output, reprompt_text, should_end_session))
```

Fuente: elaboración propia.

Las funciones *turn_on_lights*() y *turn_off_ligths*(), envían una petición HTTP hacia el tronco utilizando el método de acceso PUT para modificar el estado de la rama. Enviando sus datos de autenticación.

Figura 93. **Código fuente de la función lambda, segmento 4**

```
def build_speechlet(title, output, reprompt_text, should_end_session):
    return {
        "outputSpeech": {
            "type": "PlainText",
            "text": output
        },
        "card": {
            "type": "Simple",
            "title": title,
            "content": output
        },
        "reprompt": {
            "outputSpeech": {
                "type": "PlainText",
                "text": reprompt_text
            }
        },
        "shouldEndSession": should_end_session
    }

def build_response(session_attributes, speechlet_response):
    return {
        "version": "1.0",
        "sessionAttributes": session_attributes,
        "response": speechlet_response
    }
```

Fuente: elaboración propia.

Las funciones *build_response*() y *build_speechlet*() son las encargadas de construir el objeto JSON que se envía como respuesta para cada petición de la habilidad de Alexa.

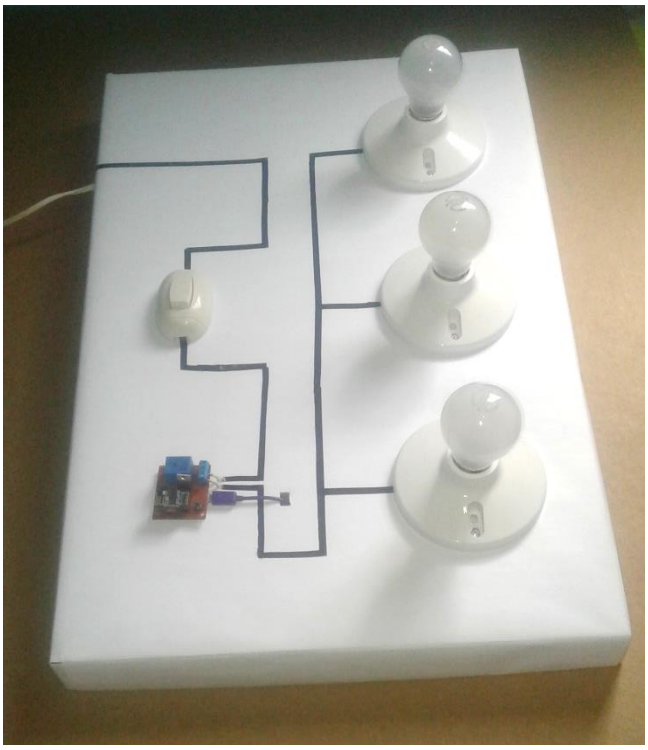
4.4. Resultado de la implementación

A continuación, se documenta con fotografías, el funcionamiento del ejemplo que se ha propuesto en este capítulo.

4.4.1. Conectando el circuito de iluminación

El circuito de iluminación se arma con tres bombillas incandescentes de 100 W conectadas en paralelo con cable paralelo de cobre de calibre 18 AWG, en serie está colocado el dispositivo actuador. Se ha añadido además un interruptor de seguridad en serie para desconectar el circuito de forma manual.

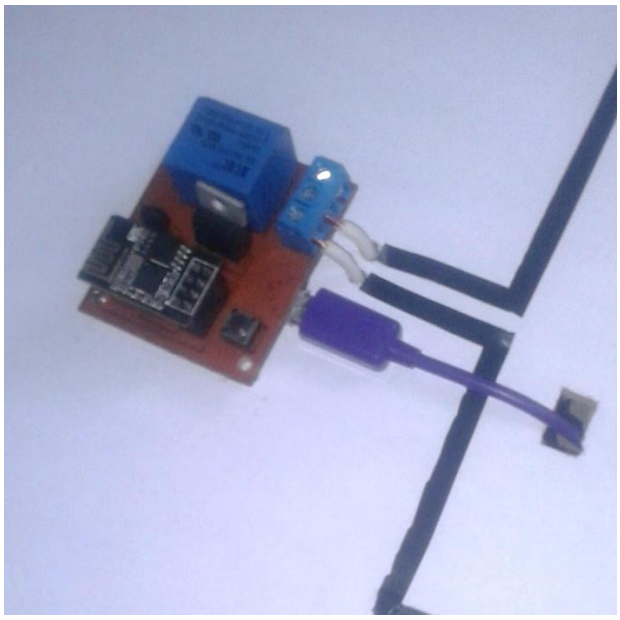
Figura 94. **Conexión de circuito de iluminación**



Fuente: elaboración propia.

Para conectar el dispositivo actuador, el cable vivo del circuito se corta y ambas puntas se conectan en las terminales NO y COM del dispositivo actuador. En el puerto microUSB se conecta la fuente de 5 V.

Figura 95. **Conexión de dispositivo actuador**

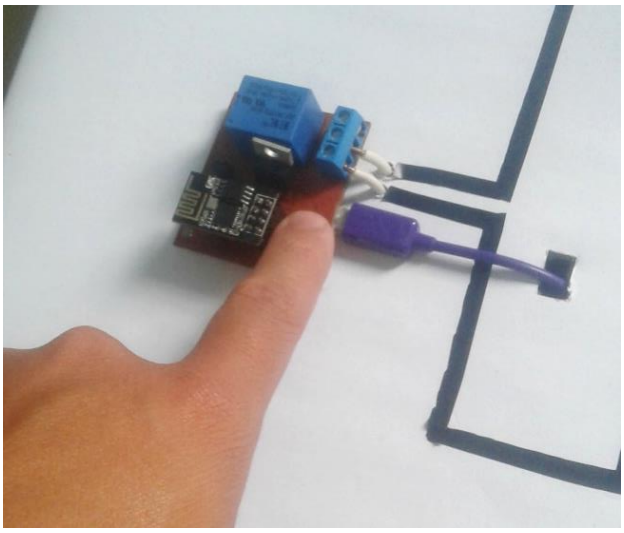


Fuente: elaboración propia.

4.4.2. Conectando el dispositivo a la red

Al conectar el dispositivo actuador, dentro de los primeros 5 s se debe presionar una vez el botón de configuración.

Figura 96. **Conectando el dispositivo a la red, paso 1**



Fuente: elaboración propia.

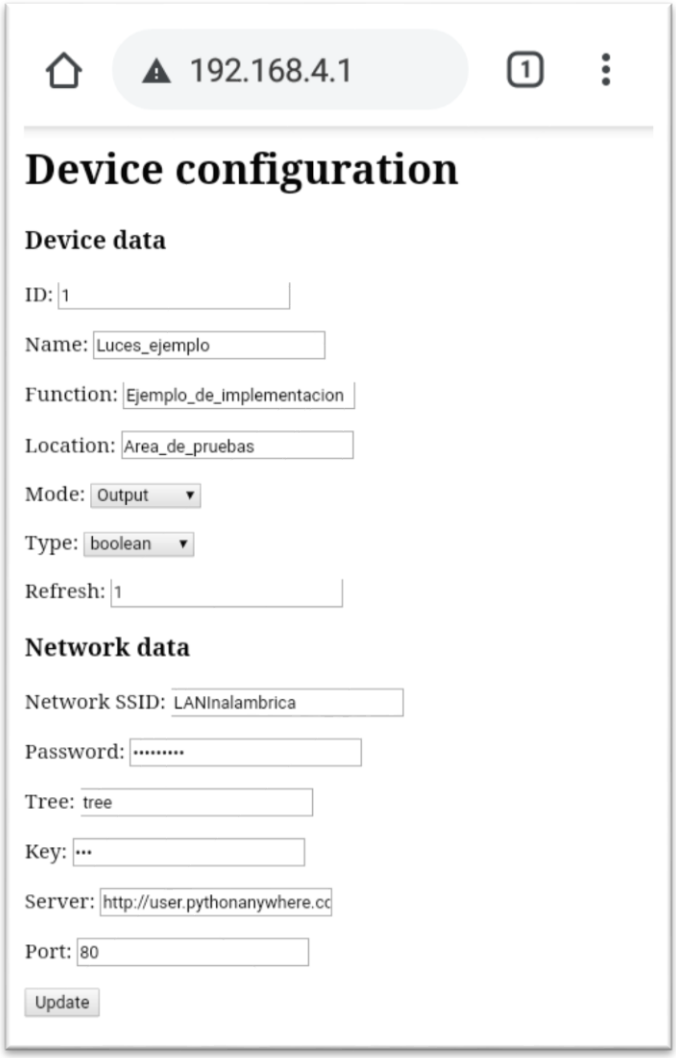
Con algún dispositivo móvil o una computadora con conexión inalámbrica se accede a la red IoTDevice.

Figura 97. **Conectando el dispositivo a la red, paso 2**

Fuente: elaboración propia, basada en captura de pantalla de *smartphone.*

Luego desde el navegador se accede a la dirección 192.168.4.1 y se ingresan los datos del dispositivo y los datos de la red a la que se debe conectar. Por último, solo resta guardar la configuración.

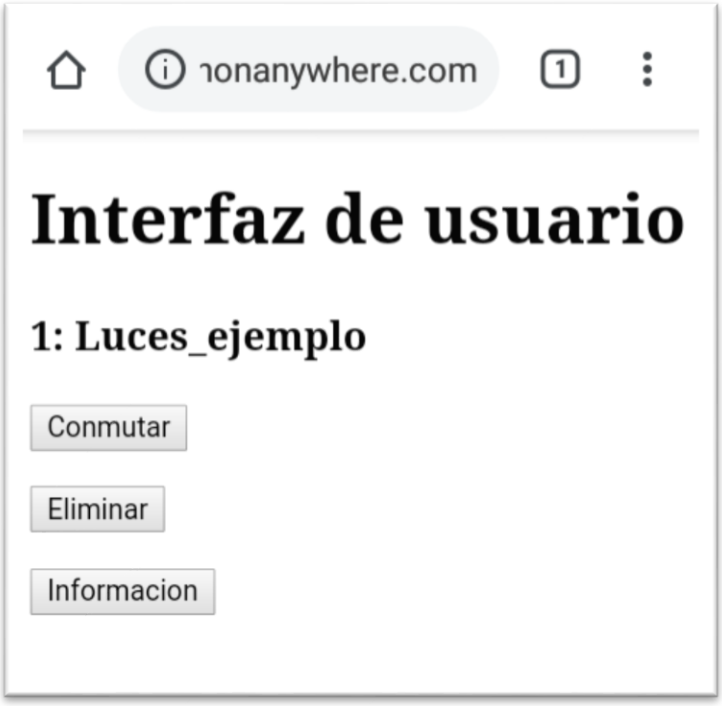
Figura 98. **Conectando el dispositivo a la red, paso 3**



Fuente: elaboración propia, basada en captura de pantalla de *smartphone.*

Si la configuración de la red es correcta el dispositivo se agrega al árbol. Esto se puede ver desde la interfaz de control de usuario.

Figura 99. **Dispositivo agregado al árbol**

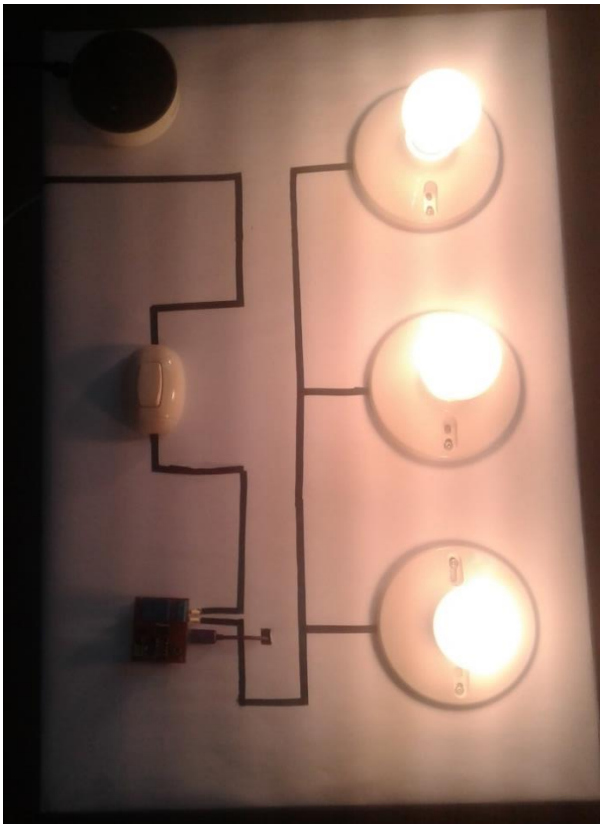


Fuente: elaboración propia, basada en captura de pantalla de *smartphone.*

4.4.3. Prueba de funcionamiento

El dispositivo actuador puede ser controlado utilizando la interfaz de control usuario. O utilizando comandos dictados en inglés a través del módulo Echo Dot, primero invocando la habilidad y luego utilizando los comandos reconocidos en los intentos o invocando la habilidad en conjunto con el comando.

Figura 100. **Circuito de iluminación en funcionamiento**



Fuente: elaboración propia.

Figura 101. **Módulo Echo Dot en funcionamiento**

Fuente: elaboración propia.

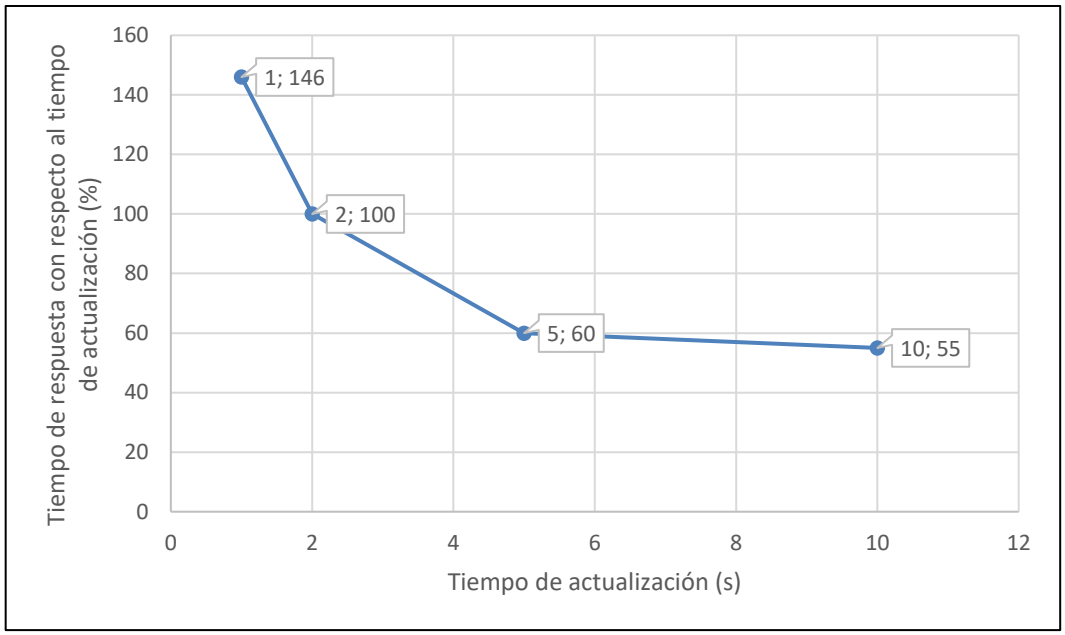
Se realizaron mediciones del tiempo de respuesta, modificando el tiempo de actualización (Ta) para obtener la diferencia entre la respuesta del circuito utilizando la interfaz web y utilizando la interfaz de voz, así como la diferencia entre el tiempo de respuesta y el tiempo de actualización del dispositivo actuador.

Tabla III. **Tiempo de respuesta en segundos con variación del tiempo de actualización**

Ta = 1 s		**Ta = 2 s**		**Ta = 5 s**		**Ta = 10 s**	
Web	**Voz**	**Web**	**Voz**	**Web**	**Voz**	**Web**	**Voz**
1,27	2,23	1,53	2,10	3,81	1,84	4,68	4,45
0,86	1,19	1,45	1,77	2,24	2,84	2,17	6,68
1,52	1,95	2,62	1,51	1,05	4,58	2,42	3,08
1,65	1,18	1,45	2,10	1,78	2,89	3,66	5,95
1,12	1,67	0,73	3,28	2,49	1,77	7,65	8,09
1,12	1,71	2,62	2,43	2,75	6,21	1,37	9,14
2,11	2,43	1,91	3,02	3,34	2,63	6,80	1,77
0,67	0,87	1,32	2,30	3,79	1,58	5,76	5,30
2,43	1,65	1,06	3,02	5,49	4,26	9,27	8,96
0,61	0,93	2,30	1,58	3,14	1,91	3,60	9,61

Fuente: elaboración propia.

Figura 102. **Tiempo de respuesta con respecto al tiempo de actualización en porcentaje**



Fuente: elaboración propia, basada en la tabla III.

En promedio la interfaz de voz tiene un retraso de 0,62 s respecto de la interfaz web. Por otra parte, se puede observar que mientras mayor es el tiempo de actualización, el tiempo de respuesta es más corto respecto del primero. Teóricamente, el tiempo de respuesta llegaría a ser el 50 % del tiempo de actualización cuando el retraso por el software y respuesta del servidor sea despreciable, pues el estado de una rama puede ser modificado en cualquier tiempo entre una actualización y otra con la misma probabilidad.

CONCLUSIONES

1. El uso de protocolos genéricos basados en estándares existentes favorece la integración del Internet de las Cosas dentro de la domótica, pues equipos heterogéneos, de diversos fabricantes, en conjunto con aplicaciones en internet, pueden comunicarse dentro del sistema de automatización y control.

2. El protocolo de transferencia de hipertexto es un protocolo robusto, utilizado en muchos servidores web en el mundo e integrado en todos los dispositivos con conectividad wifi o Ethernet, por tanto, el protocolo propuesto puede ser implementado en cualquier dispositivo que utilice protocolo de internet, con capacidades mínimas de procesamiento.

3. Las representaciones enviadas en formato de objeto JSON utilizan una menor cantidad de bytes para enviar la información hacia el servidor, por tanto, el uso de este formato, en conjunto con la utilización adecuada de cada representación mantiene al mínimo la carga de información en la red.

4. El protocolo de transferencia de hipertexto presenta una medida de identificación mínima, más no es suficiente para evitar ataques de interceptación de información en redes públicas.

5. Un servidor web centralizado puede reducir la cantidad de dispositivos utilizados, pues permite ejecutar varios roles en la misma instancia, actuando como dispositivo central, dispositivo de control e interfaz de usuario accesible a través de navegadores web.

6. El estándar REST es utilizado por una gran cantidad de servidores de contenido hipermedia y aplicaciones en internet, por lo tanto, el protocolo propuesto puede ser integrado con otros servidores que utilizan el mismo estándar.

7. El modelo de comunicación cliente-servidor permite delegar a los dispositivos periféricos la tarea de adición al conjunto de dispositivos y su modificación, eliminando esas tareas del servidor, actuando este último como mediador de actuadores y sensores.

8. Para un dispositivo con un tiempo de actualización configurado mayor que 10 s su tiempo de respuesta será aproximadamente el 50 % del tiempo de actualización.

RECOMENDACIONES

1. Cuando el protocolo propuesto se implementa en redes públicas es necesario que se agreguen medidas de seguridad adicionales para evitar ataques de interceptación de información.

2. El servidor central puede actuar como dispositivo de control automatizado y control de usuario, por consecuencia, permite reducir la cantidad de dispositivos por integrar en el sistema.

3. El formato de representaciones JSON tiene una carga menor en el ancho de banda de la red en comparación con el formato XML, por tanto, el primero debe utilizarse por defecto si la aplicación no requiere imperativamente el segundo formato.

4. Mientras el diseño del sistema de domótica lo permita, es preferible conectar los dispositivos periféricos directamente al servidor central ya que los servidores secundarios inducen retrasos asociados a su tiempo de actualización.

5. El tiempo de actualización de cada dispositivo debe ser asignado metódicamente, pues utilizar tiempos aleatorios o asignar tiempos cortos a dispositivos que no lo necesitan aumenta la concurrencia de peticiones a los servidores web, aumentando su costo.

BIBLIOGRAFÍA

1. Cisco Networking Academy. *Principios básicos de enrutamiento y switching CCNA1 V5.* Compilado por A. RAMÍREZ. Revisado por N. CONTADOR. Cisco, 2014. 598 p.

2. FIELDING, R. y otros. *Hypertext Transfer Protocol – HTTP/1.1* [en línea]. <https://tools.ietf.org/html/rfc2616>. [Consulta: 20 de marzo de 2019].

3. IPC-2221 TASK GROUP. *IPC-2221A, Generic Standard on Printed Board Design* [en línea]. <http://www.ipc.org/TPC/IPC-2221A.pdf>. [Consulta: 25 de agosto de 2019].

4. JUNESTRAND, Stefan, PASSARET, Xavier y VÁZQUEZ, Daniel. *Domótica y hogar digital.* España: Editorial Paraninfo, 2004. 228 p.

5. MADAKAM, Somayya, RAMASWAMY, R. y TRIPATHI, Siddharth. Internet of Things (IoT): A Literature Review. En: *Journal of Computer and Communications* [en línea]. ISSN 2327-5227. <http://dx.doi.org/10.4236/jcc.2015.35021>. [Consulta: 18 de febrero de 2019].

6. RODRIGUEZ, Alex. *RESTful Web Services: The Basics* [en línea]. <https://developer.ibm.com/articles/ws-restful/>. [Consulta: 25 de marzo de 2019].

7. SOLIS, Diego. *La privacidad de la información generada por dispositivos de domótica en el Internet de las Cosas*. Trabajo de graduación de Ingeniería en Ciencias y Sistemas. Facultad de Ingeniería, Universidad de San Carlos de Guatemala, 2016. 106 p.

8. Superintendencia de Telecomunicaciones de Guatemala. *Resolución SIT-DSI-349-2019* [en línea]. <https://sit.gob.gt/2019/09/11/resolucion-sit-dsi_349-2019>. [Consulta: 17 de octubre de 2019].

9. _____. *Resolución SIT-DSI-350-2019* [en línea]. <https://sit.gob.gt/2019/09/11/resolucion-sit-dsi-350-2019>. [Consulta: 17 de octubre de 2019].

APÉNDICES

Apéndice 1: **Código fuente del servidor**

```
#Bibliotecas utilizadas
from flask import Flask, request, render_template, jsonify, Response
import json

#Objetos y variables globales
app = Flask(__name__)
tree = "tree"
password = "key"
devices = ""

#Obtiene ramas de un archivo de texto
f = open("mysite/devices.txt","rb")
devices = json.loads(f.read())
f.close()

#Maneja el metodo POST para agregar ramas al arbol
@app.route('/devices', methods=['POST'])
def addDevices():
    auth = request.authorization
    if auth and auth.username == tree and auth.password == password:
        objects = request.json
        created = False
        for key in objects:
            if not devices.has_key(key):
                devices[key] = objects[key]
                created = True
        f = open("mysite/devices.txt","wb")
        f.write(json.dumps(devices))
        f.close()
        if created:
            return Response(
            '<h1>Created</h1> All or some of the devices were created', 201)
        else:
            return Response(
            '<h1>Accepted</h1> The devices were not created', 202)
    else:
        return Response(
        '<h1>Unauthorized</h1>', 401,
        {'WWW-Authenticate': 'Basic realm="Devices"'})
```

Continuación apéndice 1.

```
#Maneja el metodo PUT para modificar informacion de las ramas
@app.route('/devices/<subset>', methods=['PUT'])
@app.route('/devices', methods=['PUT'])
def updateDevices(subset=''):
    auth = request.authorization
    updated = False
    if auth and auth.username == tree and auth.password == password:
        objects = request.json
        for key in objects:
            if devices.has_key(key):
                if subset == '':
                    devices[key] = objects[key]
                else:
                    devices[key][str(subset)] = objects[key]
                updated = True
        f = open("mysite/devices.txt","wb")
        f.write(json.dumps(devices))
        f.close()
        if updated:
            return Response(
            '<h1>Created</h1> All or some of the devices were updated', 201)
        else:
            return Response(
            '<h1>Accepted</h1> The devices were not updated', 202)
    else:
        return Response(
        '<h1>Unauthorized</h1>', 401,
        {'WWW-Authenticate': 'Basic realm="Devices"'})

#Maneja el metodo DELETE para eliminar ramas del arbol
@app.route('/devices/<id>', methods=['DELETE'])
def deleteDevices(id):
    auth = request.authorization
    if auth and auth.username == tree and auth.password == password:
        deleted = False
        tmp = devices.copy()
        for key in tmp:
            if key.find(id)==0:
                deleted = True
                del(devices[key])
            else:
                continue
        f = open("mysite/devices.txt","wb")
        f.write(json.dumps(devices))
        f.close()
        if deleted:
            return Response(
            '<h1>Created</h1> All or some of the devices were deleted', 201)
        else:
            return Response(
            '<h1>Accepted</h1> The devices did not exist', 202)
    else:
        return Response(
        '<h1>Unauthorized</h1>', 401,
        {'WWW-Authenticate': 'Basic realm="Devices"'})
```

Continuación apéndice 1.

```
#Maneja el metodo GET para enviar informacion de las ramas
@app.route('/devices/<id>/<subset>', methods=['GET'])
@app.route('/devices/<id>', methods=['GET'])
def getDevices(id, subset=''):
    auth = request.authorization
    if auth and auth.username == tree and auth.password == password:
        body = {}
        for key in devices:
            if key.find(id) == 0:
                if subset == '':
                    body[key] = devices[key]
                else:
                    body[key] = devices[key][subset]
        return jsonify(body)
    else:
        return Response(
        '<h1>Unauthorized</h1>', 401,
        {'WWW-Authenticate': 'Basic realm="Devices"'})

#Maneja la interfaz de usuario
@app.route('/interface', methods=['GET','POST'])
def showInterface():
    auth = request.authorization
    if auth and auth.username == tree and auth.password == password:
        if request.method == 'POST':
            data = request.form
            key = data.keys()[0]
            if len(data)>1:
                if data.keys()[1] == "DELETE" and devices.has_key(key):
                    del(devices[key])
            else:
                if devices[key]['meta']['type']=='bool':
                    state = devices[key]['data']
                    devices[key]['data'] = not(state)
                else:
                    devices[key]['data'] = data.values()[0]
            f = open("mysite/devices.txt","wb")
            f.write(json.dumps(devices))
            f.close()
            return render_template('interface.html',objects=devices)
        else:
            return render_template('interface.html',objects=devices)
    else:
        return Response(
        '<h1>Unauthorized</h1>', 401,
        {'WWW-Authenticate': 'Basic realm="Devices"'})

#Muestra la informacion de la rama
@app.route('/interface/info/<id>', methods=['GET'])
def showDevice(id):
    return render_template('info.html', device=devices[id], id=id)
```

Fuente: elaboración propia, empleando código Python 2.7.

Apéndice 2: **Código fuente del dispositivo actuador**

```
//Bibliotecas utilizadas
#include <ESP8266WiFi.h>
#include <ArduinoJson.h>
#include "FS.h"
#include <ESP8266WebServer.h>
#include <WiFiClient.h>
#include <ESP8266HTTPClient.h>

//Interfaz de configuración
#include "interface.h"

//Definiciones
#define OUTPUT_PIN 3
#define CONFIG_PIN 1
#define CONFIG_TIME 5
#define CONFIG_FIELDS {"id","name","function","location","mode","type",
"refresh","ssid","password","tree","key","server","port"}

//Objetos globales
ESP8266WebServer configServer(80);
HTTPClient http;
WiFiClient client;

//Variables globales
String server = "";
String tree = "";
String key = "";
String id = "";
String device = "";
int refresh = 5;
bool deviceState = false;
bool communicationError = false;
```

Continuación apéndice 2.

```
//Estado de configuración
void setup() {
  initPeripherals();
  int option = getButtonTimes(CONFIG_PIN,CONFIG_TIME);
  if(option == 1){
    configureDevice();
  }else if(option == 3){
    restoreDevice();
    configureDevice();
  }
  connectDevice();
  initDevice();
  int code = postDevice();
  if (code == 202){
    int code = putMetadata();
  }
  if(code/100 != 2){
    communicationError = true;
  }
}

//Estado de ejecución
void loop() {
  if(!communicationError){
    String command = getData();
    executeCommand(command);
    delay(refresh*1000);
  }
}

//Inicialización de puertos de entrada y salida
void initPeripherals(){
  SPIFFS.begin();
  pinMode(CONFIG_PIN, INPUT_PULLUP);
  pinMode(OUTPUT_PIN, OUTPUT);
  digitalWrite(OUTPUT_PIN,LOW);
  pinMode(LED_BUILTIN, OUTPUT);
}

//Obtiene la cantidad de pulsaciones de un botón
int getButtonTimes(int button, int seconds){
  int t = millis();
  int times = 0;
  int actual = 0;
  int previous = 1;
  while((millis() - t) < seconds*1000){
    ESP.wdtFeed();
    for(int i = 0; i<1000; i++){
      actual = actual + digitalRead(button);
    }
```

Continuación apéndice 2.

```
    actual = actual/1000;
    if (actual - previous == -1){
      times = times + 1;
    }
    previous = actual;
  }
  return times;
}

//Restablece la configuración por defecto
void restoreDevice(){
  StaticJsonDocument<1024> json;
  getJsonFromFile(&json,"default.txt");
  saveJsonToFile(&json,"config.txt");
}

//Configuración de datos
void configureDevice(){
  setConfigNetwork();
  setConfigServer();
}

//Levanta la red de configuración
void setConfigNetwork(){
  String ssid = "IoTDevice";
  String pass = "1234567890";
  WiFi.mode(WIFI_AP);
  while(!WiFi.softAP(ssid,pass)){
    delay(500);
  }
}

//Levanta el servidor que muestra la interfaz en la dirección 192.168.4.1
void setConfigServer(){
  configServer.on("/",showConfigInterface);
  configServer.begin();
  while(configServer.arg("submit")!="Update"){
    configServer.handleClient();
  }
}

//Muestra la interfaz de configuración
void showConfigInterface(){
  String webpage = INTERFACE;
  configServer.send(200,"text/html",webpage);
  saveFormData();
}
```

Continuación apéndice 2.

```
//Almacena los datos obtenidos por la interfaz de configuración
void saveFormData(){
  String configFields[] = CONFIG_FIELDS;
  bool hasChanged = false;
  StaticJsonDocument<1024> json;
  getJsonFromFile(&json, "config.txt");
  for(int i=0; i<13; i++){
    String value = configServer.arg(configFields[i]);
    value.trim();
    if(value!=""){
      if(configFields[i]=="refresh" or configFields[i]=="port"){
        json[configFields[i]] = value.toInt();
      }else{
        json[configFields[i]] = value;
      }
      hasChanged = true;
    }
  }
  if (hasChanged){
    saveJsonToFile(&json,"config.txt");
  }
}

//Conecta el dispositivo a la red
void connectDevice(){
  StaticJsonDocument<1024> json;
  getJsonFromFile(&json,"config.txt");
  WiFi.mode(WIFI_STA);
  WiFi.begin(json["ssid"].as<String>(),json["password"].as<String>());
  while(WiFi.status() != WL_CONNECTED){
    delay(500);
  }
}

//Inicializa la imágen virtual de la rama
void initDevice(){
  if(WiFi.status() != WL_CONNECTED){
    connectDevice();
  }
  StaticJsonDocument<1024> json;
  getJsonFromFile(&json,"config.txt");
  server = json["server"].as<String>();
  id = json["id"].as<String>();
  refresh = json["refresh"];
  tree = json["tree"].as<String>();
  key = json["key"].as<String>();
  device = "{\""+id+"\":{\"meta\":{\"ip\":\""+WiFi.localIP().toString()+
  "\",\"mac\":\""+WiFi.macAddress()+"\",\"tree\":\""+tree+"\",\"name\":\""+
  json["name"].as<String>()+"\",\"function\":\""+json["function"].as<String>()+
  "\",\"location\":\""+json["location"].as<String>()+"\",\"type\":\""+
  json["type"].as<String>()+"\",\"mode\":\""+json["mode"].as<String>()+
  "\",\"refresh\":"+refresh+"},\"data\":"+deviceState+"}}";
}
```

Continuación apéndice 2.

```
//Agrega la rama al árbol
int postDevice(){
  if(WiFi.status() != WL_CONNECTED){
    connectDevice();
  }
  http.begin(client, server + "/devices");
  http.addHeader("Content-Type", "application/json");
  http.setAuthorization(tree.c_str(), key.c_str());
  int code = http.POST(device);
  http.end();
  return code;
}

//Modifica los metadatos de la rama
int putMetadata(){
  if(WiFi.status() != WL_CONNECTED){
    connectDevice();
  }
  String meta = "";
  StaticJsonDocument<1024> json;
  deserializeJson(json,device);
  json = json["meta"];
  serializeJson(json,meta);
  http.begin(client, server + "/devices/meta");
  http.addHeader("Content-Type", "application/json");
  http.setAuthorization(tree.c_str(), key.c_str());
  int code = http.PUT(meta);
  http.end();
  return code;
}

//Obtiene el estado de la rama
String getData(){
  if(WiFi.status() != WL_CONNECTED){
    connectDevice();
  }
  http.begin(client, server + "/devices/"+id+"/data");
  http.setAuthorization(tree.c_str(), key.c_str());
  int code = http.GET();
  if(code == 200){
    String text = http.getString();
    http.end();
    StaticJsonDocument<1024> json;
    deserializeJson(json,text);
    text = json[id].as<String>();
    if(text == ""){
      communicationError = true;
    }
    return text;
```

Continuación apéndice 2.

```
  }else{
    http.end();
    return "";
  }
}

//Exterioriza el estado de la rama
void executeCommand(String command){
  if (command!=""){
    if(command == "true"){
      deviceState = true;
      digitalWrite(OUTPUT_PIN, HIGH);
    }else{
      digitalWrite(OUTPUT_PIN, LOW);
      deviceState = false;
    }
  }
}

//Guarda un objeto json a un archivo de texto
void saveJsonToFile(StaticJsonDocument<1024> *json, String file){
  String text = "";
  serializeJsonPretty(*json,text);
  if (text != ""){
    File f = SPIFFS.open("/"+file,"w+");
    f.print(text);
    f.close();
  }
}

//Obtiene un objeto json de un archivo de texto
void getJsonFromFile(StaticJsonDocument<1024> *json, String file){
  String text = "";
  File f = SPIFFS.open("/"+file,"r");
  text = f.readString();
  deserializeJson(*json,text);
  f.close();
}
```

Fuente: elaboración propia, empleando código de Arduino (variante de C).

Apéndice 3: **Codificación HTML del archivo *interface.html* del actuador**

```
const char INTERFACE[] PROGMEM = R"=====(<!DOCTYPE html>
<html lang="en">
<head>
  <title>Configure device</title>
</head>
<body>
  <h1>Device configuration</h1>
  <form method="POST">
    <h3>Device data</h3>
    <p>
      ID:
      <input name="id" type="text">
    </p>
    <p>
      Name:
      <input name="name" type="text">
    </p>
    <p>
      Function:
      <input name="function" type="text">
    </p>
    <p>
      Location:
      <input name="location" type="text">
    </p>
    <p>
      Mode:
      <select name ="mode">
        <option value="">No change</option>
        <option value="INPUT">Input</option>
        <option value="OUTPUT">Output</option>
      </select>
    </p>
    <p>
      Type:
      <select name ="type">
        <option value="">No change</option>
        <option value="bool">boolean</option>
        <option value="number">number</option>
        <option value="text">string</option>
      </select>
    </p>
```

Continuación apéndice 3.

```
    <p>
      Refresh:
      <input type="number" name="refresh">
    </p>
    <h3>Network data</h3>
    <p>
      Network SSID:
      <input type="text" name="ssid">
    </p>
    <p>
      Password:
      <input type="password" name="password">
    </p>
    <p>
      Tree:
      <input type="text" name="tree">
    </p>
    <p>
      Key:
      <input type="password" name="key">
    </p>

    <p>
      Server:
      <input type="text" name="server">
    </p>
    <p>
      Port:
      <input type="number" name="port">
    </p>
    <input type="submit" name="submit" value="Update">
  </form>
</body>
</html>
)=====";
```

Fuente: elaboración propia, empleando lenguaje HTML.

Apéndice 4: **Código fuente de la función lambda**

```
from botocore.vendored import requests
import json

def lambda_handler(event, context):
    if event["request"]["type"] == "LaunchRequest":
        return on_launch()
    elif event["request"]["type"] == "IntentRequest":
        return on_intent(event["request"])
    elif event["request"]["type"] == "SessionEndedRequest":
        return on_end()

def on_launch():
    session_attributes = {}
    card_title = "Welcome"
    speech_output = "Welcome to the voice interface for your devices "
    reprompt_text = ""
    should_end_session = False
    return build_response(session_attributes, build_speechlet(
        card_title, speech_output, reprompt_text, should_end_session))

def on_intent(intent_request):
    intent = intent_request["intent"]
    intent_name = intent_request["intent"]["name"]
    if intent_name == "LightsOn":
        return turn_on_lights()
    elif intent_name == "LightsOff":
        return turn_off_lights()

def on_end():
    session_attributes = {}
    card_title = "Goodbye"
    speech_output = ""
    reprompt_text = "Have a good day"
    should_end_session = True
    return build_response(session_attributes, build_speechlet(
        card_title, speech_output, reprompt_text, should_end_session))


def turn_on_lights():
    session_attributes = {}
    card_title = "LightsOn"
    reprompt_text = ""
    should_end_session = False
    speech_output = "Turning on the lights"
    r = requests.put('http://user.pythonanywhere.com/devices/data', auth=('tree', 'key'),
                     json={"2":True})
    return build_response(session_attributes, build_speechlet(
        card_title, speech_output, reprompt_text, should_end_session))
```

Continuación apéndice 4.

```
def turn_off_lights():
    session_attributes = {}
    card_title = "LightsOff"
    reprompt_text = ""
    should_end_session = False
    speech_output = "Turning off the lights"
    r = requests.put('http://user.pythonanywhere.com/devices/data', auth=('tree', 'key'),
                     json={"2":False})
    return build_response(session_attributes, build_speechlet(
        card_title, speech_output, reprompt_text, should_end_session))

def build_speechlet(title, output, reprompt_text, should_end_session):
    return {
        "outputSpeech": {
            "type": "PlainText",
            "text": output
        },
        "card": {
            "type": "Simple",
            "title": title,
            "content": output
        },
        "reprompt": {
            "outputSpeech": {
                "type": "PlainText",
                "text": reprompt_text
            }
        },
        "shouldEndSession": should_end_session
    }

def build_response(session_attributes, speechlet_response):
    return {
        "version": "1.0",
        "sessionAttributes": session_attributes,
        "response": speechlet_response
    }
```

Fuente: elaboración propia, empleando lenguaje Python 2.7.

ANEXOS

Anexo 1: **Rangos de frecuencia de operación de dispositivos en una RIABAP**

Rango de frecuencia	Dimensional
13,553 – 13,567	kHz
115.0 – 135.0	kHz
325.0 – 330.0	KHz
6,765.0 – 6,795	KHz
40.66 – 40.70	MHz
312.0 – 315.0	MHz
433.05 – 434.79	MHz
902.0 – 928.0	MHz
2,400.0 – 2,500.0	MHz
5,725.0 – 5,875.0	MHz
10.5 – 10.6	GHz
17.1 – 17.3	GHz
24.05 – 26.65	GHz
61.0 – 61.5	GHz
76.0 – 81.0	GHz
122.0 – 123.0	GHz
244.0 – 246.0	GHz

Fuente: Superintendencia de Telecomunicaciones de Guatemala. Resolución SIT-DSI-349-2019. https://sit.gob.gt/2019/09/11/resolucion-sit-dsi_349-2019. Consulta: 17 de octubre de 2019.

Anexo 2: **Cargos administrativos por inscripción, actualización y modificación de RIABAP**

No.	SERVICIO	COSTO ADMINISTRATIVO
1	Inscripción	Q. 10,000.00
2	Actualización	Q. 3,000.00
3	Modificación	Q. 4,600.00

Fuente: Superintendencia de Telecomunicaciones de Guatemala. Resolución SIT-DSI-350-2019. https://sit.gob.gt/2019/09/11/resolucion-sit-dsi-350-2019. Consulta: 17 de octubre de 2019.

Anexo 3: **Lista completa de códigos de estado del protocolo HTTP**

- Información:
    - 100 *Continue*: indica que el cliente puede continuar con su petición. Informa que la parte inicial de la petición se ha recibido y no ha sido rechazada por el servidor.
    - 101 *Switching Protocols*: indica que el servidor entiende y está dispuesto a cumplir la solicitud del cliente de cambiar la versión del protocolo.
- Éxito:
    - 200 *OK*: indica que la petición fue exitosa. La respuesta dependerá del método de petición.

Continuación anexo 3.

- 201 *Created*: indica que se ha creado un recurso con éxito y puede ser referido con la URI en la respuesta.

- 202 *Accepted*: se utiliza cuando el servido ha aceptado con éxito una petición pero aún necesita procesarse.

- 203 *Non-Authoritative Information*: significa que la petición se ha completado con éxito pero no se ha obtenido de la fuente original. Esto puede indicar la presencia de un proxy.

- 204 *No Content*: el servidor ha cumplido con éxito la petición pero no necesita enviar ninguna información como respuesta.

- 205 *Reset Content*: el servidor ha completado la petición pero el cliente necesita reiniciar la vista del documento que envió la solicitud.

- 206 *Partial Content*: el servidor ha completado la petición parcial de contenido por parte del cliente.

- Redirección:

  - 300 *Multiple Choices*: el servidor indica que el recurso solicitado tiene varias representaciones y envía una lista de las ubicaciones de estas para que el cliente pueda seleccionar la que desea.

Continuación anexo 3.

- 301 *Moved Permanently*: indica que el recurso ha sido movido a otra ubicación y cualquier futura petición del mismo deberá realizarse con la ubicación indicada.

- 302 *Found*: indica que el recurso ha sido movido de forma temporal a otra ubicación por lo que futuras peticiones pueden utilizar la misma ubicación.

- 303 *See Other*: significa que la respuesta de la petición se puede obtener con otra URI y debería utilizarse un método GET hacia esa ubicación.

- 304 *Not Modified*: se envía como respuesta a un GET condicional cuando el recurso es permitido, pero no hay ninguna modificación según la condición.

- 305 *Use Proxy*: indica que la petición debe realizarse a través de un proxy indicado.

- 306 (*Unused*): es un código que no se utiliza en HTTP/1.1 y está reservado.

- 307 *Temporary Redirect*: indica que el recurso está temporalmente en otra URI indicada en la respuesta.

Continuación anexo 3.

- Error de cliente:

    - 400 *Bad Request*: indica que la petición no fue entendida por el servidor y que puede hacerse de nuevo con modificaciones.

    - 401 *Unauthorized*: este código indica que el acceso al recurso requiere autenticación, por tanto el cliente puede repetir la petición con una cabecera de autenticación.

    - 402 *Payment Required*: este código fue reservado para uso en el futuro.

    - 403 *Forbidden*: indica que la petición fue entendida por el servidor, pero no se completará aunque se envíe autenticación.

    - 404 *Not Found*: significa que el servidor no ha encontrado el recurso en la URI especificada y no hay referencias de alguna otra ubicación.

    - 405 *Method Not Allowed*: indica que el método utilizado para realizar la petición no es permitido por el recurso y debe incluir una cabecera *Allow* con los métodos permitidos.

    - 406 *Not Acceptable*: este código se utiliza cuando la respuesta genera un tipo de contenido que el cliente no puede aceptar según indicó en la petición.

Continuación anexo 3.

- 407 *Proxy Authentication Required*: indica que el cliente primero debe autenticarse con el proxy.

- 408 *Request Timeout*: indica que cliente no realizó ninguna petición en el tiempo que el servidor podía esperar. El cliente puede volver a hacer la petición después sin modificaciones.
- 409 *Conflict*: indica que ha habido algún conflicto con el recurso solicitado. Este código se usa cuando se espera que el cliente pueda realizar alguna modificación para resolver el conflicto.

- 410 *Gone*: significa que el recurso ya no está disponible y no se conoce ninguna otra ubicación.

- 411 *Length Required*: indica que el servidor no puede aceptar una petición sin una longitud definida de contenido.

- 412 *Precondition Failed*: indica que una condición probada falló.

- 413 *Request Entity Too Large*: significa que la petición es más larga de lo que el servidor puede procesar.

- 414 *Request-URI Too Long*: indica que la URI es más larga de lo que el servidor puede entender o procesar.

- 415 *Unsupported Media Type*: indica que el servidor no puede completar la petición porque la respuesta está en un formato que no soporta el cliente.

Continuación anexo 3.

- 416 *Requested Range Not Satisfiable*: se envía como respuesta cuando un rango especificado con una cabecera por un cliente no se puede obtener de un recurso.

- 417 *Expectation Failed*: indica que el servidor no pudo completar la petición de expectativa.

- Error de servidor:

  - 500 *Internal Server Error*: indica que el servidor ha encontrado una condición inesperada que provocó que no se pudiera satisfacer la petición.

  - 501 *Not Implemented*: indica que el servidor no soporta la funcionalidad requerida para completar la petición.

  - 502 *Bad Gateway*: el servidor que actúa como puerta de enlace o proxy recibió una respuesta inválida al tratar de completar la petición.

  - 503 *Service Unavailable*: indica que el servidor no ha podido completar la petición debido a que ha sido sobrecargado o se encuentra en mantenimiento.

  - 504 *Gateway Timeout*: el servidor que actúa como puerta de enlace o proxy no recibió una respuesta en su tiempo de espera.

Continuación anexo 3.

- 505 HTTP *Version Not Supported*: indica que el servidor no soporta la versión de HTTP indicada en la petición.

Fuente: FIELDING, R., et. al. Hypertext Transfer Protocol – HTTP/1.1. https://tools.ietf.org/html/rfc2616. Consulta: 20 de marzo de 2019