INTERNET DE LAS COSAS (IoT) CON PYTHON Y Raspberry PI, (Monitor Remoto)

Prof. César Rodríguez

Índice general

Capítulo 1

INTRODUCCIóN

El presente trabajo se concentra en el desarrollo de un sistema de Internet de las Cosas (IoT) utilizando Python y Raspberry Pi, cubriendo desde la configuración básica del hardware hasta la implementación de servidores seguros y de gestión de usuarios. Con este sistema se busca construir una plataforma IoT **escalable** donde múltiples usuarios puedan monitorear y controlar sus dispositivos de forma remota y segura en tiempo real.

A continuación se presenta un esquema de como está estructurado el sistema:

- 1. **Hardware y Configuración Inicial:**
  - La Raspberry Pi es el dispositivo central para el desarrollo de IoT, requiriendo de una tarjeta SD de al menos 8GB para el sistema operativo (SO). Se configura la Raspberry Pi siguiendo las instrucciones oficiales y se habilita SSH para el acceso remoto, especialmente en la Raspberry Pi 3 donde está deshabilitado por defecto.
  - En el trabajo se detalla el uso de:
    - Sensores PIR (infrarrojos pasivos) para detectar movimiento. Un sensor PIR, como el RE200BL, detecta el calor emitido por humanos o animales en forma de radiación infrarroja, sin usar energía propia para la detección. Tiene pines para tierra, VCC (5 voltios) y una salida que da un nivel lógico alto al detectar un objeto.
    - Potenciómetros para ajustar la sensibilidad y el tiempo que la señal de entrada permanece alta. Los modos de disparo pueden ser 'no repetible' (la salida vuelve a bajo después de un retraso) o 'repetible' (la salida permanece alta mientras el objeto está presente).
    - Un zumbador (buzzer) con tres pines (VCC 5V, tierra y señal). Se controla generando una onda cuadrada, alternando entre señal alta y baja con pausas de milisegundos para producir sonido.
- 2. **Desarrollo de Software con Python y Flask:**
  - **Python** es el lenguaje de programación principal utilizado para interactuar con los sensores y controlar los actuadores conectados a la Raspberry Pi.
  - **Flask** se emplea para crear un servidor web HTTP básico en la Raspberry Pi. Este servidor permite a los usuarios acceder a una página web desde sus navegadores (dentro de la misma red Wi-Fi local inicialmente) para monitorear el estado del sensor y controlar actuadores, como desactivar una alarma.
  - El servidor web envía solicitudes 'keepalive' periódicas (cada cinco segundos) desde el navegador del usuario a la Raspberry Pi para asegurar que la conexión esté activa y enviar actualizaciones del sensor al usuario en tiempo real. También se añaden botones en la página web para que los usuarios puedan interactuar con los actuadores.
  - Las funcionalidades del servidor incluyen la gestión de usuarios, diferenciando entre usuarios administradores y no administradores. Se muestra cómo desarrollar el código HTML ('index.html') para crear un panel de control donde los administradores puedan ver la lista de usuarios en línea y otorgarles permisos de lectura y escritura.
  - El servidor gestiona los IDs de usuario y la lista de usuarios en línea, convirtiendo los permisos de lectura/escritura en estados 'checked' o 'unchecked' para la interfaz web. Se utiliza Jinja

para integrar esta lógica en las plantillas HTML, permitiendo bucles para mostrar múltiples usuarios y condicionales para controlar la visibilidad del panel de administración.

- Se implementan **endpoints** en la aplicación Flask para recibir solicitudes desde el código JavaScript del cliente, por ejemplo, para conceder permisos a un usuario específico. Esto incluye almacenar los permisos en la base de datos y utilizar un servidor PubNub para otorgar acceso en tiempo real.

- 3. **Protocolos de Comunicación IoT:**
  - Se exploran dos protocolos claves: **WebSockets** y **MQTT (Message Queuing Telemetry Transport)**.
  - **WebSockets** permiten sesiones de comunicación interactivas y persistentes sobre una única conexión TCP, permitiendo que los clientes reciban actualizaciones en tiempo real sin tener que 'preguntar' constantemente al servidor (polling). Son **bidireccionales** y **dúplex completo**, lo que significa que ambas partes pueden enviar mensajes de forma independiente, y son adoptados por su baja latencia.
  - **MQTT** es un protocolo de transferencia de mensajes ligero y eficiente en ancho de banda, ideal para comunicación máquina a máquina e IoT. Utiliza un modelo de publicación/suscripción, donde los dispositivos publican datos en 'tópicos' y otros dispositivos suscritos a esos tópicos reciben la información. Funciona sobre **TCP/IP** y puede ser utilizado sobre la capa WebSocket. Se explica que MQTT es un servicio de entrega que se empaqueta dentro de un 'sobre' WebSocket, que a su vez se envuelve en un 'sobre' TCP/IP. El punto central de comunicación es el **broker** MQTT, que distribuye los mensajes basándose en los tópicos. Un inconveniente es que si el broker falla, toda la comunicación se interrumpe.
  - Se reconstruye un proyecto para usar **PubNub** como protocolo de comunicación principal en lugar de **AJAX** long-polling, lo que también desplegar el servidor IoT en la nube de **AWS**.

- 4. **Seguridad y Despliegue en la Nube:**
  - La seguridad es un aspecto crucial, cubriendo temas como **SSL/TLS**, cifrado, firmas digitales y el uso de HTTPS.
  - Para asegurar un dominio personalizado, se utiliza la autoridad de certificación **Let's Encrypt**, que es gratuita, automatizada y de código abierto. El proceso implica instalar el software **certbot** en el servidor y configurarlo para Apache, lo que permite la obtención y gestión de certificados SSL/TLS.
  - Es fundamental configurar reglas de seguridad de entrada (inbound security rules) en el servidor remoto (por ejemplo, en AWS EC2) para permitir el tráfico HTTPS en el puerto 443. Una vez configurado, el servidor redirige el tráfico HTTP a HTTPS, mostrando un 'candado verde' que indica una conexión segura.
  - Los certificados de **Let's Encrypt** son válidos por tres meses y deben renovarse.
  - La comunicación cifrada de extremo a extremo es un objetivo clave una vez que el sitio web está protegido con SSL.
  - Se implementa una funcionalidad de inicio de sesión de usuario seguro y el almacenamiento de detalles del usuario en una base de datos integrada.
  - Se busca una manera segura para que los usuarios y dispositivos IoT se conecten al servidor, utilizando la función 'Access Manager' de PubNub para que los administradores puedan conceder permisos de lectura y escritura a usuarios no administradores y a dispositivos en tiempo real.

En resumen, en este trabajo detalla cómo Python y Raspberry Pi forman la base para construir un sistema IoT robusto, que incluye la interacción con el hardware, el desarrollo de un servidor web interactivo, la implementación de protocolos de comunicación avanzados como WebSockets y MQTT, y un fuerte énfasis en la seguridad a través de SSL/TLS y la gestión de accesos de usuarios, con el objetivo de crear una plataforma IoT escalable y en tiempo real con capacidades de despliegue en la nube.

1.1. Estructura por Etapas

- **Etapa 1: Introducción a IoT**: Se enfoca en **entender qué es el Internet de las Cosas y sus principales componentes**, como dispositivos inteligentes, sensores y actuadores. También se tendrá como objetivo entender **cuatro importantes modelos de comunicación del Internet de las Cosas**.
- **Etapa 2: Primer Proyecto IoT**: Se **ejecutarán labores prácticas** desarrollando un proyecto IoT simple con sensores mostrando datos en una aplicación web, utilizando la técnica **AJAX** para la comunicación. Aquí se construirá un **dispositivo IoT con sensor PIR (Piezoelectric Infra Red)** que detecta movimiento, activa una alarma y envía alertas, permitiendo también la desactivación de la alarma.
- **Etapa 3: Seguridad y Protocolos de Comunicación**: Después de analizar ventajas y desventajas del proyecto de la etapa 2, se estudiará en profundidad los **protocolos de comunicación en tiempo real y ligeros** para IoT, como **MQTT, WebSockets**, y se hará una **demostración práctica con PubNub**. También se cubrirá **seguridad en Internet y criptografía**, incluyendo SSL/TLS y protocolos HTTP.
- **Etapa 4: Reconstrucción del Proyecto con PubNub y AWS**: Se **reconstruirá el proyecto anterior utilizando PubNub como el principal protocolo de comunicación** en lugar de **AJAX**. Además, se aprenderá a **desplegar el servidor IoT en la nube de AWS**.
- **Etapa 5: Implementación de Seguridad**: Se centrará en el desarrollo de las terminologías de seguridad estudiadas en la etapa 3. Esto incluye la adquisición de un **nombre de dominio personalizado y su aseguramiento con certificados SSL/TLS**, así como la implementación de una **funcionalidad de inicio de sesión de usuario segura** y el almacenamiento de detalles de usuario en una base de datos integrada. Se gestionarán las **reglas de seguridad de entrada para HTTPS en el servidor remoto de AWS**.
- **Etapa 6: Conexión Segura de Usuarios y Dispositivos**: Se implementará una forma segura para que usuarios y dispositivos IoT se conecten al servidor. Se utilizará la **funcionalidad de administrador de acceso de PubNub** para que los usuarios administradores puedan **otorgar acceso de lectura y escritura en tiempo real** a usuarios no administradores y dispositivos.
- **Etapa 7: Proyecto Final: Sistema de Monitoreo Remoto**: Se añadirán más sensores y actuadores para construir un **sistema de monitoreo salud/batería**, familiarizándose con convertidores digitales, interfaz periférica serial y más.

1.2. Resultados

- Al finalizar el proyecto, se **entenderá lo que se necesita para construir una solución IoT propia e integral**, desde la simplicidad a nivel de dispositivo hasta la complejidad de la infraestructura a nivel de la nube. Podrán expandir el proyecto añadiendo más dispositivos y funcionalidades según surjan necesidades.

En esencia, en este proyecto se estudian desde los fundamentos teóricos y el desarrollo local de dispositivos, hasta la implementación de una plataforma IoT segura, escalable y en la nube (servidor remoto), capacitandonos para abordar problemas del mundo real con un conjunto robusto de **tecnologías** y **conocimientos**.

1.3. Guía Prática

Se pretende con este proyecto (**IoT con Python y Raspberry Pi**) construir una **guía práctica y completa** para usuarios principiantes e intermedios en el campo del Internet de las Cosas (IoT), con el objetivo de que se **entiendan y resuelvan problemas del mundo real de IoT**.

En un contexto más amplio de esta **introducción**, esta guía práctica se manifiesta a través de varias secciones y proyectos claves:

- **Desarrollo de proyectos IoT**:
  - La etapa 2 se centra en la **puesta en práctica** con el **desarrollo de un proyecto simple de IoT utilizando sensores que muestran datos en una aplicación web**, empleando la técnica AJAX para la comunicación cliente-servidor.
  - Un ejemplo detallado de este enfoque es el **proyecto de detector de movimiento**, donde se conecta un sensor PIR y un zumbador a una Raspberry Pi, y se escribe código Python para detectar movimiento y controlar el zumbador, además de configurar un servidor web HTTP básico.
  - La etapa 4 reconstruye el proyecto anterior para utilizar PubNub como protocolo de comunicación principal y enseña **cómo desplegar un servidor IoT en la nube de AWS**.
  - La etapa 7 culmina con un **último proyecto llamado sistema de monitoreo salud/batería**. En este proyecto, se añadiran los sensores y actuadores necesarios para monitoria tales variable, y los participantes se familiarizan con convertidores digitales, interfaz periférica serial y más. El producto final es un **tablero de control que muestra gráficos visuales con datos de sensores y actuadores en tiempo real**.
- **Desarrollo de servidor y seguridad**:
  - El proyecto proporciona un equilibrio entre el **desarrollo de dispositivos en Raspberry Pi y el desarrollo de servidores locales y remotos usando Python**.
  - La etapa 5 se enfoca en el **desarrollo de conceptos de seguridad**, incluyendo la **obtención y protección de un dominio personalizado con certificados SSL/TLS de Let's Encrypt**. Esto implica instalar software de terceros como **Certbot** y configurar reglas de seguridad de entrada para HTTPS.
  - También se implementará una **funcionalidad segura de inicio de sesión de usuario** y el almacenamiento de detalles de usuario en una base de datos integrada.
  - La etapa 6 implementa una **forma segura para que los usuarios y dispositivos IoT se conecten al servidor IoT**. También utilizara la funcionalidad de **administrador de acceso de PubNub** para que los usuarios administradores puedan **otorgar acceso de lectura y escritura en tiempo real a usuarios no administradores y dispositivos**. Esto se visualizara en un **tablero de control para administradores** que listara los usuarios en línea y ofrecera botones para conceder permisos.

1.4. Plataforma IoT en la Nube

Se describe la **Plataforma IoT en la Nube** como un componente esencial para construir una **solución integral de IoT**, que abarca desde la simplicidad a nivel de dispositivo hasta la complejidad de la infraestructura a nivel de nube. En el contexto del **intrucción del proyecto**, se aprenderá a desarrollar una **plataforma en la nube segura (secure cloud platform)** donde múltiples usuarios pueden iniciar sesión, controlar y monitorear sus dispositivos autorizados en tiempo real. El proyecto se equilibra entre el **desarrollo de dispositivos en Raspberry Pi y el desarrollo de servidores locales y remotos utilizando Python**. La Plataforma IoT en la Nube se construira y se utilizará a lo largo de varias etapas:

- **Despliegue en la nube de AWS**:
  - La etapa 4 se centra en la **reconstrucción del proyecto** inicial para utilizar PubNub como protocolo de comunicación principal y se enseñara **cómo desplegar un servidor IoT en la nube de AWS (deploy IoT server into AWS cloud)**.

- La configuración de seguridad para este servidor en la nube es crucial. Por ejemplo, al asegurar un dominio personalizado con certificados SSL/TLS, se destaca la necesidad de **asignar reglas de seguridad de entrada para HTTPS en el servidor remoto de AWS** para permitir la conexión en el puerto 443.

- **Protocolos de comunicación y seguridad en la nube**:
  - Las etapas 3 y 4 abordan en profundidad los **protocolos de comunicación en tiempo real y ligeros** como MQTT y WebSockets, y su implementación con **PubNub**. PubNub es descrito como un servicio de entrega sobre WebSockets, lo que sugiere su rol como una capa de comunicación gestionada en la nube para IoT.
  - La etapa 5 se dedica al **desarrollo de conceptos de seguridad** en la plataforma en la nube, incluyendo:
    - La obtención y **protección de un dominio personalizado con certificados SSL/TLS de Let's Encrypt**. Esto implica la instalación de software de terceros como Certbot y la configuración de las reglas de seguridad de entrada para HTTPS en el servidor remoto.
    - La implementación de una **funcionalidad segura de inicio de sesión de usuario** y el almacenamiento de los detalles del usuario en una base de datos integrada.
  - La etapa 6 implementa una **forma segura para que los usuarios y dispositivos IoT se conecten al servidor IoT**. También utiliza la funcionalidad de **administrador de acceso de PubNub (PubNub access manager functionality)** para que los usuarios administradores puedan otorgar acceso de lectura y escritura en tiempo real a usuarios no administradores y dispositivos. Esto se visualizara en un **tablero de control para administradores** que lista a los usuarios en línea y permitira conceder permisos.

El objetivo final es que, al finalizar el proyecto, los participantes sepan **lo que se necesita para construir su propia solución IoT integral**, abarcando desde la simplicidad a nivel de dispositivo hasta la complejidad de la infraestructura a nivel de nube.

1.5. Enfoque Basado en la Resolución de Problemas

Se indica que en el proyecto 'IoT con Python y Raspberry Pi ' se adopta un **enfoque basado en la resolución de problemas**, particularmente centrándose en **problemas de IoT del mundo real**. Este enfoque se enmarca en una **guía práctica completa** destinada tanto a principiantes como a usuarios intermedios. En el contexto más amplio del **Resumen del Curso**, este enfoque se desarrolla de la siguiente manera:

- **Comprensión y aplicación**: El proyecto sigue un **enfoque basado en la reconstrucción completa**. Esto significa que ayuda a los usuarios del sistema a **entender el por qué antes del cómo y qué**. Esta metodología sugiere que se busca una comprensión profunda de los desafíos antes de abordar su implementación técnica.

- **Proyectos prácticos para problemas del mundo real**:
  - La etapa 2 del proyecto se enfoca en **ensuciarse las manos** con el desarrollo de un **proyecto simple de IoT utilizando sensores que muestran datos en una aplicación web**.
  - Un ejemplo concreto de esto es el **proyecto detector de movimiento**, que busca crear un dispositivo que detecta movimiento, activa una alarma y envía alertas, además de permitir la desactivación de la alarma por parte del usuario.
  - La etapa 7 presenta un **último proyecto llamado sistema de monitoreo**. En este proyecto, se añaden más sensores y actuadores acordes con lo que se quiera monitonear para **construir algo significativo para un caso de uso del mundo real**. El producto final es un tablero de control que muestra gráficos visuales con datos de sensores y actuadores en tiempo real.

- **Desarrollo de soluciones integrales**: El proyecto busca un equilibrio entre el desarrollo de dispositivos en Raspberry Pi y el desarrollo de servidores locales y remotos usando Python. El objetivo final de este enfoque basado en la resolución de problemas es que, al finalizar el proyecto, los participantes sepan **lo que se necesita para construir su propia solución IoT integral**,

abarcando desde la simplicidad a nivel de dispositivo hasta la complejidad de la infraestructura a nivel de nube. Esto incluye el desarrollo de una plataforma en la nube segura donde múltiples usuarios pueden iniciar sesión, controlar y monitorear sus dispositivos autorizados en tiempo real.

En resumen, el 'Enfoque Basado en Resolución de Problemas' es central en el proyecto, guiando a losdesarrolladores del sistema a través de la comprensión de los desafíos de IoT y proporcionándoles las herramientas y la experiencia práctica para construir soluciones funcionales y seguras para escenarios del mundo real.

1.6. Tecnologías Claves

1.6.1. Python

Python [1] es una de las **tecnologías principales** en el proyecto 'Internet de las Cosas con Python y Raspberry Pi'. Su rol es fundamental y se equilibra entre el desarrollo a nivel de dispositivo y el desarrollo de servidores. En el contexto más amplio de las **Tecnologías Clave** del proyecto, Python se utiliza para:

- **Programación de Dispositivos IoT con Raspberry Pi**:
  - Los desarrolladores aprenderán a escribir **código Python básico** para detectar señales de sensores (como el sensor PIR para movimiento) y controlar actuadores (como el zumbador). Este código permite la detección de movimiento y la activación de alarmas.
  - También se utilizara Python para implementar la lógica de comunicación bidireccional, permitiendo a los usuarios interactuar con el dispositivo, como desactivar una alarma.
- **Desarrollo de Servidores Locales y Remotos**:
  - Python se empleara para crear un **servidor web HTTP básico con Flask** en la Raspberry Pi, que se ejecutara en la red Wi-Fi local.
  - El proyecto mantiene un **equilibrio completo entre el desarrollo de dispositivos en Raspberry Pi y el desarrollo de servidores locales y remotos usando Python**. Esto incluye el despliegue de un servidor IoT en la nube de AWS.
- **Funcionalidades del Servidor IoT (con Flask)**:
  - Dentro de la aplicación Flask, Python se utilizara para añadir funcionalidades esenciales como la provisión de detalles adicionales como el ID de usuario y la lista de usuarios en línea a la página web.
  - Se implementara lógica en Python para **poblar variables con registros de usuarios en línea**, incluyendo nombres, IDs de usuario y estados de acceso de lectura y escritura.
  - Python es fundamental para **crear *endpoints*** en la aplicación Flask para recibir solicitudes, como las de concesión de permisos.
  - También se usara para manejar la **lógica de permisos de usuario**, almacenando permisos de lectura y escritura en la base de datos y llamando al servidor de PubNub para conceder acceso específico a los usuarios.
  - Se asegura que los paneles de control de acceso solo sean visibles para los usuarios administradores, utilizando la ID de usuario para añadir sentencias condicionales en el código HTML a través de las plantillas Jinja, gestionadas por Python en el servidor.

En resumen, Python es una tecnología central que une tanto la programación del hardware (Raspberry Pi) como la creación de la infraestructura de software (servidores, lógica de negocio, seguridad y gestión de usuarios) que forman la solución integral de IoT del proyecto.

1.6.2. Raspberry PI

Se establece claramente que **Raspberry Pi** [2] es una de las **tecnologías principales** abordadas en el proyecto 'Internet de las Cosas con Python y Raspberry Pi '. Su papel es fundamental para el **desarrollo de dispositivos IoT** y para albergar servidores locales. En el contexto más amplio de las **Tecnologías Clave**, la Raspberry Pi se utilizara para:

- **Plataforma de Hardware para Dispositivos IoT**:
  - El proyecto mantiene un **equilibrio completo entre el desarrollo de dispositivos en Raspberry Pi y el desarrollo de servidores locales y remotos usando Python**. Esto subraya su importancia como la plataforma física sobre la que se construyen las soluciones IoT.
  - En la etapa 2, los desarrolladores del proyecto 'se ensuciarán las manos' con un **proyecto simple de IoT** utilizando sensores y mostrando datos en una aplicación web, con la Raspberry Pi como el cerebro del dispositivo.
- **Implementación de Proyectos Prácticos**:
  - Para el **proyecto detector de movimiento**, la Raspberry Pi es el dispositivo central. Se conecta con el **sensor PIR (Passive Infrared)** para detectar movimiento y un **zumbador (buzzer)** como actuador para activar una alarma.
  - Los desarrolladores escribirán código Python para detectar señales y controlar el zumbador, así como para añadir funcionalidad de comunicación bidireccional, permitiendo a los usuarios desactivar la alarma.
  - En la etapa 7, la Raspberry Pi continuará siendo la plataforma para el **sistema de monitoreo atmosférico**, un proyecto que implica la adición de más sensores y actuadores para construir algo significativo para un caso de uso del mundo real.
- **Alojamiento de Servidores Locales**:
  - La Raspberry Pi se utilizara para ejecutar un **servidor web HTTP básico con Flask** en la red Wi-Fi local. Esto permitira que los usuarios, conectados a la misma red, accedan a una página web desde sus navegadores y reciban actualizaciones en vivo del sensor.
  - También se añadira un botón en la página web para que los usuarios puedan controlar los actuadores, como desactivar el zumbador.
- **Configuración y Preparación del Entorno**:
  - Para empezar, se sugiere a los desarrolladores configurar sus Raspberry Pi siguiendo las instrucciones oficiales, lo que incluye el uso de una tarjeta SD de al menos 8GB para el sistema operativo.
  - Se menciona la opción de utilizar un monitor HDMI o realizar una **conexión de escritorio remoto** a la Raspberry Pi.
  - Para la Raspberry Pi 3, se enfatiza la necesidad de **habilitar SSH** a través de la terminal usando `sudo raspi_config`.

En resumen, la Raspberry Pi es la **piedra angular del hardware** en este proyecto, permitiendo a los estudiantes interactuar directamente con el mundo físico a través de sensores y actuadores, y también sirve como una plataforma de bajo costo para desarrollar y probar servidores IoT a nivel local, antes de pasar a despliegues en la nube.

1.6.3. FLASK

Se indica que **Flask** [3] es una de las **tecnologías principales** utilizadas en el proyecto 'Internet de las Cosas con Python y Raspberry Pi'. Su rol es crucial para el **desarrollo de servidores IoT**, tanto a nivel local como remoto. En un contexto más amplio de las **Tecnologías Clave** del proyecto, Flask se empleara para:

- **Desarrollo de Servidores Web HTTP Locales**:
  - Los desarrolladores aprenderán a escribir un **servidor web HTTP básico en Python Flask en Raspberry Pi**. Este servidor se ejecutará en la red Wi-Fi local, permitiendo que los usuarios de la misma red accedan a una página web y reciban actualizaciones en vivo de los sensores.
  - También se añadira un botón en la página web para que los usuarios puedan controlar los actuadores, como desactivar un zumbador, comunicándose con el servidor Flask.

- **Desarrollo de Servidores Remotos y en la Nube**:
  - El proyecto mantiene un **equilibrio completo entre el desarrollo de dispositivos en Raspberry Pi y el desarrollo de servidores locales y remotos usando Python**, donde Flask juega un papel clave en la parte del servidor. Esto incluye el despliegue de servidores IoT en la nube de AWS.
- **Implementación de la Lógica del Servidor IoT**:
  - **Provisión de Detalles al Cliente**: Flask se utilizara para pasar detalles adicionales como el ID de usuario y la lista de usuarios en línea a la página web del cliente.
  - **Población de Datos de Usuarios en Línea**: Se implementa lógica en Flask (usando Python) para crear un mapa `online_user_records` que contenga el nombre del usuario, el ID de usuario y los estados de permisos de lectura y escritura (representados como 'checked' o 'unchecked' para el HTML). Este mapa es luego enviado a las plantillas Jinja en `index.html` para poblar dinámicamente la lista de usuarios en línea.
  - **Gestión de Permisos de Usuario**:
    - Flask recibira solicitudes de 'concesión' de permisos (grant) desde el código JavaScript del cliente a través de *endpoints* específicos.
    - La aplicación Flask verifica si la solicitud proviene de un usuario administrador antes de procesar la solicitud.
    - Se utilizara Flask para **almacenar los permisos de lectura y escritura del usuario en la base de datos y luego llamar al servidor de PubNub para conceder acceso de lectura y escritura específico a este usuario**.
  - **Control de Visibilidad del Panel de Acceso**: La aplicación Flask, al enviar el ID de usuario al cliente, permite que las plantillas Jinja utilicen sentencias condicionales (`if`) para que el panel de control de acceso solo sea visible para los usuarios administradores.

En síntesis, Flask es una **pieza fundamental en la arquitectura del servidor IoT** presentada en el proyecto, facilitando la comunicación entre los dispositivos, los usuarios y la infraestructura de la nube. Permite la creación de la interfaz web, la gestión de datos de usuarios, la implementación de la lógica de permisos y la interacción bidireccional, contribuyendo a la construcción de una **solución IoT segura y escalable**.

1.6.4. AWS

AWS (Amazon Web Services) [4] es reconocida como una de las **tecnologías principales** que se abordarán en el proyecto 'Internet de las Cosas con Python y Raspberry Pi'. Su rol es fundamental en el contexto más amplio de las Tecnologías Clave del proyecto, particularmente en el **desarrollo y despliegue de la infraestructura IoT en la nube**. En este contexto, AWS se utilizara para:

- **Plataforma de Despliegue en la Nube**:
  - El proyecto enseña a **desplegar el servidor IoT en la nube de AWS**. Esto complementa el desarrollo de dispositivos en Raspberry Pi y servidores locales, logrando un **'equilibrio completo '** entre ambos.
  - El objetivo es construir una **plataforma en la nube *serverless*** en la que múltiples usuarios puedan iniciar sesión de forma segura y controlar y monitorear sus dispositivos autorizados en tiempo real.
  - El producto final del proyecto incluirá una **infraestructura a nivel de la nube**.
- **Alojamiento del Servidor Remoto**:
  - AWS es la plataforma donde reside el **servidor remoto** para la aplicación IoT. Se menciona específicamente una **instancia EC2**, lo que implica el uso de servicios de cómputo virtual de AWS para alojar el servidor.
- **Configuración de Seguridad en la Nube**:

- Para asegurar el servidor alojado, se gestionanran las **reglas de seguridad de entrada (inbound security rules)** en AWS. Específicamente, se deben **asignar reglas de seguridad de entrada para HTTPS** en la configuración de la instancia EC2 para permitir el tráfico seguro a través del puerto 443. Esto es crucial para la implementación de SSL/TLS con certificados Let's Encrypt.

En resumen, AWS es la **columna vertebral de la infraestructura en la nube** para la solución IoT del proyecto, permitiendo el despliegue de servidores remotos escalables y seguros, y facilitando la gestión de la conectividad y la seguridad a nivel de la nube para los dispositivos y usuarios.

1.6.5. PubNub

PubNub [5] es explícitamente una de las **tecnologías principales** que se abordarán en el proyecto 'Internet de las Cosas con Python y Raspberry Pi '. Su papel es fundamental en el contexto más amplio de las Tecnologías Clave, especialmente en la **gestión de la comunicación en tiempo real y la seguridad de acceso** en entornos IoT. En el contexto de las Tecnologías Clave, PubNub se utiliza para:

- **Protocolo de Comunicación en Tiempo Real y Ligero**:
  - En el proyecto se estudiará en profundidad los protocolos de comunicación en tiempo real y ligeros para Internet de las Cosas, incluyendo MQTT, WebSockets y, finalmente, realizará una **demostración práctica con PubNub**.
  - Es seleccionado como el **principal protocolo de comunicación** para un proyecto reconstruido en la etapa 4, reemplazando a la técnica de *long polling* con AJAX. Esto subraya su importancia para una comunicación eficiente y reactiva en la aplicación IoT.
- **Gestión de Acceso y Permisos de Usuario**:
  - PubNub se utilizara para implementar una funcionalidad clave: el **administrador de acceso de PubNub**. Esta característica permitira a los usuarios administradores **otorgar acceso de lectura y escritura en tiempo real** a todos los usuarios no administradores y a los dispositivos.
  - La lógica de la aplicación Flask interactúa directamente con PubNub. Después de almacenar los permisos de lectura y escritura en la base de datos, la aplicación **llama al servidor de PubNub para conceder acceso de lectura y escritura específico a este usuario**.
  - La concesión de permisos de lectura y escritura a través de PubNub es el segundo paso de un proceso, siendo el primero la generación de una clave de autorización para el usuario específico y su almacenamiento en la base de datos.
- **Seguridad y Control en el Ecosistema IoT**:
  - Al permitir el otorgamiento de permisos de acceso en tiempo real, PubNub contribuye significativamente a un **ecosistema IoT fuerte, seguro, en tiempo real y escalable**. Esto es vital para asegurar que solo los usuarios y dispositivos autorizados puedan controlar y monitorear otros dispositivos.

En síntesis, PubNub es una **tecnología crucial** que no solo facilita la comunicación en tiempo real de baja latencia en el sistema IoT, sino que también es instrumental en la implementación de un **modelo de seguridad robusto** mediante la gestión dinámica de permisos de acceso, lo que es esencial para construir una **plataforma IoT multiusuario y segura**.

Capítulo 2

ETAPAS DEL SISTEMA IoT

En esta sección se presentan las etapas cubiertas para la implementación del sistema **Internet de las Cosas (IoT) con Python y Raspberry Pi**, siguiendo un enfoque práctico y progresivo para la construcción integral del mismo. De este modo, los desarrolladores del sistema podrán adquirir las competencias necesarias para implementar soluciones seguras, escalables y en tiempo real, que abarquen desde la gestión del hardware básico y el despliegue en la nube hasta la administración de usuarios.

- **Etapa 1: Introducción a IoT**:
  - Esta etapa inicial se enfocó en establecer una comprensión fundamental del Internet de las Cosas. Se aprendió qué es el Internet de las Cosas y sus principales componentes, tales como los dispositivos inteligentes, los sensores y los actuadores.
  - También se familiarizaron con cuatro importantes modelos de comunicación del Internet de las Cosas que más adelante abordaremos.
- **Etapa 2: Primer Proyecto IoT (Detector de Movimiento Antirrobo)**:
  - Esta parte se enfocó en el desarrollo de un proyecto IoT simple (sin utilizar la nube).
  - El proyecto consistió en un dispositivo IoT que detecta movimiento, activa una alarma y envía alertas. También permite la desactivación de la alarma, logrando una comunicación bidireccional. Todo dentro de una red local o Wi-Fi.
  - Se utilizó un sensor PIR (detector de movimiento) y un zumbador (*buzzer*).
  - La comunicación entre el cliente y el servidor se realizó inicialmente con la **técnica AJAX**. Se montó un servidor web HTTP Flask básico en Raspberry Pi para operar dentro de la red Wi-Fi local.
- **Etapa 3: Seguridad y Protocolos de Comunicación**:
  - Después de analizar las ventajas y desventajas del proyecto anterior, en esta etapa profundizó luego en los protocolos de comunicación en tiempo real y ligeros para IoT, como **WebSockets y MQTT** y se realizó una demostración práctica con PubNub.
  - Se abordó la seguridad en Internet y criptografía, incluyendo SSL/TLS y protocolos HTTP. WebSockets permiten una sesión de comunicación interactiva y bidireccional con baja latencia, mientras que MQTT es un protocolo de mensajería ligero para M2M e IoT, basado en un modelo de publicación/suscripción y que puede correr sobre WebSockets.
- **Etapa 4: Reconstrucción del Proyecto con PubNub y AWS**:
  - El proyecto inicial de la etapa 2 se reconstruyó para usar PubNub como el principal protocolo de comunicación en lugar de AJAX (*long polling*).
  - También se aprendió a desplegar el servidor IoT en la nube de AWS.
- **Etapa 5: Implementación de Seguridad**:
  - Esta etapa se dedicó a la implementación de las terminologías de seguridad estudiadas previamente.

- Se obtuvo un nombre de dominio personalizado y se aseguro con certificados SSL/TLS Let's Encrypt. Este proceso implico instalar *Certbot* y configurar Apache para redirigir el tráfico de HTTP a HTTPS.
- Se implemento una funcionalidad de inicio de sesión de usuario segura y se almacenaron los detalles del usuario en una base de datos integrada.
- Es crucial asignar reglas de seguridad de entrada para HTTPS (puerto 443) en el servidor remoto de AWS para permitir la conexión segura. Una vez configurado, el servidor se redirige de HTTP a HTTPS, mostrando un icono de candado verde y garantizando una comunicación cifrada de extremo a extremo.

- **Etapa 6: Conexión Segura de Usuarios y Dispositivos**:
  - Se implemento una forma segura para que usuarios y dispositivos IoT se conectaran al servidor.
  - Se utilizo la funcionalidad de administrador de acceso de PubNub para que los usuarios administradores puedan otorgar acceso de lectura y escritura en tiempo real a usuarios no administradores y a dispositivos.
  - Esto incluyo desarrollar un panel de administración para listar usuarios en línea y botones para conceder o revocar permisos, asegurando que solo los usuarios administradores puedan ver y utilizar este panel. La funcionalidad se gestiono enviando solicitudes desde el *frontend* (JavaScript) a la aplicación Flask del servidor.

- **etapa 7: Proyecto Final: Sistema de Monitoreo**:
  - Esta etapa final se centro en un sistema de monitoreo atmosférico más complejo, añadiendo más sensores y actuadores.
  - Se adquirió experiencia convertidores digitales, interfaz periférica serial (SPI) y otros conceptos avanzados.
  - El producto finalfué un dashboard mostrando gráficos visuales con datos de sensores y actuadores en tiempo real, lo que demuestro la infraestructura completa de IoT a nivel de la nube.

Al finalizar, se entendera lo que se necesita para construir una solución IoT propia e integral, abarcando desde la simplicidad a nivel de dispositivo hasta la complejidad de la infraestructura a nivel de la nube.Se podrá expandir el sistema IoT añadiendo más dispositivos y funcionalidades según surjan mas necesidades.

2.1. Introducción a IoT

La **etapa 1: Introducción a IoT** es el punto de partida fundamental del proyecto 'Internet de las Cosas con Python y Raspberry Pi ', diseñada para establecer una base sólida antes de que los estudiantes se sumerjan en proyectos prácticos. En el contexto más amplio de las Secciones del Curso, esta primera etapa es crucial porque introduce los conceptos teóricos esenciales que sustentan todo el desarrollo posterior. Los objetivos y contenidos clave de la etapa 1 incluyen:

- **Comprender qué es el Internet de las Cosas (IoT)**.
- Familiarizarse con los **principales componentes del IoT**, como:
  - **Dispositivos inteligentes**.
  - **Sensores**.
  - **Actuadores**.
- Conocer **cuatro importantes modelos de comunicación del Internet de las Cosas**.

Esta etapa es vital para cumplir el enfoque del proyecto de ayudar a los estudiantes a **'entender el porqué antes del cómo y el qué ' (understand the why before how and what)**. Al establecer una comprensión clara de los fundamentos y los modelos de comunicación en la etapa 1, los estudiantes estarán mejor equipados para abordar los proyectos y las tecnologías más avanzadas que se presentarán en las secciones posteriores, como el desarrollo de un detector de movimiento (etapa 2), protocolos de comunicación en tiempo real (etapa 3), seguridad (etapa 5) y la construcción de una plataforma en la nube *serverless* (objetivo principal del proyecto).

2.1.1. Que es IoT

En el contexto más amplio de la **etapa 1: Introducción a IoT**, se indica que el objetivo primordial de esta etapa es que los estudiantes logren una **comprensión fundamental de '¿Qué es el Internet de las Cosas (IoT)? '**. Esta es la piedra angular sobre la que se construirá todo el conocimiento y las habilidades prácticas a lo largo del proyecto. La importancia de abordar **'¿Qué es IoT? '** al inicio radica en el enfoque pedagógico del proyecto, que busca ayudar a los estudiantes a **'entender el porqué antes del cómo y el qué ' (understand the why before how and what)**. Esto significa que, antes de sumergirse en la implementación de proyectos o el uso de tecnologías específicas, es esencial comprender la naturaleza y el propósito general del Internet de las Cosas. Para lograr esta comprensión, la etapa 1 no solo define el concepto, sino que también introduce los elementos que lo constituyen y cómo interactúan:

- **Componentes Mayores**: Los estudiantes se familiarizarán con los **'principales componentes ' (major components)** del IoT, que incluyen **dispositivos inteligentes, sensores y actuadores**. Comprender estos elementos es fundamental para entender cómo los sistemas IoT recogen datos del entorno (sensores), toman decisiones y actúan sobre él (actuadores), y cómo se integran en una red más amplia (dispositivos inteligentes).
- **Modelos de Comunicación**: La etapa también aborda **'cuatro importantes modelos de comunicación del Internet de las Cosas ' (four important Internet of Things communication models)**. Esto es crucial para entender cómo los diversos componentes del IoT se conectan e intercambian información, sentando las bases para discusiones más profundas sobre protocolos y seguridad en secciones posteriores.

En resumen, la pregunta **'¿Qué es IoT? '** es el eje central de la etapa 1, que proporciona la base teórica indispensable para que los estudiantes puedan abordar con éxito los desafíos de desarrollo práctico y la construcción de una plataforma IoT completa, segura y escalable que se presenta en el resto del proyecto.

2.1.2. Componentes Mayores (Dispositivos, Sensores, Actuadores)

En el contexto más amplio de la **etapa 1: Introducción a IoT**, se establece que uno de los objetivos principales es que los estudiantes comprendan los **Componentes Mayores** del Internet de las Cosas. Esta etapa es fundamental porque sienta las bases teóricas necesarias para entender el **'porqué antes del cómo y el qué ' (understand the why before how and what)**. Específicamente, la etapa 1 se encarga de:

- **Entender qué es el Internet de las Cosas** y sus elementos clave.
- Familiarizar a los estudiantes con los **'principales componentes ' (major components)** del IoT. Estos componentes se desglosan en:
  - **Dispositivos inteligentes (smart devices)**.
  - **Sensores**.
  - **Actuadores**.

La introducción de estos componentes en la etapa 1 es crucial para el desarrollo del proyecto. Permite a los estudiantes construir un conocimiento fundamental que luego aplicarán en las secciones subsiguientes. Por ejemplo, en la etapa 2, se 'ensuciarán las manos ' al desarrollar un detector de movimiento, utilizando específicamente un **sensor PIR** y un ***buzzer*** (actuador). La comprensión de qué son los sensores y actuadores, proporcionada en la etapa 1, es un prerrequisito para comprender su función y manipulación en proyectos prácticos. De manera similar, el proyecto final en la etapa 7, un sistema de monitoreo atmosférico, involucrará la adición de 'más **sensores y actuadores** ', y la comprensión de estos componentes desde la introducción es clave para entender la complejidad y funcionalidad de dicho sistema. Además de los componentes, la etapa 1 también cubre **'cuatro importantes modelos de comunicación del Internet de las Cosas ' (four important Internet of Things communication models)**. Esto, junto con la comprensión de dispositivos, sensores y actuadores, proporciona una visión holística inicial de cómo interactúan los elementos en una solución IoT completa, preparando a los estudiantes para los desafíos de comunicación y seguridad que se abordarán en secciones posteriores como la etapa 3 (protocolos de comunicación) y la etapa 5 (seguridad).

2.1.3. Modelos de Comunicación IoT

En el contexto más amplio de la **etapa 1: Introducción a IoT**, se indican claramente que uno de los objetivos clave es que los estudiantes se familiaricen con los **'cuatro importantes modelos de comunicación del Internet de las Cosas ' (four important Internet of Things communication models)**. Esta inclusión en la primera etapa del proyecto es fundamental y se alinea con el enfoque pedagógico de ayudar a los estudiantes a **'entender el porqué antes del cómo y el qué ' (understand the why before how and what)**. Antes de sumergirse en la implementación de protocolos específicos o en la construcción de sistemas, la etapa 1 establece una comprensión conceptual de cómo interactúan los diferentes componentes en el ecosistema IoT. Al presentar estos cuatro modelos de comunicación al principio, el proyecto busca:

- **Establecer una Base Teórica**: Proporcionar a los estudiantes el conocimiento fundamental sobre las diversas formas en que los dispositivos, sensores y actuadores de IoT pueden intercambiar información.
- **Preparar para Temas Avanzados**: Sentar las bases para las discusiones más detalladas sobre protocolos de comunicación específicos que se abordarán en secciones posteriores del proyecto. Por ejemplo, en la etapa 3, se estudiarán en profundidad protocolos de comunicación en tiempo real y ligeros, como **WebSockets y MQTT**, que son fundamentales para el IoT. WebSockets permite sesiones de comunicación interactivas y bidireccionales, mientras que MQTT es un protocolo de mensajería ligero para la comunicación máquina a máquina, a menudo funcionando sobre WebSockets.

Por lo tanto, la discusión sobre los 'cuatro modelos de comunicación IoT ' en la etapa 1 es crucial para construir una comprensión integral del funcionamiento del Internet de las Cosas, permitiendo a los estudiantes contextualizar y aplicar el conocimiento de protocolos más específicos y avanzados en proyectos prácticos a lo largo del proyecto.

2.2. Primer Proyecto IoT

La **etapa 2: Primer Proyecto IoT** es una etapa fundamental en el proyecto 'Internet de las Cosas con Python y Raspberry Pi ', ya que marca el momento en que los estudiantes pasan de la teoría a la práctica, construyendo su primer sistema IoT funcional. En el contexto más amplio de las Secciones del Curso, esta etapa es crucial por varias razones:

- **Aplicación de Conceptos Fundamentales**: La etapa 2 capitaliza los conocimientos adquiridos en la etapa 1 ('Introducción a IoT '), donde se establecieron los fundamentos del IoT, sus componentes principales (dispositivos, **sensores y actuadores**) y los modelos de comunicación. Aquí, los estudiantes 'se ensucian las manos ' aplicando directamente estos conceptos al trabajar con un **sensor PIR** (un tipo de sensor de movimiento) y un ***buzzer*** (un actuador).
- **Primer Proyecto Práctico**: El proyecto principal de la etapa 2 es un **detector de movimiento**. Este proyecto abarca:
  - **Diagrama de circuito de hardware**.
  - **Código Python** para interactuar con los sensores y el servidor HTTP.
  - La detección de movimiento que activa una alarma y envía alertas al usuario, demostrando una **comunicación unidireccional** del servidor al usuario.
  - La adición de funcionalidad para que los usuarios desactiven la alarma, estableciendo una **comunicación bidireccional** del usuario al servidor.
- **Detalles de Hardware y Programación Básica**:
  - Se introduce el **sensor PIR**, que detecta el movimiento al percibir la energía térmica (radiaciones infrarrojas) emitida por humanos o animales. Se explican sus pines (Tierra, VCC para 5V, Salida de nivel lógico alto si se detecta objeto) y los potenciómetros para ajustar la sensibilidad y el tiempo de retardo. También se detallan los modos de disparo repetible y no repetible.

- Se presenta el ***buzzer***, que produce sonido al vibrar un disco metálico cuando se le aplica corriente, y se explica cómo controlarlo mediante la generación de una onda cuadrada con Python.
- Se enseña a conectar el sensor PIR y el ***buzzer*** a la Raspberry Pi y a escribir código Python para detectar señales y controlar el actuador.

- **Servidor Web Local y Comunicación AJAX**:
  - Se desarrolla un servidor web HTTP básico con **Python Flask** en la Raspberry Pi, que se ejecuta en la red Wi-Fi local.
  - Los usuarios acceden a una página web desde sus navegadores, donde se muestran el estado de detección de movimiento y el estado de conexión.
  - Se implementa una 'solicitud *keepalive* ' (latido) que el navegador envía periódicamente (cada cinco segundos) al servidor de la Raspberry Pi para mantener la conexión activa y recibir actualizaciones en vivo del sensor.
  - La comunicación entre el cliente y el servidor se realiza utilizando la **técnica AJAX**.
  - Es importante destacar que este servidor solo es accesible localmente, dentro de la misma red Wi-Fi.

- **Preparación para Secciones Futuras**: La etapa 2 sirve como un trampolín. Aunque es un proyecto funcional, se indican que se discutirán las **ventajas y desventajas** de la forma en que se desarrolló, especialmente en lo que respecta a la comunicación. Esto prepara el terreno para la **etapa 3**, donde se estudiarán a fondo **protocolos de comunicación en tiempo real y ligeros**, como **MQTT y WebSockets**, y se abordará la seguridad en Internet. De esta manera, el proyecto progresa desde una implementación básica y local hacia soluciones más robustas, escalables y seguras en la nube.

- **Requisitos Previos**: Se aconseja a los estudiantes configurar sus Raspberry Pis, incluyendo la instalación del sistema operativo en una tarjeta SD de al menos 8GB, y habilitar SSH si se desea acceso remoto.

En resumen, la etapa 2 es el primer contacto práctico y guiado con la construcción de un sistema IoT, permitiendo a los estudiantes consolidar los conocimientos teóricos de la etapa 1 y prepararse para las complejidades de la comunicación, seguridad y despliegue en la nube que se abordarán en las secciones posteriores del proyecto.

2.2.1. Detector de Movimiento Antirobo

En el contexto más amplio de la **etapa 2: Primer Proyecto IoT**, se detallan el **Detector de Movimiento Antirrobo** como el proyecto central y fundamental de esta etapa del proyecto. Este proyecto sirve como la primera incursión práctica de los estudiantes en la construcción de un sistema IoT funcional, aplicando los conceptos teóricos aprendidos en la etapa 1. Aquí se desglosa lo que se dice sobre este proyecto:

- **Objetivo y Funcionalidad General**:
  - El propósito principal es construir un **dispositivo IoT defensivo contra robos** que detecte movimiento, active una alarma y envíe alertas al usuario.
  - Inicialmente, establece una **comunicación unidireccional** del servidor al usuario para enviar alertas.
  - Para lograr una **comunicación bidireccional**, se añade una funcionalidad que permite a los usuarios desactivar la alarma desde el servidor.

- **Componentes de Hardware**:
  - **Sensor PIR (Detector de Movimiento)**: Utiliza un elemento sensor piezoeléctrico (RE200BL) que genera energía al exponerse al calor. Detecta el movimiento de humanos o animales por las radiaciones infrarrojas que emiten. Es un sensor pasivo, lo que significa que no emite energía para la detección, sino que detecta la energía emitida por otros objetos.

- Incluye una tapa de plástico para ampliar la cobertura del área de detección.
- Tiene tres pines: **Tierra**, **VCC (para 5V)** y un **pin de salida** que proporciona un nivel lógico alto si se detecta un objeto.
- Posee dos potenciómetros: uno para ajustar la **sensibilidad** del sensor y otro para ajustar el **tiempo** durante el cual la señal de entrada permanece en alto tras la detección (de 0.3 segundos a 5 minutos).
- Dispone de pines para seleccionar modos de disparo: **disparo no repetible** (la salida vuelve a bajo después del tiempo de retardo) y **disparo repetible** (la salida permanece en alto mientras el objeto detectado esté presente).
- ***Buzzer***: Compuesto por una carcasa con tres pines (**VCC 5V, Tierra y Señal**) y un elemento piezoeléctrico con un disco metálico que vibra al aplicar corriente, produciendo sonido. Se controla generando una **onda cuadrada** con Python, alternando la señal entre alto y bajo para generar el sonido.

- **Implementación Técnica y Software**:
  - Los estudiantes conectarán el **sensor PIR y el *buzzer*** a la Raspberry Pi.
  - Se escribirá **código Python** para detectar las señales del sensor y controlar el *buzzer*.
  - Se desarrollará un **servidor web HTTP básico** utilizando **Python Flask** en la Raspberry Pi. Este servidor se ejecutará en la **red Wi-Fi local**.
  - Los usuarios accederán a una página web desde sus navegadores (dentro de la misma red Wi-Fi local).
  - Una vez cargada la página, el navegador del usuario enviará una **'solicitud *keepalive* '** (latido) al servidor de la Raspberry Pi periódicamente (cada cinco segundos). Esto actúa como un latido para asegurar que la conexión con el servidor sigue activa.
  - En cada respuesta del *keepalive*, el servidor enviará el estado del sensor y los datos al usuario, proporcionando **actualizaciones en vivo**.
  - La comunicación entre el cliente y el servidor se realiza utilizando la **técnica AJAX**.
  - Se agregará un botón en la página web para que los usuarios puedan **controlar los actuadores**, específicamente para desactivar la alarma (el *buzzer*).
  - La página web mostrará el estado de detección de movimiento y el estado de la conexión.

- **Preparación y Requisitos Previos**:
  - Se aconseja a los estudiantes configurar sus Raspberry Pis, incluyendo la instalación del sistema operativo en una tarjeta SD de al menos 8 GB y la habilitación de SSH para acceso remoto.

- **Importancia en el Contexto del Curso**:
  - La etapa 2, con el proyecto del Detector de Movimiento Antirrobo, es el **primer paso práctico** del proyecto, donde los estudiantes 'se ensucian las manos'.
  - Posteriormente, en la etapa 3, se discutirán las **ventajas y desventajas** de la forma en que se desarrolló la comunicación en este proyecto. Esto servirá como base para estudiar en profundidad **protocolos de comunicación en tiempo real y ligeros** como MQTT y WebSockets, así como temas de seguridad en Internet. De esta manera, el proyecto sienta las bases para soluciones más avanzadas y seguras.

2.2.2. Sensores (PIR)

En el contexto más amplio de la **etapa 2: Primer Proyecto IoT**, el **Sensor PIR (Passive Infrared Sensor)** es un componente fundamental y el dispositivo principal de entrada para el **Detector de Movimiento Antirrobo**. Esta etapa se centra en que los estudiantes se familiaricen con el hardware y la programación básica, y el sensor PIR juega un papel crucial en este aprendizaje práctico. A continuación, se dan los detalles de el sensor PIR:

- **Nombre y Principio de Funcionamiento**

- El PIR es un **módulo detector de movimiento**.
- Su elemento sensor principal se llama **RE200BL**, que es un **sensor piezoeléctrico**.
- Funciona generando energía cuando se expone al calor.
- Detecta el movimiento porque **humanos o animales emiten energía térmica en forma de radiaciones infrarrojas**. Cuando un cuerpo entra en el rango del sensor, detecta este movimiento.
- El término 'pasivo ' significa que el sensor **no utiliza ninguna energía para el propósito de detección**, sino que funciona detectando la energía emitida por otros objetos.

- **Componentes Físicos y Pines**
  - El módulo incluye una **tapa de plástico especialmente diseñada** que se utiliza para expandir la cobertura del área de detección.
  - Posee tres pines principales:
    - **Ground (Tierra)**.
    - **VCC (para 5 voltios)**, utilizado para alimentar el sensor.
    - **Output (Salida)**, que proporciona un **nivel lógico alto** si se detecta un objeto y un nivel lógico bajo en caso contrario.

- **Potenciómetros para Ajuste**
  - El módulo dispone de **dos potenciómetros** para ajustar su comportamiento:
    - Uno para ajustar la **sensibilidad** del sensor.
    - Otro para ajustar el **tiempo** durante el cual la señal de entrada permanece en un nivel alto cuando se detecta un objeto. Este tiempo puede ajustarse desde **0.3 segundos hasta 5 minutos**.

- **Modos de Disparo (Trigger Modes)**
  - El módulo también tiene **tres pines adicionales con un *jumper*** entre dos de ellos, que sirven para seleccionar los modos de disparo:
    - **Disparo no repetible (Non-repeatable trigger)**: Cuando el sensor emite una señal alta y el tiempo de retardo ha terminado, la salida cambia automáticamente de alto a bajo.
    - **Disparo repetible (Repeatable trigger)**: La salida permanece en un nivel alto todo el tiempo que el objeto detectado esté presente en el rango del sensor.

- **Rol en el Detector de Movimiento Antirrobo**
  - En este proyecto, el sensor PIR se conecta a una Raspberry Pi.
  - El código Python se escribirá para **detectar las señales altas y bajas** que el sensor PIR produce al detectar movimiento.
  - Estas detecciones son la base para **activar la alarma** (controlando un *buzzer*) y **enviar alertas** al usuario, estableciendo así una comunicación unidireccional inicial del servidor al usuario.

En resumen, el sensor PIR es un componente didáctico clave en la etapa 2, permitiendo a los estudiantes comprender y experimentar cómo un sensor real interactúa con una Raspberry Pi para recopilar datos del entorno y utilizarlos en un sistema IoT funcional de detección de movimiento.

2.2.3. Actuadores (Zunbador)

En el contexto más amplio de la **etapa 2: Primer Proyecto IoT**, el **Actuador (Zumbador)** es un componente clave del proyecto **Detector de Movimiento Antirrobo**, funcionando como el dispositivo de salida principal para la alarma del sistema. Su implementación es fundamental para establecer la funcionalidad de alerta y, posteriormente, la comunicación bidireccional en el sistema IoT. A continuación, se dan los detalles de el zumbador:

- **Propósito en el Proyecto**:
  - El objetivo principal del proyecto es crear un dispositivo IoT defensivo contra robos que **active una alarma** al detectar movimiento. El zumbador es el componente encargado de producir esta alarma sonora.
  - Inicialmente, el proyecto se enfoca en la comunicación unidireccional del servidor al usuario. Para lograr una **comunicación bidireccional**, se añade una funcionalidad que permite a los usuarios **desactivar el zumbador (la alarma)** desde el servidor a través de una interfaz web.
- **Descripción del Hardware del Zumbador**:
  - Consta de una **carcasa exterior con tres pines**: VCC (para 5 voltios), Tierra y Señal.
  - En su interior, contiene un **elemento piezoeléctrico** rodeado por un **disco metálico que vibra**.
  - El sonido se produce cuando se **aplica corriente al zumbador**, lo que hace que el disco se contraiga y expanda, generando vibraciones y, por ende, sonido.
  - Es posible cambiar el **tono o la 'melodía '** del zumbador al variar la frecuencia de la corriente aplicada, lo que altera la velocidad de vibración del disco.
- **Control y Programación**:
  - Para controlar el zumbador, se generará una **onda cuadrada**.
  - Esto se logra alternando el pin de señal entre un estado lógico alto y bajo, con pequeños intervalos de espera de milisegundos entre cada cambio, repitiendo este proceso.
  - Los estudiantes escribirán **código Python básico** para controlar el zumbador en respuesta a la detección de movimiento del sensor PIR.
- **Interacción con la Interfaz de Usuario Web**:
  - En la página web que sirve como interfaz del proyecto, se añadirá un **botón interruptor (switch button)**. Este botón permitirá a los usuarios **controlar los actuadores**, específicamente para **desactivar el zumbador**. Esta funcionalidad es clave para la comunicación bidireccional del sistema.

2.2.4. Servidor WEB con FLASK (Comunicación Unidireccional y Bidireccional)

En el contexto más amplio de la **etapa 2: Primer Proyecto IoT**, que se centra en el desarrollo del **Detector de Movimiento Antirrobo**, el **Servidor Web con Flask** es el pilar central para la comunicación y la interacción con el dispositivo IoT. Se describe su implementación y evolución desde una comunicación unidireccional básica a una bidireccional, sentando las bases para conceptos más avanzados en el proyecto. Aquí se dan los detalles de el Servidor Web con Flask y su rol en la comunicación:

- **Propósito del Servidor Web con Flask en la etapa 2**:
  - Los estudiantes desarrollarán un **servidor web HTTP básico utilizando Python Flask** en la Raspberry Pi.
  - Este servidor es fundamental para que el dispositivo IoT defensivo contra robos pueda **enviar alertas al usuario** y, posteriormente, permitir que los usuarios **interactúen con el servidor** para controlar el dispositivo.
  - El objetivo es mostrar datos de sensores a través de una aplicación web.
- **Funcionamiento Técnico del Servidor y Comunicación Unidireccional Inicial**:
  - El servidor se ejecutará en la **red Wi-Fi local** de los usuarios.
  - Una vez que la Raspberry Pi tenga una dirección IP (por ejemplo, 192.168.1.250), los usuarios podrán acceder a una página web desde sus navegadores dentro de la misma red Wi-Fi.
  - La comunicación entre el cliente y el servidor se realizará utilizando la **técnica AJAX**.

- Para mantener la conexión activa y recibir actualizaciones, el navegador del usuario enviará una **'solicitud *keepalive* '(latido)** al servidor de la Raspberry Pi periódicamente (cada cinco segundos). Esto actúa como un 'latido ' para confirmar que la conexión con el servidor sigue activa.
- En cada respuesta del *keepalive*, el servidor **enviará el estado del sensor y los datos al usuario**, proporcionando así **actualizaciones en vivo** de, por ejemplo, el estado de detección de movimiento y el estado de la conexión.
- Esta fase inicial cumple con la **comunicación unidireccional del servidor al usuario**, donde el servidor informa al usuario sobre el estado del sensor y las alertas.

- **Evolución hacia la Comunicación Bidireccional**:
  - Para lograr una **comunicación bidireccional del usuario al servidor**, se añade una funcionalidad que permite a los usuarios **comunicarse con el servidor**.
  - Específicamente, se implementa un **botón interruptor (switch button) en la página web** que permite a los usuarios **controlar los actuadores**, como **desactivar el zumbador (alarma)**.
  - Este botón envía una solicitud desde el cliente (navegador) al servidor Flask para cambiar el estado del dispositivo. Se menciona la adición de un *endpoint* en la aplicación Flask para recibir estas solicitudes.
  - Posteriormente en el proyecto (aunque ya se conceptualiza aquí), se detallará cómo el servidor también puede devolver información como `user_ID` y una lista de `online_users` para poblar la interfaz web, lo que refuerza la naturaleza bidireccional y la capacidad de gestión de usuarios.

- **Limitaciones y Futuras Mejoras**:
  - Es importante destacar que, en esta etapa, el servidor Flask se ejecuta **localmente en la Raspberry Pi**, lo que significa que los usuarios solo pueden acceder a él desde **dentro de la misma red Wi-Fi local**.
  - Se indica también que, después de discutir las ventajas y desventajas de este método de comunicación desarrollado en la etapa 2, el proyecto procederá a estudiar en profundidad **protocolos de comunicación en tiempo real y ligeros** como **MQTT y WebSockets** en la etapa 3, así como temas de seguridad en Internet. Esto implica que el enfoque inicial con Flask y AJAX, aunque funcional, se considera una base para soluciones más robustas y escalables.

En resumen, el servidor web basado en Flask en la etapa 2 es el elemento clave que convierte la Raspberry Pi en un dispositivo IoT interactivo, permitiendo inicialmente el monitoreo de movimiento y la activación de alarmas (unidireccional), para luego evolucionar a un control remoto de la alarma por parte del usuario (bidireccional), todo ello como preparación para arquitecturas IoT más complejas y seguras.

2.2.5. Técnica AJAX para Comunicación

En el contexto más amplio de la **etapa 2: Primer Proyecto IoT**, que se centra en el desarrollo de un **Detector de Movimiento Antirrobo**, la **técnica AJAX para comunicación** es el método principal utilizado para facilitar la interacción entre el cliente (navegador web del usuario) y el servidor Flask que se ejecuta en la Raspberry Pi. Es fundamental para establecer la comunicación inicial en el proyecto y proporcionar actualizaciones en tiempo real. Se detalla lo siguiente sobre AJAX en esta etapa:

- **Propósito y Rol Inicial**:
  - La técnica AJAX se utiliza para la **comunicación entre el cliente y el servidor**.
  - Su objetivo principal es permitir que el servidor web de la Raspberry Pi **envíe alertas al usuario** y muestre datos de los sensores a través de una aplicación web. Esto cumple con la primera fase de **comunicación unidireccional del servidor al usuario**.
  - Se utiliza para obtener **actualizaciones en vivo** del estado del sensor y los datos.

- **Mecanismo de 'Keepalive Request ' (Latido)**:

- Una vez que el usuario recibe la página web del servidor Flask, el navegador del usuario enviará una **'solicitud *keepalive* '(latido)** al servidor web de la Raspberry Pi de forma **periódica, cada cinco segundos**.
- Esta solicitud actúa como un 'latido ' para **confirmar que la conexión con el servidor sigue activa**, es decir, que el servidor aún está en funcionamiento.
- En la respuesta a cada solicitud *keepalive*, el servidor tiene la capacidad de **enviar el estado del sensor y los datos al usuario**. Por ejemplo, esto incluye el estado de la detección de movimiento y el estado de la conexión.

- **Limitaciones y Evolución Futura**:
  - Aunque eficaz para la comunicación inicial y las actualizaciones en vivo, se sugieren que la técnica AJAX (junto con el 'long polling ' en general) tiene **ventajas y desventajas**.
  - Después de esta etapa 2, el proyecto abordará en profundidad **protocolos de comunicación en tiempo real y ligeros** como **MQTT y WebSockets** en la etapa 3. Esto implica que AJAX, si bien es una base de aprendizaje útil, se considera una solución inicial que será reemplazada por tecnologías más avanzadas para una comunicación IoT más robusta y escalable.

En resumen, la técnica AJAX, implementada a través de solicitudes periódicas de *keepalive*, es la piedra angular de la comunicación en la etapa 2, permitiendo que el Detector de Movimiento Antirrobo basado en Raspberry Pi proporcione información y alertas en tiempo real al usuario a través de una interfaz web local.

2.3. Protocolos de Comunicación y Seguridad

En el contexto más amplio de las **Secciones del Curso**, la **etapa 3: Protocolos de Comunicación y Seguridad** es una etapa crucial que sigue a la implementación inicial del proyecto IoT básico en la etapa 2. Su objetivo principal es profundizar en las **tecnologías esenciales para una comunicación IoT robusta, segura y en tiempo real**, abordando las limitaciones de los métodos anteriores y sentando las bases para desarrollos futuros más complejos y seguros del servidor IoT. A continuación, se dan los detalles de la etapa 3:

- **Transición y Propósito**:
  - Después de discutir las ventajas y desventajas del método de comunicación desarrollado en la etapa 2 (que utilizaba AJAX y *long polling*), el proyecto avanza para estudiar a fondo los **protocolos de comunicación en tiempo real y ligeros** para el Internet de las Cosas.
  - También se estudiará la **seguridad en Internet y la criptografía**, preparando el terreno para la implementación de un servidor seguro y un inicio de sesión de usuario en secciones posteriores.

- **Protocolos de Comunicación en Tiempo Real**:
  - La etapa 3 examinará dos de las tecnologías más famosas y adoptadas en IoT para protocolos de comunicación: **WebSockets y MQTT**.
  - Se ofrecerán **demos prácticas con PubNub**, un servicio que probablemente se usará para aplicar estos protocolos.

- **WebSockets**:
  - Permiten a los clientes recibir actualizaciones solo cuando ocurren, sin que el cliente tenga que 'preguntar ' al servidor.
  - Es una tecnología avanzada que posibilita una sesión de comunicación interactiva al abrir una única conexión TCP persistente.
  - Es **bidireccional y *full duplex***, lo que significa que ambas partes (cliente y servidor) pueden enviar mensajes de forma independiente.
  - Se estandarizó en 2011 y es altamente adoptado en aplicaciones IoT debido a su baja latencia y naturaleza bidireccional.

- Funciona mediante una solicitud HTTP inicial donde el cliente pide al servidor **actualizar a un protocolo WebSockets**. Una vez aceptada la conexión (handshake), la sesión se mantiene abierta y persistente hasta que una de las partes la cierra.
- Los paquetes de datos enviados son muy pequeños, con una longitud de trama de 2 a 14 bytes.

- **MQTT (Message Queuing Telemetry Transport)**:
  - Es un **protocolo de transferencia de mensajes ligero** para la comunicación máquina a máquina (M2M) y el Internet de las Cosas.
  - Es muy **eficiente en el uso del ancho de banda**, con solo 2 bytes de sobrecarga.
  - Permite escenarios de transmisión de datos **uno a uno, uno a muchos y muchos a muchos** para dispositivos y aplicaciones.
  - Esto se logra mediante un **modelo de publicación y suscripción**, donde un flujo de datos específico se envía sobre un 'tema' (topic), y los dispositivos suscritos a ese tema pueden recibir los datos.
  - Se ejecuta sobre la red TCP/IP, lo que significa que **puede usarse sobre una capa WebSocket**.
  - Es importante no confundir MQTT con WebSockets; MQTT es un servicio de entrega que puede funcionar *encima* de WebSockets, como si MQTT fuera un servicio de paquetería (DHL) y WebSockets proporcionaran las carreteras y el transporte.
  - El punto central de comunicación en MQTT es el **broker**, que se encarga de despachar todos los mensajes entre el remitente y los receptores.
  - Los clientes publican mensajes al broker que incluyen un 'tema ', que es información de enrutamiento para que el broker reenvíe el mensaje a los receptores suscritos a ese tema.
  - Esto lo hace **altamente escalable**, ya que los clientes no necesitan conocerse entre sí, solo comunicarse a través del tema. Sin embargo, su principal inconveniente es que si el broker falla, toda la comunicación se interrumpe.

- **Seguridad en Internet y Criptografía**:
  - Se abordarán **temas candentes** como:
    - **Algoritmos de clave simétrica y asimétrica**.
    - **Cifrado**.
    - **Firma digital**.
    - **Protocolos SSL/TLS**, también conocidos como **HTTPS**.

En resumen, la etapa 3 es la fase en la que los estudiantes adquieren conocimientos fundamentales sobre **cómo hacer que los sistemas IoT se comuniquen de manera eficiente y segura**, sentando las bases para la implementación de soluciones más complejas y empresariales en las secciones subsiguientes, incluyendo la protección de dominios personalizados y la gestión de inicios de sesión de usuarios.

2.3.1. Protocolos en Tiempo Real

En el contexto más amplio de la **etapa 3: Protocolos de Comunicación y Seguridad**, la discusión sobre los **Protocolos en Tiempo Real** es fundamental, ya que esta etapa se dedica a estudiar a fondo las tecnologías avanzadas de comunicación para el Internet de las Cosas (IoT) que superan las limitaciones de los métodos anteriores (como AJAX y *long polling*). El objetivo es permitir que los sistemas IoT se comuniquen de manera eficiente y robusta. Se detallan dos de los protocolos en tiempo real más importantes y adoptados en el IoT: **WebSockets** y **MQTT**.

WebSockets

- **Naturaleza y Propósito**:
  - Permiten a los clientes **recibir actualizaciones solo cuando ocurren**, sin necesidad de que el cliente esté 'preguntando' al servidor constantemente.

- Es una tecnología avanzada que posibilita una **sesión de comunicación interactiva** al abrir una **única conexión TCP persistente**.
- Es **bidireccional y *full duplex***, lo que significa que tanto el cliente como el servidor pueden enviar mensajes de forma independiente entre sí.
- Fue estandarizado en 2011 y es muy utilizado en aplicaciones IoT debido a su **baja latencia y naturaleza bidireccional**.

- **Funcionamiento**:
  - Inicialmente, un cliente realiza una **solicitud HTTP** pidiendo al servidor que actualice el protocolo a WebSockets.
  - El servidor responde con un **apretón de manos (handshake)**, y una vez aceptada, la sesión se mantiene **abierta y persistente** hasta que una de las partes la cierra.
  - Durante la sesión, los paquetes de datos enviados entre el cliente y el servidor son **muy pequeños**, con una longitud de trama de 2 a 14 bytes.

MQTT (Message Queuing Telemetry Transport)

- **Naturaleza y Propósito**:
  - Es un **protocolo de transferencia de mensajes ligero** (lightweight) diseñado específicamente para la comunicación máquina a máquina (M2M) y el Internet de las Cosas.
  - Es **muy eficiente en el uso del ancho de banda**, utilizando solo 2 bytes de sobrecarga.
  - Permite escenarios de transmisión de datos **uno a uno, uno a muchos y muchos a muchos** para dispositivos y aplicaciones.
- **Modelo de Publicación y Suscripción**:
  - Esto se logra mediante un **modelo de publicación y suscripción**. Un flujo de datos específico se envía sobre un **'tema' (topic)**, y los dispositivos suscritos a ese tema pueden recibir los datos. Por ejemplo, un sensor de humedad puede publicar lecturas en el tema 'jardín ', y una bomba de agua suscrita a ese tema puede encenderse si el nivel de humedad es bajo.
- **Funcionamiento con un Broker**:
  - El punto central de comunicación en MQTT es el **broker**. El broker es el encargado de despachar todos los mensajes entre el remitente y los receptores.
  - Cada cliente que publica un mensaje al broker incluye un 'tema ' dentro del mensaje. Esta información de tema es crucial para que el broker reenvíe el mensaje a los receptores que están suscritos a ese tema. Por ejemplo, un usuario puede publicar la temperatura en un tema llamado 'temperatura', y el broker reenviará estos datos a un aire acondicionado que esté suscrito al mismo tema para establecer la temperatura deseada.
- **Escalabilidad y Limitaciones**:
  - Este diseño lo hace **altamente escalable**, ya que los clientes no necesitan conocerse entre sí; solo necesitan comunicarse a través del tema.
  - El principal inconveniente de MQTT es su **dependencia de la entidad central (el broker)**: si el broker falla, toda la comunicación se interrumpe.
- **Relación con WebSockets**:
  - MQTT se ejecuta sobre la red TCP/IP, lo que significa que **puede utilizarse sobre una capa WebSocket**.
  - Es importante no confundir MQTT con WebSockets; son cosas diferentes. MQTT puede considerarse un servicio de entrega (como DHL) que funciona *sobre* WebSockets, que a su vez proporciona las 'carreteras y el transporte '. Un paquete de datos MQTT se 'envuelve ' dentro de un sobre WebSocket, que a su vez se envuelve en un sobre TCP/IP y se envía por Internet, desempaquetándose en orden inverso al llegar.

En resumen, la etapa 3 establece una base sólida para la comunicación IoT al introducir estos protocolos en tiempo real, permitiendo a los desarrolladores elegir la solución más adecuada para la interactividad y la eficiencia en sus proyectos. Además, se mencionan que esta etapa también aborda la seguridad en Internet y la criptografía, preparando el terreno para servidores seguros y gestión de usuarios.

MQTT

En el contexto más amplio de los **Protocolos en Tiempo Real** que se estudian en la etapa 3: Protocolos de Comunicación y Seguridad, **MQTT (Message Queuing Telemetry Transport)** se presenta como una de las tecnologías más adoptadas y eficientes para la comunicación en el Internet de las Cosas (IoT). Su inclusión subraya la necesidad de superar las limitaciones de métodos de comunicación anteriores, como AJAX y *long polling*, para lograr una comunicación robusta, segura y en tiempo real. Aquí se dan los detalles de MQTT:

- **Definición y Propósito**:
  - MQTT es un **protocolo de transferencia de mensajes ligero** (lightweight).
  - Está diseñado específicamente para la **comunicación máquina a máquina (M2M)** y el **Internet de las Cosas**.
  - Es notablemente **eficiente en el uso del ancho de banda**, con solo 2 bytes de sobrecarga.
- **Modelo de Publicación y Suscripción**:
  - Permite escenarios de **transmisión de datos uno a uno, uno a muchos y muchos a muchos** para dispositivos y aplicaciones.
  - Esto se logra mediante un **modelo de publicación y suscripción**.
  - Un flujo de datos específico se envía sobre un **'tema ' (topic)**, y los dispositivos suscritos a ese tema pueden recibir los datos. Por ejemplo, un sensor de humedad puede publicar el nivel de humedad en un tema llamado 'jardín ', y una bomba de agua suscrita a ese tema puede activarse si el nivel es bajo.
- **El Broker Central**:
  - El punto central de comunicación en MQTT es el **broker**.
  - El broker es el encargado de **despachar todos los mensajes** entre el remitente y los receptores.
  - Cada cliente que publica un mensaje al broker incluye un 'tema ' dentro del mensaje. Esta información de tema sirve como **información de enrutamiento** para que el broker reenvíe el mensaje a los receptores que están suscritos a ese tema. Por ejemplo, un usuario puede publicar la temperatura en un tema llamado 'temperatura ', y el broker reenvía estos datos a un aire acondicionado suscrito al mismo tema para establecer la temperatura deseada.
- **Escalabilidad y Limitaciones**:
  - Este modelo hace que MQTT sea **altamente escalable**, ya que los clientes no necesitan conocerse entre sí; solo tienen que comunicarse a través del tema.
  - Sin embargo, el **principal inconveniente** de este protocolo es su **dependencia de la entidad central (el broker)**: si el broker falla, toda la comunicación se interrumpe.
- **Relación con WebSockets**:
  - MQTT se ejecuta sobre la red **TCP/IP**, lo que significa que **puede utilizarse sobre una capa WebSocket**.
  - Es importante **no confundir MQTT con WebSockets**, ya que son cosas diferentes.
  - Se ofrece una analogía: se puede considerar MQTT como un **servicio de entrega** (como DHL) que funciona *sobre* WebSockets, que a su vez proporciona las 'carreteras y el transporte '. En esta metáfora, un paquete de datos MQTT se 'empaqueta ' dentro de un sobre WebSocket, que luego se envuelve en un sobre TCP/IP y se envía por Internet, desempaquetándose en orden inverso al llegar a su destino.

En el contexto de los protocolos en tiempo real, MQTT se presenta como una solución robusta y eficiente para la comunicación en IoT, especialmente cuando se requiere un manejo escalable de mensajes y un bajo consumo de ancho de banda. Su estudio, junto con WebSockets, en la etapa 3 es fundamental para construir un ecosistema IoT seguro, en tiempo real y escalable, sentando las bases para funcionalidades más avanzadas como la gestión de permisos de acceso en tiempo real para usuarios y dispositivos.

WebSockets

En el contexto más amplio de los **Protocolos en Tiempo Real** dentro de la etapa 3: Protocolos de Comunicación y Seguridad, **WebSockets** se presenta como una tecnología fundamental y ampliamente adoptada en el Internet de las Cosas (IoT). Esta etapa se enfoca en el estudio de protocolos avanzados para la comunicación en IoT, buscando superar las limitaciones de técnicas anteriores como AJAX y *long polling*, para permitir una comunicación eficiente, bidireccional y robusta. A continuación, se dan los detalles de WebSockets:

- **Naturaleza y Propósito**:
  - WebSockets es una **tecnología avanzada** que permite a los clientes **recibir actualizaciones solo cuando ocurren**, eliminando la necesidad de que el cliente esté consultando constantemente al servidor. Esto significa que no hay necesidad de 'preguntar' o hacer *polling* al servidor para obtener actualizaciones.
  - Habilita una **sesión de comunicación interactiva** al abrir una **única conexión TCP persistente**.
  - Es **bidireccional y *full duplex***, lo que permite que tanto el cliente como el servidor envíen mensajes de forma independiente entre sí.
- **Características Clave**:
  - Fue **estandarizado en 2011**.
  - Aunque inicialmente fue adoptado por aplicaciones como juegos, sistemas de chat y aplicaciones web, su **baja latencia y naturaleza bidireccional** lo han hecho altamente adoptado en aplicaciones IoT.
  - Los paquetes de datos enviados durante una sesión abierta son **muy pequeños**, con una longitud de trama de 2 a 14 bytes.
- **Funcionamiento**:
  - Inicialmente, un cliente realiza una **solicitud HTTP** solicitando al servidor que actualice el protocolo a WebSockets.
  - El servidor responde con un **apretón de manos (handshake)**, y una vez aceptado, la sesión se mantiene **abierta y persistente** hasta que una de las partes la cierra.
- **Relación con MQTT**:
  - Es crucial **no confundir WebSockets con MQTT**; son dos cosas diferentes.
  - MQTT, que se ejecuta sobre la red TCP/IP, **puede utilizarse sobre una capa WebSocket**.
  - Se utiliza una analogía para clarificar su relación: **MQTT puede considerarse un servicio de entrega** (como DHL) que funciona *sobre* WebSockets, siendo estos últimos los que proporcionan las 'carreteras y el transporte '. Un paquete de datos MQTT se 'empaqueta' dentro de un sobre WebSocket, que a su vez se envuelve en un sobre TCP/IP y se envía por Internet, desempaquetándose en orden inverso al llegar a su destino.

En resumen, WebSockets es un pilar fundamental para la comunicación en tiempo real en IoT, ofreciendo una solución eficiente, interactiva y de baja latencia que supera las limitaciones de los métodos tradicionales, facilitando así el desarrollo de sistemas IoT avanzados y seguros.

DEMOSTRACIÓN CON PUBNUB

En el contexto más amplio de los **Protocolos en Tiempo Real**, se mencionan a **PubNub** como una tecnología relevante, particularmente en la **etapa 3: Protocolos de Comunicación y Seguridad**, donde se estudian en profundidad los protocolos de comunicación en tiempo real y ligeros para el Internet de las Cosas (IoT), como MQTT y WebSockets. Se indica lo siguiente sobre PubNub en relación con los protocolos en tiempo real y las demostraciones:

- **Integración con Protocolos en Tiempo Real**: Después de discutir las ventajas y desventajas del método de comunicación utilizado en un proyecto anterior (probablemente AJAX y *long polling*), la etapa 3 se dedica a estudiar los protocolos de comunicación en tiempo real y ligeros, mencionando específicamente a **MQTT, WebSockets y, finalmente, algunas demostraciones prácticas con PubNub**. Esto sugiere que PubNub se presenta como una plataforma o servicio que utiliza o facilita la comunicación en tiempo real.
- **Reconstrucción de Proyectos y Despliegue en la Nube**: La **etapa 4** del proyecto se centra en **reconstruir un proyecto anterior para usar PubNub como el principal protocolo de comunicación**, en lugar de las técnicas menos eficientes como AJAX y *long polling*. En esta etapa, también se aprende a desplegar un servidor IoT en la nube de AWS (Amazon Web Services). Esto indica que PubNub no es solo un protocolo, sino una solución que se integra en la arquitectura de un servidor IoT en la nube para manejar la comunicación.
- **Gestión de Accesos en Tiempo Real**: En la **etapa 6**, se implementa una forma segura en la que los usuarios y dispositivos IoT pueden conectarse de manera segura con el servidor IoT. Aquí, se utiliza la **funcionalidad de *Access Manager* de PubNub** para que los usuarios administradores puedan otorgar **permisos de lectura y escritura en tiempo real** a usuarios no administradores y dispositivos. Un ejemplo de esto se muestra en un panel de control donde los administradores pueden ver una lista de usuarios en línea y, mediante botones de interruptor, conceder o revocar permisos de lectura y escritura, que se aplican en tiempo real al seleccionar un botón de 'aplicar'.
- **Implementación de la Lógica de Permisos**:
  - Para implementar estos permisos, se describe un proceso de **generación de una clave de autorización** para un usuario específico y su almacenamiento en la base de datos como un primer paso.
  - Luego, un administrador envía una solicitud desde el código JavaScript a la aplicación Flask, que escucha en cualquier botón de interruptor con un ID que comienza con 'XS ', extrae el ID del usuario, el estado de lectura y el estado de escritura.
  - Finalmente, esta solicitud se envía al servidor para **otorgar acceso de lectura y escritura a un usuario específico a través del servidor PubNub**. La respuesta se maneja en el cliente, y si el acceso es concedido, el canal se vuelve a suscribir.

En resumen, PubNub se presenta como una **plataforma clave para implementar la comunicación en tiempo real en sistemas IoT**, ofreciendo funcionalidades que van más allá de un simple protocolo. Permite la **reconstrucción de proyectos para usarlo como el principal mecanismo de comunicación**, se integra con el despliegue en la nube, y es crucial para la **gestión de permisos de acceso de usuarios y dispositivos en tiempo real**, lo que lo convierte en una parte integral del ecosistema IoT seguro, en tiempo real y escalable que se busca construir.

2.3.2. Seguridad en Internet y Criptografía

En el contexto más amplio de la **etapa 3: Protocolos de Comunicación y Seguridad**, se indican que se realiza un estudio en profundidad sobre la **seguridad en Internet y la criptografía**. Este estudio es fundamental para la construcción de un ecosistema IoT robusto, seguro y en tiempo real. Aquí se dan los detalles de la seguridad en Internet y la criptografía:

- **Temas Candentes de la Criptografía y Seguridad en Internet**: La etapa 3 aborda 'temas candentes ' dentro de la seguridad en Internet y la criptografía, que incluyen:
  - **Algoritmos de clave simétrica y asimétrica**.

- **Cifrado (Encryption)**.
- **Firma digital (Digital Signature)**.
- **SSL/TLS**, que también son conocidos como **protocolos HTTP**.

- **Importancia en el Contexto de Protocolos en Tiempo Real (IoT)**: El estudio de estos conceptos de seguridad, junto con protocolos de comunicación en tiempo real y ligeros como MQTT y WebSockets, es crucial para lograr la meta de desarrollar un **ecosistema IoT que sea fuerte, seguro, en tiempo real y escalable**. La seguridad de la comunicación es un pilar esencial cuando se trabaja con dispositivos y datos en el Internet de las Cosas.

- **Aplicación Práctica de los Conceptos de Seguridad (en secciones posteriores)**: Aunque los principios se estudian en la etapa 3, las secciones subsiguientes demuestran su aplicación práctica para asegurar la plataforma IoT:

  - **Asegurar un Dominio Personalizado con SSL/TLS**:
    - La **etapa 5** se centra en la aplicación de las 'terminologías de seguridad ' estudiadas en la etapa 3, específicamente para **asegurar un dominio personalizado recién creado con certificados SSL/TLS**.
    - Se utiliza **Let's Encrypt**, una autoridad de certificación gratuita, automatizada y de código abierto, respaldada por importantes patrocinadores.
    - El proceso implica:
      - Instalar software de terceros como `Certbot` en el servidor.
      - Configurar el certificado SSL para Apache utilizando el cliente `Certbot`.
      - Se pregunta al usuario si desea **redirigir el tráfico HTTP a HTTPS**, lo cual es una práctica recomendada y deseada.
      - Es fundamental **asignar reglas de seguridad de entrada para HTTPS (puerto 443)** en el servidor remoto de AWS, ya que, de lo contrario, la conexión podría fallar.
      - Los certificados de Let's Encrypt son **válidos por aproximadamente tres meses**, requiriendo una renovación periódica.
      - Una vez implementado, se verifica que el sitio web está **completamente seguro**, y que cada cliente conectado al servidor tendrá una **comunicación encriptada de extremo a extremo**. El servidor mostrará un candado verde en el navegador, indicando una conexión segura.
  - **Implementación de Inicio de Sesión Seguro y Gestión de Accesos**:
    - La **etapa 5** también incluye la implementación de una **funcionalidad de inicio de sesión de usuario seguro** y el almacenamiento de los detalles del usuario en una base de datos integrada.
    - En la **etapa 6**, se implementa una forma segura para que los usuarios y dispositivos IoT puedan conectarse de manera segura con el servidor IoT.
    - Esto se logra utilizando la funcionalidad de `Access Manager` de PubNub para permitir que los usuarios administradores otorguen **permisos de lectura y escritura en tiempo real** a usuarios no administradores y dispositivos.
    - La implementación implica generar una **clave de autorización para un usuario específico** y almacenarla en la base de datos. Las solicitudes para conceder acceso se envían al servidor de Flask, que luego llama al servidor PubNub para otorgar los permisos, asegurando que la solicitud provenga de un usuario administrador. Si el acceso es concedido, el canal se vuelve a suscribir para aplicar los cambios.

En síntesis, la etapa 3 proporciona la base teórica sobre la seguridad en Internet y la criptografía, presentando conceptos clave como el cifrado, las firmas digitales y SSL/TLS. Estas bases son luego aplicadas y demostradas en las secciones posteriores del proyecto para asegurar los servidores IoT, los dominios web y la gestión de acceso de usuarios y dispositivos, lo cual es esencial para construir una solución IoT fiable y protegida.

ALGORITMOS (SIMÉTRICO, ASIMÉTRICO)

En el contexto más amplio de la **Seguridad en Internet y la Criptografía**, se indican que la **etapa 3: Protocolos de Comunicación y Seguridad** se dedica a un estudio en profundidad de estos temas, incluyendo específicamente los **Algoritmos de clave simétrica y asimétrica**. Estos se presentan como 'temas candentes ' dentro del ámbito de la seguridad en Internet y la criptografía. Aquí se dan los detalles de estos algoritmos:

- **Estudio Teórico Fundamental**:
  - La etapa 3 del proyecto aborda un estudio en profundidad de la **seguridad en Internet y la criptografía**, con el objetivo de comprender cómo construir un ecosistema IoT **fuerte, seguro, en tiempo real y escalable**.
  - Dentro de este estudio, se mencionan explícitamente los **algoritmos de clave simétrica y asimétrica** como componentes clave.
  - Aunque no se proporcionan una definición detallada o un análisis técnico de cada tipo de algoritmo, su inclusión subraya su importancia teórica para entender la base de la seguridad digital.
- **Conceptos Relacionados**: Los algoritmos simétricos y asimétricos se estudian junto con otros pilares de la criptografía y la seguridad en Internet, como:
  - **Cifrado (Encryption)**.
  - **Firma digital (Digital Signature)**.
  - **SSL/TLS**, también conocidos como protocolos HTTP seguros.
- **Aplicación Implícita en la Seguridad Práctica**: Aunque no se describen la implementación directa de estos algoritmos, los conceptos de seguridad que ellos sustentan se aplican en las secciones prácticas subsiguientes para asegurar la plataforma IoT:
  - **Asegurar dominios con SSL/TLS**: En la etapa 5, se aplican las 'terminologías de seguridad ' estudiadas en la etapa 3 para **asegurar un dominio personalizado con certificados SSL/TLS** utilizando Let's Encrypt. La implementación de SSL/TLS inherentemente depende de la criptografía asimétrica para el intercambio seguro de claves y de la criptografía simétrica para la encriptación eficiente de los datos de la comunicación una vez establecida la conexión. Esto asegura una **comunicación encriptada de extremo a extremo** para cada cliente conectado al servidor.
  - **Funcionalidad de inicio de sesión seguro**: La etapa 5 también involucra la implementación de una **funcionalidad de inicio de sesión de usuario seguro** y el almacenamiento de detalles de usuario en una base de datos integrada, lo que a menudo implica el uso de hashing (una forma de criptografía) para almacenar contraseñas de forma segura.
  - **Gestión de accesos en tiempo real**: En la etapa 6, se implementa una forma segura para que usuarios y dispositivos IoT se conecten con el servidor IoT. Esto se realiza utilizando la funcionalidad de `Access Manager` de PubNub para otorgar permisos de lectura y escritura en tiempo real, lo que implica la **generación de una clave de autorización para usuarios específicos** y su gestión segura. Este proceso se basa en principios criptográficos para garantizar que solo los usuarios autorizados puedan realizar ciertas acciones.

En síntesis, los algoritmos simétricos y asimétricos son presentados como componentes conceptuales vitales en la etapa 3, proporcionando las bases teóricas necesarias para comprender cómo se construyen sistemas IoT seguros. Aunque no se detallan a nivel de código, su presencia es fundamental para las implementaciones prácticas de seguridad como SSL/TLS, el inicio de sesión de usuarios y la gestión de accesos, que se abordan en las secciones posteriores del proyecto.

CIFRADO

En el contexto más amplio de la **Seguridad en Internet y la Criptografía**, se destacan que el **cifrado (encryption)** es un tema fundamental y 'candente ' que se estudia en profundidad. Aquí se dan los detalles de el cifrado:

- **El Cifrado como Componente Clave de la Criptografía**:
  - La **etapa 3: Protocolos de Comunicación y Seguridad** se enfoca en un estudio en profundidad de la seguridad en Internet y la criptografía.
  - Dentro de este estudio, el **cifrado (encryption)** se menciona explícitamente como uno de los 'temas candentes ', junto con los algoritmos de clave simétrica y asimétrica, la firma digital y los protocolos SSL/TLS.
  - La comprensión del cifrado es esencial para construir un **ecosistema IoT fuerte, seguro, en tiempo real y escalable**.
- **El Cifrado y SSL/TLS**:
  - Se vincula directamente el cifrado con los **protocolos SSL/TLS**, mencionando que estos son 'también conocidos como protocolos HTTP '.
  - La **etapa 5** del proyecto está dedicada al desarrollo de las terminologías de seguridad estudiadas en la etapa 3. Un objetivo principal es **asegurar un dominio personalizado recién creado con certificados SSL/TLS**.
  - La implementación de SSL/TLS se lleva a cabo utilizando **Let's Encrypt**, una autoridad de certificación gratuita, automatizada y de código abierto.
- **Implementación Práctica del Cifrado a través de SSL/TLS**:
  - Para asegurar un sitio web, se instala software de terceros como `Certbot` en el servidor.
  - Luego, se configura el certificado SSL para Apache utilizando el cliente `Certbot`, proporcionando el nombre de dominio y una dirección de correo electrónico para recuperación.
  - Una acción crucial es redirigir el tráfico HTTP a HTTPS, lo cual es altamente deseado para garantizar la seguridad.
  - Es imperativo **asignar reglas de seguridad de entrada para HTTPS (puerto 443)** en el servidor remoto (por ejemplo, AWS), ya que de lo contrario la conexión fallaría.
  - Una vez implementado, se puede verificar el estado del certificado SSL a través de herramientas como `ssllabs.com`. El certificado es emitido por Let's Encrypt y tiene una validez de aproximadamente tres meses, lo que requiere renovaciones periódicas.
  - El resultado final es que el sitio web se considera **'completamente seguro '**, y cada cliente conectado al servidor tendrá una **'comunicación encriptada de extremo a extremo '**. El navegador mostrará un 'candado verde ' indicando una conexión segura.
- **Contexto en la Plataforma IoT**:
  - La implementación del cifrado, especialmente a través de SSL/TLS, es un paso fundamental para asegurar los servidores IoT.
  - Este enfoque garantiza que la comunicación entre los clientes (usuarios o dispositivos IoT) y el servidor sea privada y segura, protegiendo los datos que se transmiten.

En resumen, se enfatizan que el **cifrado** es un pilar fundamental de la seguridad en Internet y la criptografía, enseñado teóricamente en la etapa 3 y aplicado directamente a través de **SSL/TLS** en la etapa 5 para garantizar una **comunicación encriptada de extremo a extremo** en la plataforma IoT.

FIRMA DIGITAL

En el contexto más amplio de la **Seguridad en Internet y la Criptografía**, se indican que la **firma digital (digital signature)** es un tema fundamental y 'candente ' que forma parte de un estudio en profundidad. Aquí se dan los detalles de la firma digital:

- **Tema Candente de la Criptografía y Seguridad en Internet**:
  - La **etapa 3: Protocolos de Comunicación y Seguridad** (aunque se menciona inicialmente etapa 4, otra clarifica que es etapa 3 para el estudio teórico) se dedica a un estudio en profundidad de la seguridad en Internet y la criptografía.

    - Dentro de este estudio, la **firma digital** se menciona explícitamente como uno de los 'temas candentes '.
    - Este estudio es crucial para construir un **ecosistema IoT fuerte, seguro, en tiempo real y escalable**.
- **Contexto con Otros Conceptos de Seguridad**: La firma digital se presenta junto con otros pilares de la criptografía y la seguridad en Internet, tales como:
    - Algoritmos de clave simétrica y asimétrica.
    - Cifrado (Encryption).
    - SSL/TLS (también conocidos como protocolos HTTP).
- **Ausencia de Detalles Específicos sobre su Implementación**: Es importante señalar que, si bien se identifican la firma digital como un tema clave a estudiar, **no proporcionan detalles adicionales** sobre qué es una firma digital, cómo funciona, o cómo se implementa en la práctica dentro del proyecto IoT descrito. La mención se limita a su inclusión como un concepto teórico relevante dentro del ámbito de la seguridad en Internet y la criptografía.

En resumen, se establece que la **firma digital** es un concepto teórico importante en la seguridad en Internet y la criptografía, abordado en la etapa 3 del proyecto. Se la presenta como parte del conocimiento fundamental necesario para desarrollar soluciones IoT seguras, aunque no se ofrecen detalles específicos sobre su funcionamiento o aplicación práctica en los extractos proporcionados.

SSL/TLS (HTTPS)

En el contexto más amplio de la **Seguridad en Internet y la Criptografía**, se enfatizan que **SSL/TLS (HTTPS)** es un protocolo fundamental, un 'tema candente ', y una implementación práctica crucial para asegurar la comunicación en línea. Aquí se dan los detalles de SSL/TLS:

- **Identificación como 'Protocolos HTTP Seguros ' y Tema Candente**:
    - La **etapa 3: Protocolos de Comunicación y Seguridad** del proyecto se dedica a un estudio en profundidad de la seguridad en Internet y la criptografía.
    - Dentro de esta etapa, **SSL/TLS** se menciona explícitamente como uno de los 'temas candentes ', también conocido como **protocolos HTTP seguros**.
    - Su estudio es esencial para comprender cómo construir un **ecosistema IoT fuerte, seguro, en tiempo real y escalable**.
- **Implementación Práctica en la Seguridad del Servidor IoT**:
    - La **etapa 5** del proyecto se centra específicamente en el desarrollo de las terminologías de seguridad estudiadas en la etapa 3, y el primer video de esta etapa trata sobre **'secure HTTP IOT server and user login '**.
    - El objetivo principal es **asegurar un dominio personalizado** recién creado con **certificados SSL/TLS**.
- **Uso de Let's Encrypt como Autoridad de Certificación**:
    - Para obtener el certificado SSL, se utiliza **Let's Encrypt**, una autoridad de certificación **gratuita, automatizada y de código abierto**.
    - Es respaldada por importantes patrocinadores y utilizada por muchos desarrolladores y empresas.
- **Proceso de Configuración y Aseguramiento**:
    1. **Instalación de** `Certbot`: Se instala software de terceros como `Certbot` en el servidor. Esto se hace agregando el repositorio de Certbot (`sudo add_apt_repository ppa:certbot/certbot`), actualizando la lista de paquetes (`sudo apt-get update`) e instalando `python_certbot_apache` (`sudo apt-get install python_certbot_apache`).

2. **Configuración del Certificado SSL para Apache**: Se ejecuta el cliente Certbot `sudo certbot --apache -d` proporcionando el nombre de dominio (con y sin `www`) y una dirección de correo electrónico para recuperación de clave.
3. **Redirección de HTTP a HTTPS**: Se pregunta si se desea **redirigir el tráfico HTTP a HTTPS**, lo cual es enfáticamente deseado y se selecciona esta opción.
4. **Configuración de Reglas de Seguridad de Entrada (Inbound Security Rules)**: Un paso crítico es **asignar reglas de seguridad de entrada para HTTPS (puerto 443)** en el servidor remoto (ej. AWS EC2), ya que la conexión fallaría sin esto. Inicialmente, solo se tienen reglas para HTTP y SSH, por lo que se debe añadir HTTPS.
5. **Verificación del Certificado**: Una vez completada la instalación, los archivos del certificado se encuentran en `/etc/letsencrypt/live`. El estado del certificado SSL se puede verificar utilizando herramientas como SSL Labs (`ssllabs.com/ssltest`).
6. **Validación y Renovación**: El certificado emitido por Let's Encrypt es válido por **aproximadamente tres meses**, requiriendo **renovaciones periódicas**.

- **Resultados y Beneficios del SSL/TLS (HTTPS)**:
  - El sitio web se considera **'completamente seguro '**.
  - **Cada cliente conectado al servidor tendrá una comunicación encriptada de extremo a extremo**.
  - El navegador mostrará un **'candado verde '** (green lock) indicando una conexión segura.
  - El certificado se emite a nombre del servidor (ej. 'pact IOT server ') y es firmado por Let's Encrypt.

En síntesis, se posicionan a **SSL/TLS** como un concepto teórico crucial en la criptografía y la seguridad en Internet (etapa 3), que se traduce directamente en una **implementación práctica esencial** para garantizar la seguridad de la plataforma IoT (etapa 5). Su uso asegura la **confidencialidad e integridad de la comunicación** entre el servidor y sus clientes, transformando el tráfico HTTP no seguro en HTTPS encriptado y verificable.

2.4. Reconstrucción del Proyecto y Despliegue en la NUBE

En el contexto más amplio de las **Secciones del Curso**, se describen la **etapa 4: Reconstrucción del Proyecto y Despliegue en la Nube** como una fase fundamental para mejorar y escalar el proyecto IoT existente. Aquí se dan los detalles de esta etapa:

- **Reconstrucción del Proyecto Principal**: En la etapa 4, el proyecto previamente desarrollado se **reconstruirá**. Esto implica una revisión y modificación significativa de la arquitectura de comunicación.
- **Transición a PubNub para Comunicación**: El cambio principal en esta reconstrucción es la adopción de **PubNub como el principal protocolo de comunicación**. Esto reemplaza la técnica de 'AJAX long polling' que se había utilizado anteriormente. PubNub es una de las tecnologías que se estudian en profundidad en el proyecto.
- **Despliegue del Servidor IoT en la Nube**: Además de la reconstrucción del protocolo de comunicación, la etapa 4 también cubre cómo **desplegar el servidor IoT en la nube de AWS (Amazon Web Services)**. Esto es crucial para la escalabilidad y accesibilidad del ecosistema IoT.

En resumen, la **etapa 4** se centra en la **evolución del proyecto IoT** desde una implementación inicial a una solución más robusta y escalable. Introduce **PubNub** como una tecnología clave para la comunicación en tiempo real y enseña los pasos para el **despliegue en la infraestructura de la nube de AWS**, sentando las bases para un 'ecosistema IoT fuerte, seguro, en tiempo real y escalable ' como se menciona en el contexto general del proyecto.

2.4.1. Uso de PubNub Como Protocolo Principal (en lugar de AJAX)

En el contexto más amplio de la **etapa 4: Reconstrucción del Proyecto y Despliegue en la Nube**, se destacan el **uso de PubNub como protocolo de comunicación principal** como una mejora fundamental, que **reemplaza la técnica de AJAX long polling** utilizada anteriormente. Aquí se dan los detalles de este cambio:

- **Problemas con AJAX en la Comunicación IoT (Implícito)**:
  - En la etapa 2, el proyecto inicial utiliza la técnica de **AJAX para la comunicación** entre el cliente y el servidor.
  - Se menciona que, después de discutir las 'ventajas y desventajas ' de la forma en que se desarrolló el proyecto anterior en la etapa 2, se pasa a estudiar protocolos de comunicación en tiempo real y ligeros en la etapa 3, lo que sugiere que AJAX tiene limitaciones para un ecosistema IoT avanzado. Específicamente, AJAX *long polling* se menciona como la técnica a reemplazar.
- **Introducción y Estudio de PubNub**:
  - PubNub se identifica como una de las **principales tecnologías** con las que se trabajará en el proyecto, junto con Python, Raspberry Pi, Flask y AWS.
  - En la etapa 3, se realiza un estudio en profundidad de protocolos de comunicación en tiempo real y ligeros para IoT, que incluyen MQTT, WebSockets, y finalmente, se presenta una **demostración práctica con PubNub**. Esto prepara el terreno para su adopción posterior.
- **Reconstrucción del Proyecto en la etapa 4**:
  - La **etapa 4** se centra específicamente en **reconstruir el proyecto principal** para **utilizar PubNub como el principal protocolo de comunicación**, en lugar de AJAX *long polling*.
  - Este cambio es una parte integral de la evolución del proyecto, buscando mejorar su arquitectura de comunicación.
- **Beneficios del Cambio a PubNub**:
  - La adopción de PubNub como protocolo principal, junto con el despliegue en la nube de AWS (también cubierto en la etapa 4), contribuye a la creación de un **'ecosistema IoT fuerte, seguro, en tiempo real y escalable '**.
  - Aunque no se detallan directamente las *ventajas específicas* de PubNub sobre AJAX en este contexto, la implicación es que PubNub ofrece una solución más adecuada para la comunicación en tiempo real y bidireccional que requiere un sistema IoT moderno, superando las limitaciones de las técnicas de polling como AJAX. Otros protocolos estudiados como WebSockets y MQTT se describen como bidireccionales y eficientes en ancho de banda, lo que sugiere las características buscadas con la adopción de PubNub.

En resumen, la **etapa 4** marca un punto de inflexión en el desarrollo del proyecto IoT, donde el **cambio de AJAX *long polling* a PubNub como protocolo de comunicación principal** es una decisión clave para modernizar y mejorar el sistema, preparándolo para ser desplegado en la nube y logrando un ecosistema IoT más robusto y escalable.

2.4.2. Despliegue del Servidor IoT en AWS

En el contexto más amplio de la **etapa 4: Reconstrucción del Proyecto y Despliegue en la Nube**, se indican que el **despliegue del servidor IoT en AWS (Amazon Web Services)** es una etapa crucial para escalar y mejorar la infraestructura del proyecto. Aquí se dan los detalles de este aspecto:

- **Ubicación en el Curso**:
  - La **etapa 4** del proyecto se dedica explícitamente a la **reconstrucción del proyecto principal** y a enseñar cómo **desplegar el servidor IoT en la nube de AWS**.
- **Propósito del Despliegue en la Nube**:

- Este despliegue es un componente clave para construir un **'ecosistema IoT fuerte, seguro, en tiempo real y escalable '**. Al mover el servidor a la nube, se superan las limitaciones de ejecutarlo localmente (como se hacía inicialmente en la etapa 2), permitiendo un acceso y una gestión más robustos y globales.

- **Tecnologías Involucradas**:
  - **AWS** se menciona como una de las principales tecnologías con las que se trabajará en el proyecto, junto con Python, Raspberry Pi, Flask y PubNub. Esto subraya su importancia como pilar tecnológico en la plataforma IoT desarrollada.

- **Detalles de Implementación (implícitos y futuros)**:
  - Mientras que la etapa 4 introduce el concepto de desplegar en AWS, los detalles más específicos relacionados con la **seguridad del servidor en la nube**, como la asignación de reglas de seguridad de entrada para HTTPS (puerto 443) en una instancia EC2 de AWS, se abordan en la etapa 5. Esto sugiere que la etapa 4 sienta las bases para el despliegue, y las secciones posteriores construyen sobre esa infraestructura para añadir seguridad.

En resumen, la **etapa 4** es el punto en el que el proyecto IoT existente evoluciona de una configuración local a una **infraestructura basada en la nube de AWS**. Este paso es esencial para la **escalabilidad, accesibilidad y robustez** del 'ecosistema IoT ' propuesto, aunque los aspectos de seguridad más detallados se profundizan en secciones posteriores del proyecto.

2.5. Desarrollo de Seguridad

En el contexto más amplio de las **Secciones del Curso**, la **etapa 5: Desarrollo de Seguridad** se presenta como una fase crucial y directamente práctica donde se implementan los conceptos de seguridad e criptografía estudiados previamente. Es el punto donde el proyecto IoT previamente reconstruido y desplegado en la nube se fortalece con medidas de seguridad esenciales. Aquí se dan los detalles de esta etapa:

- **Implementación de Terminologías de Seguridad**:
  - La etapa 5 está diseñada para involucrar principalmente el **desarrollo de todas las terminologías de seguridad** que se discutieron y estudiaron en la etapa 3. Esto significa que los conceptos teóricos de seguridad en Internet y criptografía (como SSL/TLS, algoritmos de clave simétrica y asimétrica, cifrado, firma digital) encuentran su aplicación práctica aquí.

- **Aseguramiento del Dominio Personalizado con SSL/TLS**:
  - El **primer paso** en la etapa 5 es **obtener nuestro propio nombre de dominio personalizado** y luego **asegurar ese nombre de dominio** utilizando la autoridad de certificación Let's Encrypt.
  - Esto implica un proceso detallado que incluye:
    - **Instalar el software de terceros** `Certbot` en el servidor.
    - **Actualizar la lista de paquetes** e instalar `python-certbot-apache`.
    - **Configurar el certificado SSL para Apache** utilizando `Certbot`, proporcionando el nombre de dominio y una dirección de correo

- **Funcionalidad de Inicio de Sesión Seguro y Gestión de Usuarios**:
  - Posteriormente, la etapa 5 se enfoca en **implementar una funcionalidad segura de inicio de sesión de usuario** y el **almacenamiento de los detalles del usuario** en una base de datos integrada.
  - Se implementa una manera segura en la que los **usuarios y los dispositivos IoT pueden conectarse de forma segura** a nuestro servidor IoT.
  - Un aspecto clave es la **creación de reglas para usuarios administradores y no administradores**.

- El panel de control (donde se gestionan los permisos) solo es visible para los usuarios administradores.
- Se muestra una lista de usuarios en línea en el panel de administración, donde se pueden otorgar o revocar permisos de lectura y escritura.
- La lógica del servidor maneja la población de esta lista de usuarios, incluyendo sus IDs y estados de permisos (leído/no leído, escrito/no escrito).

- **Gestión de Accesos en Tiempo Real con PubNub Access Manager**:
  - Se utiliza la funcionalidad **PubNub Access Manager** para permitir a los usuarios administradores **conceder en tiempo real acceso de lectura y escritura** a todos los usuarios no administradores y a los dispositivos.
  - Cuando un administrador utiliza el botón 'aplicar ', se envía una solicitud al servidor Flask para actualizar los permisos de un usuario específico.
  - El servidor verifica que la solicitud provenga de un usuario administrador y luego **almacena los permisos de lectura y escritura del usuario en la base de datos**.
  - Finalmente, se realiza una llamada al **servidor PubNub para otorgar o denegar el acceso de lectura y escritura** al usuario específico.
  - Antes de esto, se requiere **generar una clave de autorización** para el usuario específico y almacenarla en la base de datos.

En resumen, la **etapa 5** es la culminación práctica de los principios de seguridad. Se pasa de un servidor local a uno desplegado en la nube con un **dominio personalizado asegurado por SSL/TLS**, y se construye una **sólida capa de gestión de usuarios y autenticación/autorización**. La integración de **PubNub Access Manager** subraya el compromiso del proyecto con la **seguridad en tiempo real y la escalabilidad** dentro del 'ecosistema IoT fuerte, seguro, en tiempo real y escalable' que se busca construir.

2.5.1. Dominio Personalizado Seguro con SSL/TLS (Encriptación)

En el contexto más amplio de la **etapa 5: Desarrollo de Seguridad**, el aseguramiento de un **Dominio Personalizado con SSL/TLS utilizando Let's Encrypt** es el paso fundamental inicial para establecer una base segura para el servidor IoT. Esta fase traduce los conceptos teóricos de seguridad estudiados previamente en una implementación práctica, garantizando que el 'ecosistema IoT ' sea robusto y confiable. Se detalla este proceso de la siguiente manera:

- **Propósito y Ubicación en la etapa 5**:
  - La etapa 5 comienza con la tarea de **asegurar un dominio personalizado recién creado con certificados SSL/TLS de Let's Encrypt**.
  - Este es el **primer paso** en el desarrollo de la seguridad, involucrando la aplicación práctica de 'todas las terminologías de seguridad' discutidas en la etapa 3.
  - El objetivo es que **cada cliente conectado al servidor tenga una comunicación cifrada de extremo a extremo**.

- **Let's Encrypt como Autoridad de Certificación**:
  - Let's Encrypt es la autoridad de certificación elegida, descrita como **'gratuita, automatizada y de código abierto '**. Es ampliamente utilizada por desarrolladores y empresas y cuenta con el respaldo de importantes patrocinadores.
  - Los certificados generados por Let's Encrypt son **válidos por aproximadamente tres meses** y requieren renovación después de su vencimiento.

- **Pasos para la Implementación de SSL/TLS**:
  1. **Instalación de** `Certbot`: El primer paso técnico es **instalar software de terceros,** `Certbot`, en el servidor. Esto implica agregar el repositorio de Certbot (`sudo add-apt-repository ppa:certbot/certbo`

2. **Actualización e Instalación del Paquete**: Posteriormente, se debe **actualizar la lista de paquetes** (`sudo apt-get update`) y **finalmente instalar** `python-certbot-apache` desde el nuevo repositorio.
3. **Configuración del Certificado SSL para Apache**: Una vez instalado `Certbot`, se ejecuta el cliente para **configurar el certificado SSL para Apache**, usando el comando `sudo certbot --apache -d [nombre-de-dominio]` y añadiendo también el subdominio `www`. Se requiere una dirección de correo electrónico para la recuperación de claves y se pregunta si se desea **redirigir el tráfico HTTP a HTTPS**, lo cual se debe aceptar. Los archivos de certificado se guardan en `/etc/letsencrypt/`.
4. **Configuración de Reglas de Seguridad en AWS**: Un paso crítico, a menudo pasado por alto, es **asignar reglas de seguridad de entrada para HTTPS (puerto 443) en el servidor remoto de AWS (instancia EC2)**. Sin esta regla, el servidor no sería accesible a través de HTTPS, lo que resultaría en un fallo en la verificación del certificado. Se debe editar la configuración de entrada para agregar esta regla.

- **Verificación y Resultado**:
  - El estado del certificado SSL puede **verificarse utilizando SSL Labs** (`ssllabs.com/ssltest`).
  - Una vez que todos los pasos se han completado correctamente, el resultado es que al visitar el dominio personalizado del servidor IoT, se produce una **redirección de HTTP a HTTPS, y el navegador muestra un 'candado verde ' (green lock)**, confirmando que el sitio es seguro y que la comunicación está cifrada. Los detalles del certificado mostrarán que fue emitido a 'pact IOT server ' por la autoridad de Let's Encrypt.

En resumen, la etapa 5 establece un **dominio web seguro con SSL/TLS** como el pilar fundamental del 'Desarrollo de Seguridad', garantizando una **comunicación cifrada de extremo a extremo** y sentando las bases para funcionalidades de seguridad más avanzadas, como el inicio de sesión de usuarios y la gestión de accesos.

2.5.2. Funcionalidad Segura de Inicio de Sesión de Usuario

En el contexto más amplio de la **etapa 5: Desarrollo de Seguridad**, la **Funcionalidad Segura de Inicio de Sesión de Usuario** es un componente esencial que sigue a la securización del dominio con SSL/TLS. Esta funcionalidad transforma el servidor IoT en una plataforma multiusuario donde los dispositivos y usuarios pueden interactuar de forma segura y controlada. Se detallan los siguientes aspectos clave de esta funcionalidad:

- **Propósito General**:
  - La etapa 5 se enfoca en la implementación de una **funcionalidad segura de inicio de sesión de usuario** y el **almacenamiento de los detalles del usuario en una base de datos integrada**.
  - El objetivo es permitir que **múltiples usuarios puedan iniciar sesión de forma segura y controlar y monitorear sus dispositivos autorizados en tiempo real**.
- **Implementación de Roles de Usuario (Administrador y No Administrador)**:
  - Se crean **reglas para usuarios administradores y no administradores**.
  - El **panel de control del servidor, que muestra la lista de usuarios en línea y permite gestionar permisos, es visible exclusivamente para los usuarios administradores**.
  - Para implementar esto, se utiliza una sentencia `if` en el código HTML (`index.html`) que verifica si el `user ID` de la sesión coincide con el `user ID` del administrador antes de mostrar el panel de control. Esto asegura que los usuarios no administradores no tengan acceso a esta interfaz.
  - Un ejemplo práctico muestra que un usuario no administrador, al iniciar sesión, no obtiene acceso al panel de control de acceso, mientras que el administrador sí puede ver la lista de usuarios en línea.
- **Visualización y Gestión de Usuarios en el Panel de Administración**:

- El panel de administración muestra una **lista de usuarios en línea**.
- Junto al nombre de cada usuario en línea, hay **botones de conmutación (switch buttons) para otorgar permisos de lectura y escritura**.
- Un botón de 'aplicar ' permite guardar los cambios de permisos.
- La interfaz (`index.html`) está diseñada con secciones `div` y un panel Bootstrap que contiene un encabezado para 'Online Users' y una tabla (`li` dentro de `ul`) para la lista de usuarios. Cada fila muestra el nombre del usuario y los botones de lectura/escritura.

- **Población Dinámica de la Lista de Usuarios**:
  - La lógica en el lado del servidor (en `my DB`) recupera los usuarios conectados y sus detalles, incluyendo el ID de usuario, el nombre, y los estados de permisos de lectura y escritura.
  - Estos datos se almacenan en un mapa (`online_user_records`) y se envían a la página web principal (`index.html`) utilizando plantillas Jinja. Se utiliza un bucle `for` en HTML para generar dinámicamente las filas de la tabla para cada usuario en línea, mostrando su nombre, ID de usuario y el estado 'checked' o 'unchecked ' de los botones de permiso según los datos del servidor.

- **Manejo Seguro de Permisos (Lectura/Escritura)**:
  - Cuando un administrador utiliza el botón 'aplicar ', se envía una **solicitud POST desde el código JavaScript (`main.js`) al servidor Flask**. Esta solicitud incluye el ID de usuario, y los estados de lectura y escritura.
  - El servidor Flask, al recibir la solicitud, realiza una **verificación crucial para asegurar que la solicitud proviene de un usuario administrador**. Si no es así, se deniega el acceso.
  - Si la solicitud es válida (proviene de un administrador), se realizan dos acciones fundamentales:
    1. **Almacenar los permisos de lectura y escritura del usuario en la base de datos**.
    2. **Realizar una llamada al servidor PubNub para otorgar o denegar el acceso de lectura y escritura a este usuario específico**.
  - **Generación de Clave de Autorización**: Antes de poder conceder permisos de lectura y escritura, es necesario **generar una clave de autorización para cada usuario específico y almacenarla en la base de datos**. Esta es una 'primera etapa ' esencial antes de conceder permisos de lectura y escritura.

- **Integración con PubNub Access Manager**:
  - Se utiliza la funcionalidad de **PubNub Access Manager** para permitir a los usuarios administradores **conceder en tiempo real acceso de lectura y escritura** a todos los usuarios no administradores y a los dispositivos.

En síntesis, la **Funcionalidad Segura de Inicio de Sesión de Usuario** en la etapa 5 abarca no solo la autenticación inicial sino también un robusto sistema de gestión de acceso y roles. Esto incluye la visualización de usuarios en línea, la definición clara de roles (administrador/no administrador), la gestión de permisos en tiempo real a través de una interfaz segura y la integración con una base de datos y PubNub para un control de acceso granular y escalable en el ecosistema IoT.

2.5.3. Almacenamiento de Detalles de Usuario en DB Integrada

En el contexto más amplio de la **etapa 5: Desarrollo de Seguridad**, el **Almacenamiento de Detalles de Usuario en DB Integrada** es una parte fundamental, que se implementa después de asegurar el dominio con SSL/TLS. Este proceso es clave para habilitar una **funcionalidad segura de inicio de sesión de usuario** y para gestionar el acceso de múltiples usuarios a los dispositivos IoT de forma controlada y segura. Se describe este aspecto de la siguiente manera:

- **Propósito Fundamental en la etapa 5**:
  - La etapa 5 incluye la implementación de una funcionalidad segura de inicio de sesión de usuario y el **almacenamiento de los detalles del usuario en una base de datos integrada**. Esto permite que múltiples usuarios puedan iniciar sesión de forma segura, controlar y monitorear sus dispositivos autorizados en tiempo real.

- Este almacenamiento es crucial para el desarrollo de todas las terminologías de seguridad discutidas en la etapa 3, aplicándolas a un ecosistema IoT.

- **Detalles Específicos del Usuario Almacenados**:
  - La base de datos (referida como `my DB`) es utilizada para almacenar y gestionar diversos detalles de los usuarios.
  - Se utiliza un mapa (`online_user_records`) que se **rellena con el nombre del usuario (índice 0), el ID del usuario (índice 1), el estado de permiso de lectura (índice 2) y el estado de permiso de escritura (índice 3)**.
  - Estos estados de lectura y escritura (`read` y `write`) se transforman en 'checked' o 'unchecked ' para la interfaz HTML, dependiendo de si el acceso es `1` (concedido) o `0` (denegado).
  - Antes de conceder permisos de lectura y escritura, un **'primer paso ' esencial es generar una clave de autorización (`authorization key`) para cada usuario específico y almacenarla en la base de datos**.

- **Mecanismo de Almacenamiento y Recuperación**:
  - Cuando se envían datos a la página principal (`index.html`), se incluyen detalles adicionales como el `user_ID` de la sesión y la **lista de usuarios en línea obtenida de la función `get all logged in users` de `my DB`**.
  - La lista de usuarios en línea se construye en el lado del servidor, recuperando información de la base de datos y organizándola en un formato que puede ser utilizado por las plantillas Jinja en el HTML para poblar dinámicamente la interfaz.
  - Cuando un administrador utiliza el botón 'aplicar ' para cambiar permisos, se envía una solicitud POST al servidor Flask. Si la solicitud es válida (proviene de un administrador), el servidor realiza dos acciones clave:
    1. **Almacenar los permisos de lectura y escritura del usuario en la base de datos**.
    2. Realizar una llamada al servidor PubNub para otorgar o denegar el acceso de lectura y escritura a ese usuario.

- **Rol en la Gestión de Acceso y Roles (Admin/No-Admin)**:
  - La información almacenada en la base de datos es fundamental para diferenciar entre usuarios administradores y no administradores.
  - El **panel de control, que muestra la lista de usuarios en línea y permite gestionar sus permisos, solo es visible para los usuarios administradores**, utilizando una sentencia `if` que verifica el `user ID` de la sesión contra el `user ID` del administrador.
  - Los botones de conmutación para los permisos de lectura y escritura, así como el botón 'aplicar ', interactúan directamente con la base de datos para persistir los cambios.

En síntesis, el **Almacenamiento de Detalles de Usuario en la DB Integrada** es la columna vertebral que soporta la gestión de usuarios, roles y permisos dentro de la etapa 5. Permite un control granular sobre quién puede acceder y qué acciones puede realizar en el ecosistema IoT, asegurando que la comunicación y la interacción con los dispositivos sean tanto seguras como eficientes.

2.6. Conexión Segura de Usuarios Y Dispositivos

En el contexto más amplio de las **Secciones del Curso**, la **etapa 6: Conexión Segura de Usuarios y Dispositivos** se posiciona como una etapa crucial para establecer un ecosistema IoT robusto y controlado, construyendo sobre los fundamentos de seguridad y gestión de usuarios establecidos en secciones anteriores. Se detalla lo siguiente sobre esta etapa:

- **Propósito Principal**: La etapa 6 se centrará en la implementación de una **manera segura para que los usuarios y los dispositivos IoT puedan conectarse de forma segura con el servidor IoT**. Esto es fundamental para garantizar que solo las entidades autorizadas puedan interactuar con la plataforma y sus dispositivos.

- **Funcionalidad Clave**: Para lograr esta conexión segura y la gestión de permisos, la etapa utilizará la **funcionalidad PubNub Access Manager**. Esta herramienta permitirá a los usuarios administradores (admins) **otorgar acceso de lectura y escritura en tiempo real** a todos los usuarios no administradores y a los propios dispositivos.

- **Relación con Secciones Anteriores**:

  - **Fundamentos de Seguridad (etapa 3)**: La etapa 6 implica el desarrollo de todas las terminologías de seguridad que se discutieron y estudiaron en la etapa 3. Esto sugiere que los principios de seguridad y criptografía (mencionados en la etapa 3 como temas como cifrado, firmas digitales y protocolos SSL/TLS) son aplicados en la práctica en esta etapa.
  - **Desarrollo de Seguridad (etapa 5)**: La etapa 6 sigue directamente a la etapa 5. En la etapa 5, se establecen las bases con la obtención de un nombre de dominio personalizado, su aseguramiento con certificados SSL/TLS (Let's Encrypt) y la implementación de una funcionalidad segura de inicio de sesión de usuario con el almacenamiento de los detalles del usuario en una base de datos integrada. La interfaz de administración que permite a los usuarios administradores ver la lista de usuarios en línea y conceder permisos de lectura y escritura a través de botones de conmutación también se desarrolla en la etapa 5. Por lo tanto, la etapa 6 parece ser la implementación de la lógica detrás de estos permisos, utilizando PubNub.

- **Contexto en el Ecosistema IoT**: Al implementar una conexión segura y una gestión de acceso granular, la etapa 6 contribuye a la creación de un **ecosistema IoT fuerte, seguro, en tiempo real y escalable**. Esto es esencial para la plataforma en la que múltiples usuarios pueden iniciar sesión de forma segura y controlar y monitorear sus dispositivos autorizados en tiempo real.

- **Transición a Secciones Posteriores**: Una vez que se ha desarrollado este 'fuerte, seguro, en tiempo real y escalable ecosistema IoT', la etapa 7 se encarga de **agregar más sensores y actuadores para construir un caso de uso significativo para el mundo real**, como un sistema de monitoreo atmosférico. Esto significa que la funcionalidad de conexión segura y gestión de permisos de la etapa 6 es una base habilitadora para la expansión y aplicación práctica del sistema IoT.

En resumen, la etapa 6 es el punto donde la teoría de la seguridad y la gestión de usuarios converge para crear un mecanismo práctico y en tiempo real para controlar el acceso a los dispositivos y a la información en el ecosistema IoT, utilizando herramientas como PubNub Access Manager y construyendo sobre la infraestructura segura establecida en las secciones previas del proyecto.

2.6.1. Conexión Segura con Servidor IoT

En el contexto más amplio de las **Secciones del Curso**, la **etapa 6: Conexión Segura de Usuarios y Dispositivos** se dedica específicamente a establecer la **Conexión Segura con el Servidor IoT** para usuarios y dispositivos, construyendo sobre la infraestructura de seguridad establecida en las secciones previas. Se detalla los siguientes aspectos clave sobre la Conexión Segura con el Servidor IoT en la etapa 6:

- **Propósito Fundamental de la etapa 6**: El objetivo principal de esta etapa es implementar una **manera segura para que los usuarios y los dispositivos IoT puedan conectarse de forma segura con el servidor IoT**. Esto asegura que solo las entidades autorizadas puedan interactuar con la plataforma.

- **Tecnología Clave para la Conexión Segura**: Para lograr esta seguridad en la conexión y la gestión de permisos, la etapa 6 utiliza la **funcionalidad PubNub Access Manager**. Esta herramienta permite a los usuarios administradores (admins) **otorgar acceso de lectura y escritura en tiempo real** tanto a los usuarios no administradores como a los propios dispositivos.

- **Proceso para Conceder Permisos de Acceso**:

  - **Paso Inicial: Generación de Claves de Autorización**: Antes de conceder permisos de lectura y escritura, un **'primer paso' esencial es generar una clave de autorización (`authorization key`) para cada usuario específico y almacenarla en la base de datos**. Esta clave es fundamental para el proceso de autorización.

- **Mecanismo de Otorgamiento de Permisos**: Cuando un administrador utiliza un botón 'aplicar' en el panel de control (desarrollado en la etapa 5), se envía una solicitud al servidor Flask. Si la solicitud es válida (confirmando que proviene de un administrador), el servidor realiza dos acciones clave para la conexión segura:
  1. **Almacenar los permisos de lectura y escritura del usuario en la base de datos**.
  2. **Llamar al servidor PubNub para otorgar o denegar el acceso de lectura y escritura a ese usuario específico**.

- **Fundamentos de Seguridad Previos**: La Conexión Segura con el Servidor IoT en la etapa 6 se apoya en los desarrollos de secciones anteriores:
  - **Seguridad a Nivel de Servidor (etapa 5)**: Previamente, en la etapa 5, se asegura el servidor web mismo. Se obtiene un nombre de dominio personalizado y se protege con **certificados SSL/TLS (Let's Encrypt)**. Esto garantiza que **cualquier cliente conectado con el servidor tendrá una comunicación cifrada de extremo a extremo**. La redirección de HTTP a HTTPS, indicada por un 'candado verde' en el navegador, confirma que el sitio es seguro.
  - **Login de Usuario Seguro y DB Integrada (etapa 5)**: También en la etapa 5, se implementa una funcionalidad de inicio de sesión de usuario segura y el almacenamiento de los detalles del usuario en una base de datos integrada. Esta base de datos es utilizada para gestionar los permisos que luego serán aplicados por PubNub Access Manager. El panel de control para administrar estos permisos es visible solo para usuarios administradores.
  - **Terminologías de Seguridad (etapa 3)**: La etapa 6 implica el **desarrollo de todas las terminologías de seguridad** discutidas y estudiadas en la etapa 3, lo que sugiere que principios de criptografía, firmas digitales, SSL/TLS, etc., son aplicados en esta etapa.

- **Impacto en el Ecosistema IoT**: Al implementar esta conexión segura y la gestión de acceso granular, la etapa 6 contribuye a la creación de un **ecosistema IoT robusto, seguro, en tiempo real y escalable**. Esto es vital para una plataforma donde múltiples usuarios pueden iniciar sesión de forma segura, controlar y monitorear sus dispositivos autorizados en tiempo real.

En resumen, la Conexión Segura con el Servidor IoT en la etapa 6 es el resultado de asegurar la comunicación del servidor con SSL/TLS (etapa 5), implementar un sistema de gestión de usuarios y permisos (etapa 5), y luego utilizar el **PubNub Access Manager** para aplicar esos permisos en tiempo real, controlando el acceso de usuarios y dispositivos al servidor mediante la generación de claves de autorización y la llamada al servicio PubNub.

2.6.2. PubNub Access Manager (Usuarios Admin/No Admin)

En el contexto de la **etapa 6: Conexión Segura de Usuarios y Dispositivos**, el **PubNub Access Manager** es una funcionalidad clave y central para la implementación de un sistema de seguridad que permite la distinción y gestión de permisos entre usuarios administradores (admins) y no administradores al conectarse al servidor IoT. Aquí se dan los detalles de PubNub Access Manager, especialmente en relación con los roles de usuarios admin y no admin:

- **Propósito Fundamental en la etapa 6**: La etapa 6 tiene como objetivo implementar una **forma segura para que los usuarios y dispositivos IoT puedan conectarse de forma segura con el servidor IoT**. Para lograr esto, se utiliza la **funcionalidad PubNub Access Manager**, que permite a los usuarios administradores **otorgar acceso de lectura y escritura en tiempo real** tanto a los usuarios no administradores como a los propios dispositivos.

- **Capacidades de los Usuarios Administradores (Admins)**:
  - **Panel de Control Exclusivo**: Los usuarios administradores tienen acceso a un **panel de control (dashboard)** dedicado. Este panel es **visible exclusivamente para ellos**.
  - **Visualización de Usuarios en Línea**: En este panel, los administradores pueden ver una **lista de todos los usuarios en línea**. Cada fila de la tabla representa un usuario, mostrando su nombre de usuario.

- **Gestión de Permisos en Tiempo Real**: Junto al nombre de cada usuario en línea, el panel de control muestra **botones de conmutación (switch buttons)** para **conceder permisos de lectura (`read permissions`) y escritura (`write permissions`)**.
- **Aplicación de Cambios**: Un **botón 'aplicar ' (`apply button`)** permite a los administradores registrar y aplicar los cambios en los permisos en tiempo real.
- **Verificación del Rol**: Al procesar las solicitudes de cambio de permisos, el servidor Flask **verifica que la solicitud provenga de un usuario administrador**; de lo contrario, la solicitud es denegada con un mensaje de 'acceso denegado'.

- **Limitaciones de los Usuarios No Administradores**:
  - **Sin Acceso al Panel de Control**: Los usuarios que no son administradores **no tienen acceso al panel de control de acceso**. Un ejemplo claro muestra cómo un usuario no administrador no verá este panel.
  - **Permisos Otorgados por Admins**: Sus permisos de lectura y escritura son **gestionados y otorgados por los usuarios administradores** a través del PubNub Access Manager.
- **Mecanismo para Conceder Permisos con PubNub Access Manager**:
  1. **Generación de Clave de Autorización**: El **primer paso** antes de conceder permisos de lectura y escritura es **generar una clave de autorización (`authorization key`) para cada usuario específico y almacenarla en la base de datos**.
  2. **Interacción en el Dashboard**: Cuando un administrador manipula los botones de conmutación de lectura/escritura y presiona el botón 'aplicar ' en el dashboard, se envía una solicitud POST desde el código JavaScript (en `main.js`) a la aplicación Flask del servidor. Esta solicitud contiene el ID del usuario objetivo, así como el estado de los permisos de lectura y escritura.
  3. **Procesamiento en el Servidor Flask**:
     - El servidor Flask recibe la solicitud y, como se mencionó, **verifica que el remitente sea un usuario administrador**.
     - Si la solicitud es válida, el servidor realiza dos acciones cruciales:
       - **Almacena los permisos de lectura y escritura del usuario en la base de datos**.
       - **Realiza una llamada al servidor PubNub para otorgar o denegar el acceso de lectura y escritura a ese usuario específico**.
     - Después de que se concede el acceso, el cliente puede necesitar **reiniciar su suscripción al canal PubNub**.
- **Integración con Secciones Previas**: La implementación de esta gestión de usuarios y permisos mediante PubNub Access Manager se basa en las funcionalidades desarrolladas en secciones anteriores, como el **inicio de sesión de usuario seguro y el almacenamiento de los detalles del usuario en una base de datos integrada** (etapa 5). La etapa 6 implica la aplicación de todas las **terminologías de seguridad** estudiadas en la etapa 3.

En definitiva, el **PubNub Access Manager** es la herramienta clave que permite a los usuarios administradores tener un **control granular y en tiempo real** sobre el acceso de otros usuarios y dispositivos al servidor IoT. Esto asegura que solo las entidades autorizadas puedan interactuar con la plataforma, contribuyendo a la creación de un **ecosistema IoT robusto, seguro, en tiempo real y escalable**, donde múltiples usuarios pueden iniciar sesión de forma segura y controlar o monitorear solo sus dispositivos autorizados.

2.6.3. Panel de Control (Lista de Usuarios online, Permisos de Lectura/Escritura)

En el contexto más amplio de la **etapa 6: Conexión Segura de Usuarios y Dispositivos**, el **Panel de Control** es una interfaz esencial que implementa la funcionalidad del PubNub Access Manager, permitiendo a los usuarios administradores gestionar de forma segura y en tiempo real el acceso de otros usuarios y dispositivos al servidor IoT. Se proporcionan los siguientes detalles sobre el Panel de Control, las listas de usuarios online y los permisos de lectura/escritura:

- **Propósito y Visibilidad en la etapa 6**:
  - La etapa 6 se enfoca en implementar una **forma segura para que los usuarios y dispositivos IoT se conecten con el servidor IoT**, utilizando la funcionalidad del PubNub Access Manager.
  - El Panel de Control es una **etapa adicional en el dashboard** del servidor IoT, diseñada para gestionar estos permisos.
  - Es crucial destacar que **este panel de control es visible exclusivamente para los usuarios administradores**. Si un usuario no administrador intenta acceder, no lo verá.
- **Contenido y Componentes del Panel de Control**:
  - El panel se presenta como una nueva etapa debajo de las existentes en el dashboard.
  - Incluye una **lista de todos los usuarios online**.
  - Cada fila de la lista muestra el **nombre de un usuario en línea**.
  - Junto al nombre de cada usuario, hay **botones de conmutación (switch buttons)**. Uno para **otorgar permisos de lectura (`read permissions`)** y otro para **otorgar permisos de escritura (`write permissions`)**.
  - Los estados iniciales de estos botones (activado/desactivado) se determinan si el acceso de lectura o escritura es uno (activado) o cero (desactivado) en la base de datos.
  - También cuenta con un **botón 'aplicar ' (`apply button`)** para registrar y aplicar los cambios realizados en los permisos.
  - La estructura del panel utiliza un panel primario de Bootstrap con un encabezado para 'usuarios en línea ' y una clase de tabla como 'list group ', con cada fila definida por un elemento (`<li>`).
- **Funcionamiento de las Listas de Usuarios Online y Permisos**:
  1. **Población Dinámica de la Lista**:
     - El servidor Flask es el encargado de **proporcionar el ID de usuario y la lista de usuarios online** a la página HTML principal.
     - Una función `get all logged in users` en la base de datos retorna un mapa con la clave `online users` y un valor que es una lista de registros de usuarios online.
     - Cada registro incluye el nombre del usuario (índice 0), el ID del usuario (índice 1), y los estados de los permisos de lectura (índice 2) y escritura (índice 3).
     - La página `index.html` utiliza **plantillas Jinja** y un bucle `for` para iterar sobre la lista de usuarios online y crear dinámicamente una fila para cada uno, mostrando su nombre y configurando los botones de lectura/escritura según sus permisos actuales.
  2. **Modificación de Permisos por Administradores**:
     - Cuando un administrador interactúa con los botones de conmutación de lectura/escritura y presiona el botón 'aplicar', el código JavaScript (`main.js`) envía una **solicitud POST** a la aplicación Flask del servidor.
     - Esta solicitud (`send event`) contiene el ID del usuario objetivo, el estado de lectura y el estado de escritura.
     - En la aplicación Flask, se añade un endpoint (`/grant/user_id/read/write`) para recibir esta solicitud.
  3. **Validación y Aplicación en el Servidor**:
     - El servidor Flask primero **verifica que la solicitud provenga de un usuario administrador**. Si no es así, la respuesta es 'acceso denegado'.
     - Si la solicitud es válida (es decir, proviene de un administrador), el servidor realiza dos acciones cruciales:
       - **Almacena los nuevos permisos de lectura y escritura del usuario en la base de datos**.
       - **Realiza una llamada al servidor PubNub para otorgar o denegar el acceso de lectura y escritura a ese usuario específico**.

- Después de conceder el acceso, el cliente puede necesitar **reiniciar su suscripción al canal PubNub**.

4. **Paso Previo Requerido**: Antes de conceder estos permisos de lectura y escritura, el **primer paso es generar una clave de autorización (`authorization key`) para ese usuario específico y almacenarla en la base de datos**.

En resumen, el Panel de Control con sus listas de usuarios online y controles de permisos de lectura/escritura es la **interfaz clave del PubNub Access Manager**. Permite a los **administradores un control en tiempo real y granular** sobre quién puede leer o escribir datos en la plataforma IoT, lo que es fundamental para la **conexión segura de usuarios y dispositivos** descrita en la etapa 6, asegurando que solo los usuarios autorizados puedan interactuar con sus dispositivos.

2.7. Proyecto Final: Sistema de Monitoreo (Paciente Diabético/Nivel de Carga de Batería)

La **etapa 7: Proyecto Final: Sistema de Monitoreo Atmosférico** se presenta como la culminación del proyecto 'Internet de las Cosas con Python y Raspberry Pi ', integrando y expandiendo los conocimientos adquiridos en las secciones anteriores para construir una solución IoT completa y significativa. En el contexto más amplio de las **Secciones del Curso**, la etapa 7 se posiciona como el **proyecto final**, donde se aplica todo lo aprendido para desarrollar un sistema funcional. Aquí se dan los detalles de la etapa 7:

- **Propósito y Ubicación en el Curso**:
  - Es el **'último proyecto '** del proyecto, denominado 'Sistema de Monitoreo Atmosférico '.
  - Tiene como objetivo **añadir más sensores y actuadores** para construir algo significativo para un caso de uso del mundo real.
  - Representa la fase donde se desarrolla un **ecosistema IoT robusto, seguro, en tiempo real y escalable**, utilizando las bases sentadas en secciones previas, como la comunicación segura (etapa 6).
- **Nuevos Conceptos y Habilidades a Aprender**:
  - Durante esta etapa, los estudiantes se familiarizarán con **convertidores digitales**, la **interfaz periférica serie (SPI)**, y 'mucho más '. Esto sugiere la exploración de nuevas formas de interacción con hardware y adquisición de datos.
- **Producto Final y Características**:
  - El **producto final será un dashboard (panel de control) que mostrará gráficos visuales**.
  - Estos gráficos presentarán **datos en tiempo real de los sensores y actuadores**.
  - El objetivo es que este dashboard sea 'bastante agradable a la vista ' ('quite possible to the eyes ').
  - El sistema buscará ser una **solución IoT integral ('one-stop IOT solution ')**, abarcando desde la simplicidad a nivel de dispositivo hasta la complejidad de la infraestructura a nivel de la nube.
- **Escalabilidad y Aplicabilidad Futura**:
  - Al finalizar, los participantes serán capaces de **expandir fácilmente este proyecto**.
  - Podrán **añadir más dispositivos y funcionalidades** según sus necesidades empresariales.

En resumen, la etapa 7 actúa como el **gran proyecto integrador** del proyecto. Tras haber cubierto desde los fundamentos de IoT (etapa 1), pasando por la comunicación (etapa 2, 3 y 4) y, crucialmente, la seguridad y la gestión de acceso de usuarios (etapa 5 y 6), el 'Sistema de Monitoreo Atmosférico ' permite a los estudiantes aplicar todas estas habilidades para construir una solución IoT avanzada y con un **impacto en el mundo real**, culminando con un dashboard visual y en tiempo real.

2.7.1. Mas Sensores y Actuadores

En el contexto más amplio de la **etapa 7: Proyecto Final: Sistema de Monitoreo Atmosférico**, la adición de **'Más Sensores y Actuadores'** es un componente fundamental que permite transformar los conocimientos adquiridos en el proyecto en un sistema IoT completo y aplicable a un caso de uso real. Se detalla lo siguiente sobre la incorporación de más sensores y actuadores en esta etapa:

- **Propósito de Añadir Más Sensores y Actuadores**:
  - La etapa 7 se enfoca en el **'último proyecto '** del proyecto, denominado 'Sistema de Monitoreo Atmosférico '.
  - El objetivo principal es **'añadir más sensores y actuadores '** para construir algo **'significativo para un caso de uso del mundo real '**. Esto indica una progresión desde proyectos más básicos (como el detector de movimiento de la etapa 2, que utilizaba un sensor PIR y un zumbador) hacia una solución más compleja y funcional.
  - La meta es desarrollar un **'ecosistema IoT robusto, seguro, en tiempo real y escalable '**.
- **Nuevas Tecnologías Asociadas a los Sensores y Actuadores**:
  - Para la integración de estos nuevos dispositivos, los estudiantes se familiarizarán con **'convertidores digitales '** y la **'interfaz periférica serie (SPI) '**. Esto sugiere que los nuevos sensores y actuadores podrían requerir métodos de comunicación y procesamiento de datos más avanzados que los GPIO básicos usados anteriormente.
- **Resultados y Visualización de Datos de los Sensores/Actuadores**:
  - El producto final de la etapa 7 será un **'dashboard (panel de control) que mostrará gráficos visuales '**.
  - Estos gráficos presentarán **'datos en tiempo real de los sensores y actuadores '**.
  - El objetivo es que este dashboard sea 'bastante agradable a la vista '.
- **Escalabilidad del Proyecto Final**:
  - Al completar el proyecto, los participantes serán capaces de **'expandir fácilmente este proyecto '**, añadiendo **'más dispositivos y funcionalidades '** según sus necesidades empresariales. Esto subraya la flexibilidad de la arquitectura construida en la etapa 7 para integrar futuros sensores y actuadores.

En resumen, la adición de **'Más Sensores y Actuadores '** en la etapa 7 es un paso crucial para construir un **'Sistema de Monitoreo Atmosférico'** que simula una aplicación IoT del mundo real. Este proyecto final no solo implica la integración de hardware más complejo (requiriendo el aprendizaje de convertidores digitales y SPI), sino que también culmina en la visualización en tiempo real de los datos de estos sensores y el ontrol de los actuadores a través de un dashboard amigable, validando la capacidad de los estudiantes para desarrollar soluciones IoT escalables y completas.

2.7.2. Convertidores Digitales

En el contexto más amplio de la **etapa 7: Proyecto Final: Sistema de Monitoreo Atmosférico**, se indican que los **Convertidores Digitales** son una de las nuevas tecnologías y habilidades con las que los estudiantes se familiarizarán. Específicamente, se mencionan lo siguiente:

- **Introducción en la etapa 7**: La etapa 7, descrita como el 'último proyecto ' del proyecto, tiene como objetivo construir un 'Sistema de Monitoreo Atmosférico '. En esta etapa, los participantes se familiarizarán con 'los convertidores digitales, la interfaz periférica serie y mucho más '. Esto sugiere que la comprensión y el uso de convertidores digitales son esenciales para el desarrollo de este proyecto final.

- **Contexto de Nuevos Sensores y Actuadores**: La etapa 7 se centra en 'añadir más sensores y actuadores y construir algo significativo para un caso de uso del mundo real '. La mención de 'convertidores digitales ' junto con la 'interfaz periférica serie (SPI) ' implica que los nuevos sensores y actuadores que se incorporarán en este proyecto podrían ser de naturaleza más compleja o requerir una forma específica de interconexión con la Raspberry Pi, como la conversión de señales analógicas a digitales.
- **Desarrollo de un Ecosistema IoT Robusto**: El proyecto busca desarrollar un 'ecosistema IoT robusto, seguro, en tiempo real y escalable'. La introducción de convertidores digitales es fundamental para lograr esta robustez, ya que permite la integración de una gama más amplia de sensores que pueden generar datos analógicos, transformándolos en un formato digital procesable por los dispositivos IoT.

En resumen, los **Convertidores Digitales** son un concepto técnico clave introducido en la **etapa 7** que permitirá a los estudiantes interactuar con **sensores y actuadores más avanzados**. Su estudio y aplicación son fundamentales para la recolección de datos de manera efectiva en el 'Sistema de Monitoreo Atmosférico ', facilitando la creación de un dashboard que muestre 'datos en tiempo real de los sensores y actuadores' y expandiendo las capacidades del proyecto para incluir una diversidad de componentes del mundo real.

2.7.3. Iinterfaz Periférica Serial

En el contexto más amplio de la **etapa 7: Proyecto Final: Sistema de Monitoreo Atmosférico**, se indican que la **Interfaz Periférica Serial (SPI)** es una de las nuevas tecnologías y habilidades con las que los estudiantes se familiarizarán. Específicamente, se mencionan lo siguiente:

- **Introducción en la etapa 7**: La etapa 7, descrita como el 'último proyecto ' del proyecto, tiene como objetivo construir un 'Sistema de Monitoreo Atmosférico '. En esta etapa, los participantes se familiarizarán con 'los convertidores digitales, la interfaz periférica serie y mucho más '. Esto sugiere que la comprensión y el uso de la Interfaz Periférica Serial son esenciales para el desarrollo de este proyecto final.
- **Contexto de Nuevos Sensores y Actuadores**: La etapa 7 se centra en 'añadir más sensores y actuadores y construir algo significativo para un caso de uso del mundo real '. La mención de 'interfaz periférica serie (SPI) ' junto con los 'convertidores digitales ' implica que los nuevos sensores y actuadores que se incorporarán en este proyecto podrían ser de naturaleza más compleja o requerir una forma específica de interconexión con la Raspberry Pi, como la comunicación en serie.
- **Desarrollo de un Ecosistema IoT Robusto**: El proyecto busca desarrollar un 'ecosistema IoT robusto, seguro, en tiempo real y escalable '. La introducción de la Interfaz Periférica Serial es fundamental para lograr esta robustez, ya que permite la comunicación eficiente con múltiples dispositivos periféricos, como sensores y convertidores, utilizando menos pines que las interfaces paralelas.

En resumen, la **Interfaz Periférica Serial (SPI)** es un concepto técnico clave introducido en la **etapa 7** que permitirá a los estudiantes interactuar con **sensores y actuadores más avanzados**. Su estudio y aplicación son fundamentales para la comunicación efectiva entre los componentes en el 'Sistema de Monitoreo Atmosférico ', facilitando la creación de un dashboard que muestre 'datos en tiempo real de los sensores y actuadores' y expandiendo las capacidades del proyecto para incluir una diversidad de componentes del mundo real.

2.7.4. Panel con Gráficos Visuales en Tiempo Real

Se indica que el **Panel con Gráficos Visuales en Tiempo Real** es una característica central y el producto final del proyecto, desarrollado en el contexto más amplio de la etapa 7: **Proyecto Final: Sistema de Monitoreo Atmosférico**. A continuación, se detalla este aspecto:

- **Producto final del Curso**: El proyecto concluye con un 'dashboard ' que muestra gráficos visuales con los datos de los sensores y actuadores en tiempo real. Se describe como una interfaz 'bastante agradable a la vista '.

- **Objetivo de la etapa 7**: La etapa 7 se dedica a añadir más sensores y actuadores para construir un caso de uso significativo en el mundo real. El Sistema de Monitoreo Atmosférico es este proyecto final, y el panel de control con gráficos en tiempo real es la interfaz que presentará los datos recogidos por estos dispositivos.
- **Contexto de Desarrollo**: Este panel se construye después de haber desarrollado un ecosistema IoT robusto, seguro, en tiempo real y escalable. La creación de este 'dashboard ' representa la culminación de los conocimientos adquiridos en el proyecto, abarcando desde la simplicidad a nivel de dispositivo hasta la complejidad de la infraestructura en la nube.
- **Capacidad de Ampliación**: Al finalizar el proyecto, los participantes estarán capacitados para expandir este proyecto añadiendo más dispositivos y funcionalidades según sus propias necesidades.

En resumen, el Panel con Gráficos Visuales en Tiempo Real en la etapa 7 no es solo una parte, sino la representación visual y operativa del Sistema de Monitoreo Atmosférico, consolidando todas las habilidades y tecnologías aprendidas para construir una solución IoT completa y funcional.

Capítulo 3

PRIMER PROYECTO IoT: Detector de Movimiento

En este apartado se describe el proyecto IoT 'Detector de Movimiento', previamente reseñado en la etapa 2, como un ejemplo tangible de la implementación de sistemas de Internet de las Cosas (IoT). Este proyecto constituye una introducción práctica al desarrollo en este campo, donde los participantes tubieron la oportunidad de aplicar sus conocimientos creando un sistema funcional con sensores que reportan datos a una aplicación web. A continuación, se detallan los aspectos claves del proyecto:

- **Propósito y Funcionalidad**:
  - El objetivo es construir un dispositivo IoT detector de movimiento.
  - Detecta movimiento, activa una alarma y envía alertas al usuario, estableciendo una comunicación unidireccional del servidor al usuario.
  - También incluye una funcionalidad para que los usuarios desactiven la alarma, logrando así una comunicación bidireccional.
- **Componentes de Hardware**:
  - Sensor PIR (Passive Infrared): Este es el módulo detector de movimiento.
    - Utiliza un elemento sensor llamado RE200BL, un sensor piroeléctrico que genera energía al exponerse al calor.
    - Detecta el movimiento de cuerpos humanos o animales al captar la energía de calor (radiación infrarroja) que emiten.
    - Tiene tres pines: tierra (ground), VCC (para 5 voltios de alimentación) y un pin de salida que da un nivel lógico alto si se detecta algun objeto que emita calor.
    - Incluye dos potenciómetros: uno para ajustar la sensibilidad y otro para ajustar el tiempo que la señal de entrada permanece en alto tras la detección (de 0.3 segundos a 5 minutos).
    - También tiene pines para seleccionar modos de disparo (trigger modes): 'non repeatable trigger' (la salida vuelve a bajo automáticamente tras el tiempo de retardo) y 'repeatable trigger' (la salida se mantiene en alto mientras el objeto detectado esté presente).
  - Buzzer (Zumbador): Se activa como una alarma.
    - Consiste en una carcasa exterior con tres pines (VCC, tierra y señal) y un elemento piezoeléctrico.
    - Produce sonido al aplicar corriente, lo que causa que el disco metálico vibre.
    - Se controlará generando una onda cuadrada, alternando el pin de señal entre alto y bajo con pequeñas pausas.
- **Funcionamiento Técnico y Software**:
  - Se conectan el sensor PIR y el buzzer a una Raspberry Pi.
  - Se escribe código Python básico para detectar señales altas/bajas del movimiento y controlar el buzzer.

- Se desarrolla un servidor web HTTP básico en Python Flask que correrá en la Raspberry Pi dentro de la red Wi-Fi local.
- Para la comunicación, se utiliza la técnica AJAX.
- El usuario accede a una página web a través de la dirección IP local de la Raspberry Pi.
- Una vez cargada la página, el navegador del usuario enviará solicitudes 'keep-alive' periódicamente (cada cinco segundos) al servidor web de la Raspberry Pi. La respuesta a estas solicitudes incluye el estado y los datos del sensor, proporcionando actualizaciones en vivo al usuario.
- Un botón en la página web permite a los usuarios controlar actuadores, como desactivar el buzzer.
- Inicialmente, el servidor funciona localmente, lo que significa que los usuarios deben estar conectados a la misma red Wi-Fi para acceder a él.

- **Preparación y Configuración del Entorno**:
  - Se requiere una tarjeta SD de al menos 8GB para instalar el sistema operativo (OS) en la Raspberry Pi.
  - Se puede usar un monitor HDMI o acceso remoto (Remote Desktop).
  - Para Raspberry Pi 3, el SSH está deshabilitado por defecto y debe activarse mediante el comando `sudo raspi-config` en la terminal.

Este Primer Proyecto IoT establece las bases del sistema **IoT con Python y Raspberry PI, (*Monitor Remoto*)**, sentando las primeras experiencias con hardware, programación de sensores y creación de servidores web básicos, para luego evolucionar hacia conceptos más avanzados de comunicación y seguridad IoT en etapas posteriores.

3.1. Descripción del Proyecto

Se describe el Primer Proyecto IoT: Detector de Movimiento como un proyecto práctico introductorio del proyecto 'Internet de las Cosas con Python y Raspberry Pi', ubicado en la etapa 2.

En el contexto más amplio de este primer proyecto, la descripción general se centra en la construcción de un dispositivo IoT funcional que aborda un caso de uso real:

- **Propósito Principal**
  - El objetivo es construir un dispositivo IoT que detecte movimiento.
- **Funcionalidad Central**
  - Detección de Movimiento: El dispositivo detectará movimiento utilizando un sensor PIR.
  - Activación de Alarma: Tras la detección de movimiento, se activará una alarma, utilizando un zumbador (buzzer).
  - Envío de Alertas: El sistema enviará alertas al usuario, estableciendo una comunicación unidireccional del servidor al usuario.
  - Desactivación Remota: Para lograr una comunicación bidireccional, se añadirá una funcionalidad que permitirá a los usuarios desactivar la alarma desde una interfaz web.
- **Visión General Técnica**
  - El proyecto implica la conexión de un sensor PIR y un buzzer a una Raspberry Pi.
  - Se utilizará código Python básico para la detección de señales y el control del buzzer.
  - Se desarrollará un servidor web HTTP básico con Flask en Python que se ejecutará en la Raspberry Pi, accesible a través de la red Wi-Fi local.
  - Para la comunicación entre el cliente (navegador web) y el servidor, se empleará la técnica AJAX, enviando solicitudes 'keep-alive' periódicamente (cada cinco segundos) para recibir actualizaciones en vivo del estado y datos del sensor.

- La interfaz web incluirá un botón para que los usuarios puedan controlar los actuadores, como desactivar el buzzer.

- **Experiencia del Usuario**
  - La página web mostrará el estado de la detección de movimiento y el estado de la conexión, indicando si la conexión está activa.

- **Naturaleza Introductoria**
  - Este proyecto es la primera oportunidad para que los participantes 'se ensucien las manos ' con el desarrollo IoT, aprendiendo a usar sensores y actuadores en un entorno y a mostrar datos a través de una aplicación web.

En resumen, la descripción del proyecto destaca la creación de un sistema de seguridad básico pero funcional, sentando las bases para una comprensión práctica de los componentes de hardware, la programación y la comunicación web en el ámbito del IoT.

3.1.1. Dispositivo IoT de Defensa Antirrobo

El Dispositivo IoT de Defensa Antirrobo es el proyecto inicial y fundamental desarrollado en el proyecto, diseñado para detectar movimiento y activar una alarma, integrándose en un contexto más amplio de una plataforma IoT segura y controlable por múltiples usuarios.

A continuación, se dan los detalles de este dispositivo en el contexto de la descripción del proyecto:

- Propósito Principal.
  - El objetivo es crear un 'dispositivo IoT que detectará movimiento'.
  - Basado en la detección de movimiento, el dispositivo 'activará una alarma '.
  - También 'enviará alertas al usuario ', estableciendo una comunicación unidireccional del servidor al usuario.
  - Además, incorpora una funcionalidad para 'desactivar la alarma ' por parte del usuario, lo que permite la comunicación bidireccional del usuario al servidor.

- Componentes Principales.
  - Sensor PIR (Infrarrojo Pasivo): Es el módulo detector de movimiento.
    - Utiliza un elemento sensor llamado RE200BL, un sensor piezoeléctrico que genera energía cuando se expone al calor.
    - Detecta movimiento porque los humanos o animales emiten energía térmica en forma de radiaciones infrarrojas.
    - Es 'pasivo ' porque no usa energía para detectar, solo detecta la energía emitida por otros objetos.
    - Incluye una tapa de plástico para expandir el área de cobertura de detección.
    - Tiene tres pines: tierra, VCC (5 voltios) para alimentación y un pin de salida que emite un nivel lógico alto si se detecta un objeto y viceversa.
    - Dispone de dos potenciómetros: uno para ajustar la sensibilidad y otro para ajustar el tiempo que la señal de entrada permanece alta (desde 0.3 segundos hasta 5 minutos).
    - Ofrece dos modos de disparo: 'no repetible ' (la salida cambia de alta a baja una vez finalizado el tiempo de retardo) y 'repetible ' (la salida permanece alta mientras el objeto detectado esté presente).
  - Zumbador (Buzzer): Actúa como el sistema de alarma.
    - Consiste en una carcasa exterior con tres pines: VCC (5 voltios), tierra y señal.
    - En su interior, un elemento piezoeléctrico rodeado por un disco de vibración metálico, que al recibir corriente, se contrae y expande, generando sonido.
    - Se controla generando una onda cuadrada; se activa la señal (HIGH), se espera unos milisegundos, se desactiva (LOW) y se repite el proceso.

- Implementación y Funcionamiento.

- Los sensores y actuadores (PIR y buzzer) se conectan a una Raspberry Pi.
- Se escribe código Python para detectar las señales del sensor PIR y controlar el buzzer.
- Se implementa un servidor web HTTP básico con Flask en la Raspberry Pi que corre en la red Wi-Fi local.
- El usuario accede a una página web desde su navegador utilizando la dirección IP de la Raspberry Pi (ej. '192.168.1.250 ').
- Una vez cargada la página, el navegador del usuario envía solicitudes 'keepalive ' (mantener vivo) al servidor cada cinco segundos; estas solicitudes actúan como un 'latido ' para asegurar la conexión.
- En cada respuesta keepalive, el servidor envía el estado del sensor y los datos al usuario, lo que permite actualizaciones en tiempo real.
- Se añade un botón en la página web que permite a los usuarios 'controlar los actuadores ', específicamente para 'desactivar el zumbador '.
- Inicialmente, el servidor funciona localmente, por lo que los usuarios solo pueden acceder desde la misma red Wi-Fi.

- Contexto en la Descripción General del Proyecto.
  - Este proyecto es la primera experiencia práctica en la etapa 2, donde se usa un sensor para mostrar datos en una aplicación web.
  - Se utiliza la técnica AJAX para la comunicación entre el cliente y el servidor en esta etapa.
  - La página web inicial muestra un 'panel negro que muestra el estado de la detección de movimiento ' y un 'estado de conexión' en la parte superior.
  - El proyecto general busca construir una 'plataforma en la nube escalable donde múltiples usuarios pueden iniciar sesión de forma segura y controlar y monitorear sus dispositivos autorizados en tiempo real'. El dispositivo detector de movimiento es el primer ejemplo de un 'dispositivo autorizado ' dentro de este ecosistema.
  - Posteriormente, en secciones avanzadas, el proyecto se reconstruirá para usar protocolos de comunicación más ligeros como PubNub, y se desplegará en la nube de AWS. Finalmente, se añadirán más sensores y actuadores para casos de uso más complejos, como un sistema de monitoreo atmosférico.

3.1.2. Detección de Movimiento

La detección de movimiento es la funcionalidad central del primer proyecto IoT desarrollado en el proyecto, el Dispositivo IoT de Defensa Antirrobo. Este proyecto inicial sienta las bases para una plataforma IoT más compleja y escalable. A continuación, se dan los detalles de la detección de movimiento en el contexto de la descripción del proyecto:

- **El Sensor PIR como Base de la Detección de Movimiento**
  - Identificación del Sensor: El componente clave para la detección de movimiento es el módulo detector de movimiento PIR (Infrarrojo Pasivo).
- **Principios de Funcionamiento**:
  - Utiliza un elemento sensor llamado RE200BL, que es un sensor piezoeléctrico que genera energía cuando se expone al calor.
  - La detección de movimiento ocurre porque los cuerpos humanos o animales emiten energía térmica en forma de radiaciones infrarrojas. Cuando una persona o animal entra en el rango del sensor, este detecta ese movimiento.
  - El término 'pasivo ' significa que el sensor no utiliza energía propia para detectar, sino que funciona detectando la energía emitida por otros objetos.
  - Incorpora una tapa de plástico especialmente diseñada que se usa para expandir el área de cobertura de detección.

- **Conectividad y Salida**
  - El módulo tiene tres pines: tierra, VCC (5 voltios) para alimentación y un pin de salida que emite un nivel lógico alto si se detecta un objeto y viceversa.
- **Configuración y Modos de Disparo**
  - Dispone de dos potenciómetros: uno para ajustar la sensibilidad del sensor y otro para ajustar el tiempo que la señal de entrada permanece alta después de detectar un objeto, ajustable desde 0.3 segundos hasta 5 minutos.
  - Ofrece dos modos de disparo mediante un jumper.
    - No repetible: La salida cambia de alta a baja una vez finalizado el tiempo de retardo.
    - Repetible: La salida permanece alta todo el tiempo mientras el objeto detectado esté presente dentro del rango del sensor.
- Detección de Movimiento en el Contexto del Proyecto Inicial 'Antirrobo '.
  - Propósito del Dispositivo: El objetivo principal es construir un 'dispositivo que detectará movimiento '.
- Acciones Post-Detección.
  - Basado en la detección de movimiento, el dispositivo 'activará una alarma ' (a través de un zumbador).
  - También 'enviará alertas al usuario '
- Implementación Técnica Inicial.
  - El sensor PIR se conecta a una Raspberry Pi, donde se escribe código Python para 'detectar las señales altas y bajas al detectar movimiento ' y controlar el zumbador.
  - Se implementa un servidor web HTTP básico con Flask en la Raspberry Pi que funciona en la red Wi-Fi local.
  - La página web inicial, a la que el usuario accede desde su navegador (por ejemplo, usando la IP de la Raspberry Pi), muestra un 'panel negro que muestra el estado de la detección de movimiento '.
  - El servidor envía el 'estado del sensor y los datos al usuario ' en las respuestas a las solicitudes 'keepalive ' (mantener vivo) que el navegador envía periódicamente (cada cinco segundos), proporcionando actualizaciones en tiempo real.
- Comunicación Bidireccional: El proyecto incluye la funcionalidad para que el usuario pueda 'desactivar la alarma ' desde la página web, completando la comunicación bidireccional entre el usuario y el servidor.
- Detección de Movimiento en el Contexto Más Amplio de la Descripción del Proyecto.
  - Fundamento del Curso: La creación de este dispositivo detector de movimiento es la 'primera experiencia práctica ' del proyecto, permitiendo a los estudiantes interactuar con un sensor y mostrar sus datos en una aplicación web.
  - Evolución hacia una Plataforma Escalable: Aunque el proyecto inicial se ejecuta localmente y utiliza AJAX para la comunicación, la detección de movimiento es un caso de uso fundamental para la meta general del proyecto: construir una 'plataforma en la nube escalable donde múltiples usuarios pueden iniciar sesión de forma segura y controlar y monitorear sus dispositivos autorizados en tiempo real '.
  - Futuras Mejoras: Se indica que, después de este proyecto inicial, se estudiarán y adoptarán protocolos de comunicación más ligeros y en tiempo real como PubNub (en lugar de HTTP/AJAX) y se implementará en la nube de AWS. Esto demuestra cómo la funcionalidad básica de detección de movimiento se integra en una arquitectura IoT más avanzada y segura, capaz de manejar múltiples dispositivos y usuarios.

- Ampliación de Casos de Uso: El proyecto culminará en la etapa 7 con un proyecto más complejo llamado 'sistema de monitoreo atmosférico ', lo que subraya que la detección de movimiento es solo un ejemplo de las muchas funcionalidades basadas en sensores que se pueden integrar en la plataforma IoT desarrollada.

- En resumen, la detección de movimiento, facilitada por el sensor PIR, es el pilar del primer proyecto práctico del sistema. Este proyecto, aunque simple en sus inicios (servidor local, AJAX), es crucial para entender los conceptos básicos de IoT y sienta las bases para el desarrollo de una plataforma en la nube segura, escalable y en tiempo real que puede monitorear y controlar diversos dispositivos.

3.1.3. Activación de la Alarma

La activación de alarma es una funcionalidad central del primer proyecto práctico del proyecto, el Dispositivo IoT de Defensa Antirrobo, y se integra como un ejemplo clave de control de actuadores dentro del marco más amplio de una plataforma IoT escalable y segura. A continuación, se dan los detalles de la activación de alarma en el contexto de la descripción del proyecto:

- **Mecanismo de Activación y Propósito Principal**
  - **Detección de Movimiento como Disparador**
    - El dispositivo IoT está diseñado para detectar movimiento, y en base a ello, activará una alarma. Esta es la acción directa resultante de la detección de un intruso.
  - **Envio de Alertas al Usuario**
    - Además de la alarma sonora, el sistema también enviará alertas al usuario, lo que cumple con una comunicación unidireccional del servidor al usuario.
- **El Actuador de Alarma: El Zumbador (Buzzer)**
  - **Componente Físico**
    - La alarma se activa mediante un zumbador (buzzer).
      - El buzzer consta de una carcasa exterior con tres pines: VCC (5 voltios), tierra y señal.
      - En su interior, tiene un elemento piezoeléctrico rodeado por un disco de vibración metálico que genera sonido al recibir corriente.
  - **Control de Sonido**
    - La frecuencia del zumbador puede cambiarse para modificar la velocidad de vibración del disco, lo que a su vez altera el tono del sonido generado.
  - **Control Mediante Onda Cuadrada**
    - Para controlar el zumbador, se genera una onda cuadrada. En términos simples, esto implica alternar el pin de señal a un nivel alto (HIGH), esperar unos milisegundos, luego a un nivel bajo (LOW), esperar otros milisegundos y repetir el proceso.
- **Implementación Técnica en el Proyecto Inicial**
  - **Conexión a Raspberry Pi**
    - El zumbador se conecta a una Raspberry Pi junto con el sensor PIR.
  - **Código Python**
    - Se escribe código Python para detectar las señales altas y bajas al detectar movimiento y controlar el zumbador.
  - **Servidor Web y Dashboard**
    - El dispositivo inicial utiliza un servidor web HTTP básico con Flask en la Raspberry Pi. Aunque la página web muestra el estado de la detección de movimiento, la activación de la alarma ocurre en el dispositivo.
  - **Control del Usuario para Desactivación**
    - Para permitir una comunicación bidireccional del usuario al servidor, se añade una funcionalidad en la que los usuarios pueden desactivar la alarma.

        - ◇ Esto se implementa con un botón de interruptor en la página web que permite a los usuarios controlar los actuadores, específicamente para desactivar el zumbador.
        - ◇ Este botón es parte de la etapa de caja negra del dashboard web que muestra el estado de movimiento.

- **La Activación de Alarma en el Contexto Más Amplio del Proyecto**
    - **Primera Experiencia Práctica**
        - La implementación de la activación y desactivación de la alarma es parte de la primera experiencia práctica del proyecto en la etapa 2, donde se utilizan sensores y se muestran datos en una aplicación web, demostrando la interacción cliente-servidor a través de la técnica AJAX.
    - **Evolución hacia una Plataforma Segura y Escalable**
        - Aunque inicialmente el servidor se ejecuta localmente, el proyecto de defensa detector de movimiento sienta las bases para una plataforma en la nube escalable donde múltiples usuarios pueden iniciar sesión de forma segura y controlar y monitorear sus dispositivos autorizados en tiempo real.
    - **Seguridad y Control de Acceso**
        - En secciones posteriores del proyecto, se implementarán características de seguridad como la protección de dominios con certificados SSL/TLS, un login de usuario seguro, y un sistema de gestión de acceso donde los usuarios administradores pueden otorgar permisos de lectura y escritura en tiempo real a usuarios no administradores y dispositivos. Esto significa que la capacidad de activar o desactivar la alarma, o cualquier otro actuador, se integrará en un sistema de permisos detallado y seguro, gestionado a través de un dashboard de administrador.
    - **Protocolos de Comunicación Avanzados**
        - La funcionalidad de la alarma, que inicialmente se comunica a través de solicitudes HTTP keepalive y AJAX, eventualmente se reconstruirá para usar protocolos de comunicación en tiempo real y más ligeros como PubNub. Esto mejorará la eficiencia y la capacidad de respuesta para la activación y control de la alarma en un entorno IoT distribuido.

3.1.4. Envío de Alertas (Unidireccional)

El envío de alertas unidireccionales es una funcionalidad fundamental e inicial en el desarrollo del proyecto IoT, particularmente en el Dispositivo IoT de Defensa Antirrobo de la etapa 2. Este tipo de comunicación es el primer paso para permitir al usuario monitorear el estado de sus dispositivos.

Aquí se dan los detalles de el envío de alertas unidireccionales en el contexto más amplio de la descripción del proyecto:

- **Definición y Propósito Inicial**
    - **Comunicación Unidireccional**
        - El proyecto inicial detector de movimiento está diseñado para enviar alertas al usuario, lo que cumple con la comunicación unidireccional del servidor al usuario.
        -
    - **Activación por Evento**
        - Estas alertas se envían en base a la detección de movimiento realizada por el sensor PIR. Así, cuando se detecta un intruso, el dispositivo no solo activa una alarma local, sino que también informa al usuario.
- **Implementación Técnica Inicial**
    - **Servidor Web Local y Raspberry Pi**
        - Las alertas son gestionadas por un servidor web HTTP básico con Flask en la Raspberry Pi.
    - **Solicitudes Keepalive y AJAX**

- Una vez que el usuario carga la página web en su navegador, se envían solicitudes keepalive desde el navegador del usuario al servidor web de la Raspberry Pi periódicamente cada cinco segundos.

- **Envío de Estado y Datos**
  - La respuesta a cada solicitud keepalive se considera un latido para asegurar la conexión. En estas respuestas, el servidor también envía el estado del sensor y los datos al usuario, lo que permite que el usuario reciba actualizaciones en tiempo real sobre la detección de movimiento.
- **Interfaz de Usuario**
  - La página web inicial, que es un panel negro, muestra el estado de la detección de movimiento. Aunque no se especifica si las alertas son una notificación visual explícita o la simple actualización del estado del sensor, la comunicación de estado y datos constituye la base de la alerta.

- **Rol en el Contexto Más Amplio del Proyecto**
  - **Paso Fundamental**
    - El envío unidireccional de alertas es parte de la primera experiencia práctica del proyecto, donde los estudiantes aprenden a usar sensores y mostrar datos en una aplicación web utilizando la técnica AJAX para la comunicación cliente-servidor.
  - **Transición a la Comunicación Bidireccional**
    - Si bien el envío de alertas comienza siendo unidireccional (servidor al usuario), el mismo proyecto inicial incorpora la funcionalidad para que el usuario pueda desactivar la alarma desde la página web, completando así la comunicación bidireccional del usuario al servidor. Esto demuestra que la unidireccionalidad es una etapa temprana en el camino hacia sistemas más interactivos.
  - **Evolución hacia Protocolos en Tiempo Real**
    - Las limitaciones de la comunicación basada en HTTP/AJAX para alertas y actualizaciones se abordan en secciones posteriores. El proyecto explora y adopta protocolos de comunicación en tiempo real y ligeros como WebSockets y MQTT, y específicamente PubNub, para mejorar la eficiencia y la inmediatez de la comunicación, reemplazando las solicitudes keepalive.
  - **Integración en una Plataforma Escalable y Segura**
    - La capacidad de enviar alertas se integra en la visión general del proyecto de construir una plataforma en la nube escalable donde múltiples usuarios pueden iniciar sesión de forma segura y controlar y monitorear sus dispositivos autorizados en tiempo real.
    - Aunque las alertas básicas pueden ser unidireccionales, la plataforma general se vuelve más robusta con la implementación de seguridad HTTP IOT, login de usuario seguro y la protección de un dominio personalizado con certificados SSL/TLS.
    - Además, se establecen reglas de acceso para usuarios administradores y no administradores, permitiendo a los administradores otorgar permisos de lectura y escritura en tiempo real a usuarios y dispositivos. Esto asegura que las alertas y el acceso a los datos del sensor se gestionen de forma segura y con control de permisos, incluso si la alerta en sí es un flujo de datos de un solo sentido desde el dispositivo.

En conclusión, el envío de alertas unidireccionales es un componente esencial del proyecto inicial detector de movimiento, proporcionando al usuario la capacidad básica de monitorear eventos (como la detección de movimiento) desde el servidor a través de una aplicación web. Este mecanismo, inicialmente implementado con HTTP y AJAX, sienta las bases para la comunicación en tiempo real y la gestión de datos en una plataforma IoT más avanzada, segura y bidireccional que se desarrolla a lo largo del proyecto.

3.1.5. Funcionalidad para Desactivar Alarma (Bidireccional)

La funcionalidad para desactivar la alarma es un componente crucial que establece la comunicación bidireccional dentro del proyecto del Dispositivo IoT de Defensa Antirrobo, evolucionando desde una simple interacción local hasta una operación segura y gestionada en la nube.

A continuación, se dan los detalles de esta funcionalidad en el contexto más amplio del proyecto:

- **Establecimiento de Comunicación Bidireccional**
  - **Complemento a las Alertas Unidireccionales**
    - Inicialmente, el proyecto se centra en la detección de movimiento para activar una alarma y enviar alertas al usuario, lo que cumple con la comunicación unidireccional del servidor al usuario. Sin embargo, para completar la comunicación bidireccional del usuario al servidor, se añade la funcionalidad para que los usuarios puedan desactivar la alarma.
  - **Control de Actuadores**
    - Esta capacidad de desactivar la alarma es un ejemplo clave de cómo los usuarios pueden controlar los actuadores del sistema, en este caso, el zumbador (buzzer).
- **Implementación Técnica Inicial**
  - **Interfaz de Usuario**
    - La desactivación se logra mediante un botón de interruptor añadido a la página web del proyecto. Este botón se encuentra en la etapa de caja negra del dashboard web, junto al estado de detección de movimiento.
  - **Conexión y Lógica del Zumbador**
    - La alarma es generada por un zumbador (buzzer) que se controla generando una onda cuadrada. La funcionalidad de desactivación interrumpe o modifica esta onda para silenciar el zumbador.
  - **Servidor Web y Raspberry Pi**
    - El control se realiza a través de un servidor web HTTP básico con Flask en la Raspberry Pi. La comunicación inicial utiliza solicitudes keepalive y la técnica AJAX para enviar y recibir datos. Cuando el usuario interactúa con el botón de desactivación, se envía una solicitud al servidor local.
  - **Acceso Local**
    - En esta etapa inicial, el servidor se ejecuta localmente, lo que significa que los usuarios solo pueden acceder a él desde dentro de la red (conectados al mismo router Wi-Fi).
- **Evolución hacia una Plataforma Segura y Escalable**
  - **Gestión de Permisos (etapa 5 y 6)**
    - La capacidad de desactivar la alarma se integra en un sistema de gestión de acceso más sofisticado.
    - Se crean reglas para usuarios administradores y no administradores.
    - Los usuarios administradores tienen un panel de control donde pueden ver una lista de usuarios en línea y, a través de botones de interruptor, otorgar permisos de lectura y escritura en tiempo real a usuarios no administradores y dispositivos. La desactivación de la alarma sería una acción que requiere permisos de escritura.
    - El servidor valida que la solicitud de cambio de permisos proviene de un usuario administrador antes de procesarla.
    - Los permisos (lectura/escritura) se almacenan en la base de datos y se utilizan para otorgar acceso de lectura y escritura a usuarios específicos en el servidor PubNub.
  - **Seguridad Mejorada (etapa 5)**
    - Para asegurar esta comunicación bidireccional y el control de los dispositivos.
    - El dominio personalizado se protege con certificados SSL/TLS de Let's Encrypt, garantizando una comunicación cifrada de extremo a extremo entre el cliente y el servidor para todas las interacciones. Esto incluye las solicitudes para desactivar la alarma.
    - Se implementa una funcionalidad de inicio de sesión de usuario seguro y el almacenamiento de detalles del usuario en una base de datos integrada.
  - **Protocolos de Comunicación Avanzados (etapa 3 y 4)**
    - La comunicación inicial basada en HTTP/AJAX se mejora con protocolos de comunicación en tiempo real y ligeros como WebSockets y MQTT.

- PubNub se adopta como el protocolo principal, reemplazando el long pooling de AJAX. PubNub es una solución basada en el modelo publicar/suscribir que permite a los clientes enviar mensajes al servidor y recibir respuestas basadas en eventos sin tener que realizar un sondeo. Esto es fundamental para un control eficiente y en tiempo real de los actuadores como la alarma.
- El uso de PubNub Access Manager permite gestionar los permisos de manera granular para que solo los usuarios autorizados puedan enviar comandos (como desactivar la alarma) a los dispositivos.

En resumen, la funcionalidad para desactivar la alarma no solo representa la culminación de la comunicación bidireccional en el proyecto inicial del detector de movimiento, sino que también sirve como un ejemplo práctico de cómo el control de actuadores se integra en una plataforma IoT segura, escalable y en tiempo real. Esta evolución implica el uso de certificados SSL/TLS para la seguridad, un sistema de gestión de acceso basado en roles para permisos de lectura/escritura, y la adopción de protocolos de comunicación avanzados como PubNub para un control eficiente y en tiempo real.

3.2. Componentes de Hardware

Se detallan los Componentes de Hardware esenciales para el Primer Proyecto IoT: Detector de Movimiento, que se aborda en la etapa 2 del proyecto Internet de las Cosas con Python y Raspberry Pi. Este proyecto tiene como objetivo construir un dispositivo que detecte movimiento, active una alarma y envíe alertas al usuario, permitiendo también la desactivación de la alarma.

Los componentes de hardware claves para este proyecto son:

- **Sensor PIR (Passive Infrared Sensor)**
  - Función principal: Es el módulo detector de movimiento.
  - Principio de funcionamiento: Utiliza un elemento sensor llamado RE200BL, que es un sensor piroeléctrico que genera energía cuando se expone al calor. Detecta el movimiento de cuerpos humanos o animales al captar la energía de calor (radiación infrarroja) que emiten. Se le denomina pasivo porque no emite energía para detectar, solo detecta la energía emitida por otros objetos.
  - Cobertura: Incluye una tapa de plástico especialmente diseñada para expandir el área de cobertura de detección.
  - Pines: Cuenta con tres pines
    - Ground (Tierra).
    - VCC: Para alimentación de 5 voltios.
    - Output (Salida): Proporciona un nivel lógico alto si se detecta un objeto y viceversa.
  - Potenciómetros: Dispone de dos para ajustar su comportamiento
    - Uno para ajustar la sensibilidad del sensor.
    - Otro para ajustar el tiempo que la señal de entrada permanece en alto después de la detección, con un rango que va desde 0.3 segundos hasta 5 minutos.
  - Modos de disparo (Trigger Modes): Tiene tres pines con un puente (jumper) entre dos de ellos para seleccionar los modos de disparo
    - Non repeatable trigger: La salida vuelve automáticamente de alto a bajo una vez que el tiempo de retardo ha terminado, incluso si el objeto sigue presente.
    - Repeatable trigger: La salida se mantiene en alto mientras el objeto detectado permanezca dentro del rango del sensor.
- **Zumbador (Buzzer)**
  - Función principal: Se utiliza para activar la alarma.
  - Componentes: Consiste en una carcasa exterior con tres pines (VCC para 5 voltios, tierra y señal) y un elemento piezoeléctrico rodeado por un disco de vibración metálico.

- Generación de sonido: Cuando se le aplica corriente, el disco metálico se contrae y expande, causando vibraciones que producen el sonido. Cambiar la frecuencia de la corriente aplicada puede alterar el tono del sonido.
- Control: Se controlará generando una onda cuadrada, lo que implica alternar el pin de señal entre un estado alto y bajo con pequeñas pausas entre cada cambio.

- **Raspberry Pi**
  - Rol: Sirve como el cerebro del sistema. A ella se conectarán el sensor PIR y el buzzer.
  - Programación: Se utilizará para ejecutar el código Python que detecta las señales del sensor y controla el buzzer.
  - Servidor Web: Ejecutará un servidor web HTTP básico en Python Flask, el cual funcionará en la red Wi-Fi local para que los usuarios puedan interactuar con el dispositivo.
  - Configuración inicial:
    - Requiere una tarjeta SD de al menos 8GB para instalar el sistema operativo.
    - Se puede usar un monitor HDMI o, para acceso remoto, es necesario habilitar SSH (deshabilitado por defecto en Raspberry Pi 3) usando el comando `sudo raspi-config` en la terminal.

Estos componentes trabajan en conjunto para permitir la detección de movimiento, la emisión de una alarma y la interacción bidireccional con el usuario a través de una interfaz web básica.

3.2.1. Sensor PIR (Infrarrojo Pasivo)

El Sensor PIR (Infrarrojo Pasivo) es un componente de hardware fundamental y central en el Dispositivo IoT de Defensa Antirrobo del proyecto, sirviendo como el principal sensor de detección de movimiento.

Aquí se dan los detalles de el Sensor PIR en el contexto más amplio de los componentes de hardware:

- **Definición y Principio de Funcionamiento**
  - Módulo Detector de Movimiento: El Sensor PIR se describe como un módulo detector de movimiento.
  - Sensor Piroeléctrico: Contiene un elemento sensor llamado RE200 BL, que es un sensor piroeléctrico.
  - Detección por Calor: Este tipo de sensor genera energía cuando se expone al calor. Detecta el movimiento porque los cuerpos humanos o animales emiten energía calorífica en forma de radiaciones infrarrojas.
  - Naturaleza Pasiva: El término pasivo en su nombre significa que el sensor no utiliza ninguna energía para el propósito de detección, sino que funciona simplemente detectando la energía emitida por otros objetos.
  - Cobertura de Detección: El módulo incluye una tapa de plástico especialmente diseñada que se utiliza para expandir la cobertura del área de detección.

- **Pines y Conexión**
  - Tres Pines Principales: El módulo PIR tiene tres pines esenciales para su funcionamiento:
    - Tierra (Ground): Para la conexión a tierra.
    - VCC (5V): Para el suministro de energía (cinco voltios).
    - Salida (Output): Este pin proporciona un nivel lógico alto si se detecta un objeto y viceversa (un nivel lógico bajo si no hay detección).
  - Conexión a Raspberry Pi:
    - El Sensor PIR se conecta directamente a la Raspberry Pi.

- **Configuraciones Ajustables y Modos de Disparo**
- El módulo PIR ofrece opciones de configuración para adaptar su comportamiento.

- Potenciómetros:
  - Uno para ajustar la sensibilidad del sensor.
  - Otro para ajustar el tiempo en que la señal de entrada permanece en alto (el tiempo que la salida permanece activada después de la detección). Este tiempo se puede ajustar desde 0.3 segundos hasta 5 minutos.
- Modos de Disparo (Trigger Modes):
- Dispone de tres pines adicionales con un puente (jumper) entre dos de ellos para seleccionar los modos de disparo:
  - Disparador No Repetible (Non-Repeatable Trigger): Cuando el sensor detecta y el tiempo de retardo ha terminado, la salida cambia automáticamente de alta a baja.
  - Disparador Repetible (Repeatable Trigger): La salida se mantendrá alta todo el tiempo hasta que el objeto detectado esté presente en el rango de detección del sensor.

- **Rol en el Contexto del Proyecto de Hardware**
  - Base del Proyecto Antirrobo:
    - El Sensor PIR es el corazón del proyecto detector de movimiento en la etapa 2. Su función es detectar el movimiento para activar una alarma y enviar alertas al usuario.
  - Interacción con Raspberry Pi y Software:
    - Una vez conectado a la Raspberry Pi, se escribe un código Python básico que detecta las señales altas y bajas al detectar movimiento provenientes del sensor. Estos datos son cruciales para el funcionamiento del servidor web HTTP básico en la Raspberry Pi, que a su vez envía el estado del sensor y los datos al usuario para actualizaciones en tiempo real.
  - Componente en una Plataforma IoT Más Amplia:
    - Aunque el proyecto inicial lo usa para una función específica, la comprensión y el uso de sensores como el PIR son una parte fundamental del proyecto, que busca construir una plataforma en la nube escalable donde múltiples usuarios pueden iniciar sesión de forma segura y controlar y monitorear sus dispositivos autorizados en tiempo real. La capacidad de un sensor PIR para proporcionar datos de eventos (detección de movimiento) es la base para la toma de decisiones y el control en un sistema IoT.

En síntesis, el Sensor PIR es el ojo del sistema IoT, detectando el movimiento a través del calor infrarrojo. Sus características técnicas, como los pines, las opciones de sensibilidad y tiempo de retardo, y los modos de disparo, permiten configurarlo para una detección precisa. Su integración con la Raspberry Pi y el software de Python es el primer paso para convertir las detecciones físicas en datos utilizables por el usuario y para el control de actuadores como el zumbador, sentando las bases para sistemas IoT más complejos y seguros.

ELEMENTO SENSOR: RE200B

Aquí se detallan claramente que el Elemento Sensor: RE200B es el componente clave dentro del Sensor PIR (Infrarrojo Pasivo).

En el contexto más amplio del Sensor PIR, se indican lo siguiente sobre el RE200B:

- Identificación del Elemento Sensor: El módulo detector de movimiento PIR (sensor de infrarrojos pasivo) tiene un elemento sensor llamado RE200B.
- Naturaleza del RE200B: Se describe como un sensor piroeléctrico.
- Principio de Funcionamiento: Este elemento sensor genera energía cuando se expone al calor. Esto es crucial porque significa que cuando un cuerpo humano o animal entra en el rango del sensor, detectará un movimiento debido a que los humanos o animales emiten energía térmica en forma de radiación infrarroja.
- Nomenclatura Pasivo: El término pasivo en Sensor Infrarrojo Pasivo proviene de que el sensor no utiliza ninguna energía para fines de detección; simplemente funciona detectando la energía emitida por otros objetos. Esto es directamente aplicable a cómo el RE200B capta las emisiones de calor.

- Función en el Proyecto Anti-robo: El sensor PIR, con su elemento RE200B, es la parte del dispositivo IoT que detecta el movimiento, lo que a su vez activará una alarma.

En resumen, el RE200B es el corazón del sensor PIR, siendo el elemento piroeléctrico responsable de detectar la radiación infrarroja (calor) emitida por humanos o animales, permitiendo así la detección de movimiento sin emitir su propia energía. Esta capacidad es fundamental para el funcionamiento del proyecto de detección de movimiento.

TIPO: PIEZOELéCTRICO

Aquí se establece que el tipo piezoeléctrico es una característica fundamental del Sensor PIR (Infrarrojo Pasivo), particularmente en relación con su elemento sensor clave, el RE200B.

En el contexto más amplio del Sensor PIR, se detallan lo siguiente sobre su naturaleza piezoeléctrica:

- Identificación del Elemento Sensor como Piezoeléctrico:
  - El módulo detector de movimiento PIR (sensor de infrarrojos pasivo) contiene un elemento sensor llamado RE200B, y se describe explícitamente como un sensor piroeléctrico. La termología piroeléctrico se refiere a la capacidad de ciertos materiales de generar una carga eléctrica en respuesta a un cambio de temperatura, lo cual es una forma de piezoelectricidad inducida por el calor.
- Principio de Funcionamiento Basado en el Efecto Piezoeléctrico:
  - El funcionamiento del sensor se basa directamente en esta propiedad piezoeléctrica. El elemento sensor genera energía cuando se expone al calor. Esto significa que cuando un cuerpo humano o animal entra en el rango del sensor, detectará un movimiento porque los humanos o animales emiten energía térmica en forma de radiación infrarroja.
- Fundamento de la Detección de Movimiento:
  - La capacidad del RE200B para convertir la energía térmica (infrarroja) en una señal eléctrica (energía) es lo que permite al sensor PIR detectar el movimiento. Sin esta característica piezoeléctrica, el sensor no podría percibir los cambios en la radiación infrarroja del entorno.
- Relación con el Término Pasivo:
  - El término pasivo en Sensor Infrarrojo Pasivo se explica porque el sensor no utiliza ninguna energía para fines de detección; simplemente funciona detectando la energía emitida por otros objetos. La naturaleza piezoeléctrica del RE200B es inherentemente pasiva en este sentido, ya que reacciona a la energía existente en lugar de emitirla.

En resumen, el tipo piezoeléctrico es la base del funcionamiento del Sensor PIR. A través de su elemento RE200B, este sensor aprovecha la propiedad de generar una señal eléctrica al exponerse al calor, lo que le permite detectar de manera pasiva el movimiento de cuerpos que emiten radiación infrarroja (como humanos y animales) y es fundamental para su rol en el proyecto de detección de movimiento.

DETECCIóN: CALOR (RADIACIóN INFRAROJA DE CUERPOS)

Aquí se detalla de manera concisa el mecanismo de Detección: Calor (Radiación Infrarroja de Cuerpos) como el principio fundamental del Sensor PIR (Infrarrojo Pasivo).

En el contexto más amplio del Sensor PIR, se explican lo siguiente sobre su método de detección:

- Naturaleza de la Detección:
  - El módulo detector de movimiento PIR, que es un sensor de infrarrojos pasivo, detecta el movimiento basándose en la detección de calor.
- Elemento Sensor (RE200B):
  - El corazón de esta detección es el elemento sensor llamado RE200B, que es un sensor piroeléctrico. Este elemento genera energía cuando se expone al calor.
- Fuente del Calor (Cuerpos Humanos o Animales):

- La detección se activa cuando un cuerpo humano o animal entra en el rango del sensor. Esto se debe a que los humanos o animales emiten energía térmica en forma de radiación infrarroja.

- Detección de Movimiento:
  - Al detectar esta radiación infrarroja (calor) emitida por cuerpos, el sensor puede detectar un movimiento.
- Significado de Pasivo:
  - El término pasivo en Sensor Infrarrojo Pasivo se explica porque el sensor no utiliza ninguna energía para fines de detección; simplemente funciona detectando la energía emitida por otros objetos. Esto subraya que el sensor no emite su propia energía, sino que reacciona a la radiación térmica presente en su entorno.

En resumen, el Sensor PIR detecta el movimiento al identificar los cambios en la radiación infrarroja (calor) emitida de forma natural por cuerpos humanos o animales. Su naturaleza pasiva significa que simplemente percibe esta energía existente sin emitir la suya propia, utilizando un elemento piroeléctrico para convertir el calor detectado en una señal que indica movimiento.

PASIVO: NO EMITE ENERGíA

Aquí se resalta un aspecto fundamental del Sensor PIR (Infrarrojo Pasivo): su cualidad de ser Pasivo: No Emite Energía.

En el contexto más amplio del Sensor PIR, se explican lo siguiente sobre esta característica:

- Origen del Nombre Pasivo: El nombre pasivo en Sensor Infrarrojo Pasivo se deriva directamente de su principio de funcionamiento.
- Principio de No Emisión de Energía: La razón por la que se le llama pasivo es porque el sensor no utiliza ninguna energía para fines de detección. Esto significa que el sensor no irradia su propia energía (por ejemplo, infrarroja o microondas) para iluminar su entorno y luego detectar reflexiones.
- Detección de Energía Emitida por Otros Objetos: En cambio, el sensor PIR simplemente funciona detectando la energía emitida por otros objetos. Esta energía son las radiaciones infrarrojas o el calor que los humanos o animales emiten.
- Rol del Elemento Sensor (RE200B): El elemento sensor RE200B, que es un sensor piroeléctrico, es el encargado de generar energía cuando se expone al calor. Al ser piroeléctrico, reacciona a los cambios en la radiación térmica existente en su campo de visión, sin necesidad de emitir nada por sí mismo.

En resumen, la naturaleza pasiva del Sensor PIR significa que no emite su propia energía para detectar. En su lugar, detecta el movimiento percibiendo la radiación infrarroja (calor) que de forma natural desprenden cuerpos tanto humanos como animales que entran en su rango de detección. Esta característica es clave para su funcionamiento como detector de movimiento en proyectos como el dispositivo IoT detector de movimiento.

TAPA PLASTICA: EXPANDE áREA DE DETECCIóN

Aquí se resalta una característica importante del Sensor PIR (Infrarrojo Pasivo): la Tapa Plástica: Expande Área de Detección.

En el contexto más amplio del Sensor PIR, se indican lo siguiente sobre este componente:

- Componente del Módulo: El módulo del detector de movimiento PIR también consta de una tapa de plástico especialmente diseñada.
- Función Principal: La función explícita de esta tapa de plástico es expandir la cobertura del área de detección.

Esta tapa, a menudo conocida como lente Fresnel, es crucial porque el elemento sensor piroeléctrico (como el RE200B) tiene un campo de visión muy limitado. Al expandir el área de detección, la tapa plástica permite que el sensor perciba cambios en la radiación infrarroja de un área mucho mayor, lo que es esencial para su eficacia en la detección de movimiento de cuerpos como humanos o animales.

En resumen, la tapa de plástico es un componente integral del Sensor PIR cuya finalidad es ampliar significativamente el área de cobertura en la que el sensor puede detectar la radiación infrarroja y, por ende, el movimiento, optimizando así su rendimiento en aplicaciones como sistema detector de movimiento.

PINES

Aquí se detalla de manera específica la configuración de los pines del Sensor PIR (Infrarrojo Pasivo).
En el contexto más amplio del Sensor PIR, se describen lo siguiente sobre sus pines y otros elementos de conexión:

- Pines Principales para Conexión a Raspberry Pi: El módulo del sensor PIR tiene tres pines principales para su funcionamiento básico:
  - Ground (Tierra): Para la conexión a tierra.
  - VCC (5 voltios): Para alimentar el sensor.
  - Output pin (Pin de Salida): Este pin da un nivel lógico alto si se detecta un objeto y viceversa. Esto es fundamental para que la Raspberry Pi pueda interpretar si ha habido movimiento.
  -
- Potenciómetros para Ajustes: Además de los pines de conexión, el módulo cuenta con dos potenciómetros para afinar su comportamiento:
  - Ajuste de Sensibilidad: Uno de los potenciómetros es para ajustar la sensibilidad del sensor.
  - Ajuste de Tiempo de Señal Alta: El otro es para ajustar el tiempo que la señal de entrada permanece en alto cuando se detecta el objeto. Este tiempo se puede ajustar desde 0.3 segundos hasta 5 minutos.
- Pines Adicionales para Modos de Disparo (Trigger Modes) El módulo también incorpora tres pines más con un jumper entre dos de ellos que se utilizan para seleccionar los modos de disparo:
  - Disparo no repetible (Non repeatable trigger): En este modo, cuando el sensor da una salida alta y el tiempo de retardo ha terminado, la salida cambia automáticamente de alta a baja.
  - Disparo repetible (Repeatable trigger): Este modo mantendrá la salida en alto todo el tiempo hasta que el objeto detectado esté presente en el rango del sensor.

En resumen, los pines del Sensor PIR son cruciales para su integración con la Raspberry Pi, permitiendo la alimentación (VCC, Ground) y la transmisión de la señal de detección de movimiento (Output). Los potenciómetros ofrecen una flexibilidad importante para calibrar la sensibilidad y el tiempo de respuesta del sensor, mientras que los pines adicionales con el jumper permiten configurar los modos de disparo, adaptando el comportamiento del sensor a las necesidades específicas del proyecto, como la detección de movimiento.

POTENCIOMETROS

Aquí se especifica la presencia y función de los potenciómetros como componentes ajustables clave en el Sensor PIR (Infrarrojo Pasivo).
En el contexto más amplio del Sensor PIR, se describen lo siguiente sobre sus potenciómetros:

- Número y Ubicación: El módulo del sensor PIR cuenta con dos potenciómetros.
- Ajuste de Sensibilidad: Uno de estos potenciómetros se utiliza para ajustar la sensibilidad del sensor. Esto permite al usuario calibrar qué tan fácilmente el sensor detectará cambios en la radiación infrarroja, adaptándolo al entorno específico de su aplicación (por ejemplo, para evitar falsas alarmas o asegurar una detección precisa).
- Ajuste del Tiempo de Señal Alta: El otro potenciómetro sirve para ajustar el tiempo que la señal de entrada permanece en alto cuando se detecta el objeto. Esta es una función crucial para controlar la duración de la señal de detección.

- Rango de Ajuste: Este tiempo se puede ajustar en un rango significativo, desde 0.3 segundos hasta 5 minutos. Esta flexibilidad es vital para adaptar el comportamiento del sensor a los requisitos de un proyecto, como un sistema detector de movimiento, donde la duración de la alarma o el estado de alerta puede necesitar configuraciones específicas.

En resumen, los potenciómetros del Sensor PIR son elementos de control fundamentales que permiten a los usuarios personalizar la sensibilidad de detección y la duración de la señal de salida tras la detección de movimiento. Estas capacidades de ajuste son esenciales para la integración efectiva del sensor en proyectos de IoT, como un dispositvo detector de movimiento, asegurando que su rendimiento se adapte con precisión a las necesidades y condiciones del entorno.

MODOS DE DISPARO (JUMPER)

Aquí se describen los Modos de Disparo (Jumper) como una característica configurable del Sensor PIR (Infrarrojo Pasivo).

En el contexto más amplio del Sensor PIR, se explican lo siguiente sobre esta funcionalidad:

- Configuración Física: El módulo del sensor PIR incluye tres pines más con un jumper entre dos de ellos. Este jumper se utiliza específicamente para seleccionar los modos de disparo.
- Modos de Disparo Disponibles: Se detallan dos modos principales:
  - Disparo no repetible (Non repeatable trigger): En este modo, cuando el sensor da una salida alta y el tiempo de retardo ha terminado, la salida cambia automáticamente de alta a baja. Esto significa que, después de detectar movimiento y mantener la señal alta por el tiempo configurado, el sensor se reinicia a un estado bajo, esperando una nueva detección antes de volver a activarse.
  - Disparo repetible (Repeatable trigger): Este modo mantendrá la salida en alto todo el tiempo hasta que el objeto detectado esté presente en el rango del sensor. A diferencia del modo no repetible, la señal de salida permanecerá alta mientras haya movimiento continuo dentro del rango de detección, lo cual es útil para aplicaciones donde se requiere una señal constante mientras el objeto está presente.

En resumen, los Modos de Disparo (Jumper) del Sensor PIR ofrecen una flexibilidad importante para adaptar el comportamiento de la señal de salida del sensor después de una detección. Mediante la colocación de un jumper en pines específicos, el usuario puede elegir entre un disparo no repetible (donde la señal se restablece después de un retardo fijo) o un disparo repetible (donde la señal permanece activa mientras se detecta movimiento), lo que permite optimizar su funcionamiento en diversas aplicaciones como, por ejemplo, un sistema de seguridad.

3.2.2. Zumbador

El Zumbador (Buzzer) es un componente de hardware esencial en el proyecto del Dispositivo IoT de Defensa Antirrobo, funcionando como el actuador principal para la alarma. Su integración permite que el sistema responda a la detección de movimiento y que el usuario interactúe con el dispositivo para desactivar dicha alarma.

A continuación, se dan los detalles de el Zumbador en el contexto más amplio de los componentes de hardware:

- Rol y Función en el Proyecto.
  - Generador de Alarma: El zumbador es el componente encargado de activar una alarma cuando se detecta movimiento. Es una parte crucial de la funcionalidad de detección de movimiento.
  - Actuador Controlable por el Usuario: El proyecto permite a los usuarios controlar los actuadores, siendo el zumbador un ejemplo principal de esto. Se añade un botón de interruptor en la página web para que los usuarios puedan desactivar el zumbador (que representa la alarma). Esto completa la comunicación bidireccional del usuario al servidor.
- Descripción Física y Conexiones
  - Componentes: Un zumbador consta de una caja exterior con tres pines y, en su interior, un elemento piezoeléctrico rodeado por un disco metálico vibratorio.

- Pines: Los tres pines del zumbador son:
  - VCC (5V): Para el suministro de energía.
  - Tierra (Ground): Para la conexión a tierra.
  - Señal (Signal): El pin a través del cual se controla el zumbador.
- Conexión a Raspberry Pi: El zumbador se conecta directamente con la Raspberry Pi.

- Principio de Funcionamiento y Control
  - Generación de Sonido: Cuando se aplica corriente al zumbador, el disco metálico se contrae y expande, lo que provoca su vibración y, como resultado, la producción de sonido.
  - Control del Tono (Pitch): Si se cambia la frecuencia de la corriente aplicada al zumbador, la velocidad de vibración del disco también cambia, lo que a su vez cambia el tono (pitch) del sonido resultante. Esto permite generar diferentes melodías o sonidos.
  - Control mediante Onda Cuadrada: Para controlar el zumbador, se genera una onda cuadrada. En términos sencillos, esto implica alternar el pin de señal entre un estado alto y un estado bajo durante ciertos milisegundos y repetir este proceso.
  - Programación en Python: Se escribe un código Python básico para controlar el zumbador, manejando la lógica de la onda cuadrada.

- Integración en el Sistema
  - Activación por Sensor PIR: El zumbador se activa en respuesta a la detección de movimiento por parte del Sensor PIR.
  - Servidor Web y Desactivación: El estado del zumbador (activado/desactivado) se puede ver y controlar a través de un servidor web HTTP básico con Flask en la Raspberry Pi. Un botón en la interfaz web permite a los usuarios desactivar el zumbador, interrumpiendo o modificando la generación de la onda cuadrada que produce la alarma.

En síntesis, el Zumbador es el principal actuador sonoro del sistema detector de movimiento. Su capacidad para generar una alarma visible y controlable por el usuario, mediante la aplicación de una onda cuadrada y la interacción a través de una interfaz web, lo convierte en un componente crucial para la funcionalidad bidireccional y la respuesta del sistema IoT detector de movimiento.

COMPONENTE: ELEMENTO PIEZOELéCTRICO, DISCO DE VIBRACIóN METáLICO.

Aquí se explica de manera directa los Componentes: Elemento Piezoeléctrico, Disco de Vibración Metálico en el contexto más amplio de un Zumbador (Buzzer).

Según las notas técnicas:

- Estructura Interna del Zumbador: Un zumbador, más allá de su carcasa exterior y sus pines de conexión (VCC, Ground, Signal), está compuesto internamente por un elemento piezoeléctrico que está rodeado por un disco de vibración metálico.

- Mecanismo de Generación de Sonido:
  - Cuando se aplica corriente al zumbador, el disco hace que el disco se contraiga y se expanda.
  - Esta contracción y expansión, a su vez, hace que el disco circundante vibre, lo cual produce el sonido que se escucha.

- Control del Sonido:
  - La frecuencia de la corriente aplicada al zumbador determina la velocidad de la vibración del disco, lo que a su vez cambia el tono del sonido resultante. Esto permite generar diferentes melodías o sonidos.
  - Para controlar el zumbador, se puede generar una onda cuadrada. En términos sencillos, esto implica alternar el pin de señal entre un estado alto y un estado bajo, esperando unos milisegundos entre cada cambio y repitiendo el proceso.

En resumen, se indican que el elemento piezoeléctrico y el disco de vibración metálico son los componentes internos clave de un zumbador. Estos trabajan en conjunto: el elemento piezoeléctrico, al recibir corriente, hace que el disco metálico vibre, produciendo así el sonido, cuya frecuencia puede ser modulada para cambiar el tono.

SONIDO: VIBRACIóN DE DISCO

Aquí se explica claramente cómo se genera el Sonido: Vibración del Disco en el contexto más amplio de un Zumbador (Buzzer).
Según notas técnicas:

- Componentes Clave: Un zumbador está compuesto internamente por un elemento piezoeléctrico que está rodeado por un disco de vibración metálico.
- Mecanismo de Generación de Sonido:
  - Cuando se aplica corriente eléctrica al zumbador, el elemento piezoeléctrico hace que el disco de vibración metálico se contraiga y se expanda.
  - Esta acción de contracción y expansión, a su vez, provoca que el disco circundante vibre.
  - Es precisamente esta vibración del disco la que produce el sonido que se escucha.
- Control del Tono del Sonido:
  - La frecuencia de la corriente que se aplica al zumbador es crucial, ya que esta determina la velocidad de la vibración del disco.
  - Un cambio en la velocidad de vibración resulta en un cambio en el tono del sonido resultante, lo que permite generar diferentes melodías.
  - Para controlar el zumbador y generar sonidos, se puede utilizar una onda cuadrada, que implica alternar el pin de señal entre estados alto y bajo con esperas de milisegundos.

En síntesis, el sonido de un zumbador se origina directamente de la vibración de un disco metálico, la cual es inducida por la contracción y expansión generada por un elemento piezoeléctrico al recibir corriente eléctrica. La capacidad de controlar la frecuencia de esta vibración permite variar el tono del sonido producido.

PINES

Aquí se especifica la configuración de los pines de un Zumbador (Buzzer).
El zumbador:

- Está formado por una carcasa exterior a la que se le adjuntan tres pines.
- Estos tres pines tienen funciones específicas para su conexión y control:
  - VCC (5 voltios): Para la alimentación del zumbador.
  - Ground (Tierra): Para la conexión a tierra.
  - Signal (Señal): Este pin es crucial para el control del zumbador, ya que a través de él se genera una onda cuadrada, alternando entre estados alto y bajo para producir el sonido.

En resumen, los pines del zumbador son esenciales para su integración y funcionamiento, permitiendo tanto su alimentación como el control de la generación del sonido mediante la señal.

CONTROL: ONDA CUADRADA (ALTA, ESPERAR; BAJA, ESPERAR)

Aquí se explica de manera específica el concepto de Control: Onda Cuadrada (Alta, Esperar; Baja, Esperar) en el contexto más amplio de un Zumbador (Buzzer).
Según notas técnicas:

- Método de Control: Para controlar el zumbador y generar sonido, se utilizará la generación de una onda cuadrada.
- Funcionamiento Simplificado: En términos sencillos, el proceso implica:
  - Poner el pin de señal en alto (High).
  - Esperar algunos milisegundos.
  - Ponerlo en bajo (Low).

- Esperar algunos milisegundos nuevamente.
- Repetir el proceso otra vez.

- Generación de Sonido y Tono: Esta alternancia rápida entre estados altos y bajos es lo que crea la onda cuadrada, la cual, al aplicarse al elemento piezoeléctrico y al disco vibratorio del zumbador, causa que el disco se contraiga y se expanda. Esta contracción y expansión, a su vez, hace que el disco circundante vibre, produciendo así el sonido. Además, la frecuencia de la corriente aplicada (es decir, la velocidad a la que se alternan los estados alto y bajo en la onda cuadrada) determina la velocidad de la vibración del disco, lo que permite cambiar el tono del sonido resultante y, en consecuencia, generar algunas melodías hermosas.

En resumen, el control de un zumbador mediante una onda cuadrada es el método fundamental para producir sonido. Este control se logra alternando el pin de señal entre un estado alto y uno bajo, con pausas en milisegundos entre cada cambio. La velocidad de esta alternancia (la frecuencia de la onda cuadrada) es crucial, ya que permite modular la vibración del disco interno del zumbador y, por ende, el tono del sonido generado.

3.3. Funcionamiento Técnico

En el contexto más amplio del Primer Proyecto IoT: Detector de Movimiento, el funcionamiento técnico se refiere a cómo los componentes de hardware y software interactúan para crear un sistema de seguridad anti-robo funcional. Este proyecto se presenta como una introducción práctica en la etapa 2 del proyecto Internet de las Cosas con Python y Raspberry Pi.

El funcionamiento técnico se detalla a través de los siguientes puntos:

- Conexión de Hardware:
  - El sensor PIR (Passive Infrared Sensor) y el buzzer (zumbador) se conectan físicamente a la Raspberry Pi.
  - El sensor PIR detecta el movimiento de cuerpos (humanos o animales) captando la energía de calor (radiación infrarroja) que emiten. Tiene un pin de salida que proporciona un nivel lógico alto cuando se detecta un objeto.
  - El buzzer, que se utiliza como alarma, produce sonido al aplicarle corriente, lo que causa la vibración de su disco metálico.

- Programación con Python en Raspberry Pi:
  - Se escribe código Python básico en la Raspberry Pi. Este código es responsable de:
    - Detectar las señales de nivel alto y bajo provenientes del sensor PIR cuando se detecta movimiento.
    - Controlar el buzzer generando una onda cuadrada, lo que implica alternar el pin de señal entre un estado alto y bajo con pequeñas pausas, para producir el sonido de la alarma.

- Servidor Web HTTP Básico con Flask:
  - Se desarrolla un servidor web HTTP básico utilizando el framework Flask de Python en la Raspberry Pi.
  - Este servidor se ejecuta en la red Wi-Fi local.
  - Los usuarios pueden acceder a la página web del dispositivo a través de la dirección IP local asignada a la Raspberry Pi (por ejemplo, 192.168.1.250) desde sus navegadores.

- Comunicación Cliente-Servidor mediante AJAX (Unidireccional):
  - Una vez que el navegador del usuario carga la página web, se envían solicitudes keep-alive periódicamente (cada cinco segundos) desde el navegador (cliente) al servidor web de la Raspberry Pi.
  - La respuesta a cada solicitud keep-alive se considera un latido para confirmar que la conexión con el servidor sigue activa.

- La Raspberry Pi aprovecha estas respuestas para enviar el estado del sensor y los datos al usuario, lo que permite actualizaciones en vivo en la interfaz web. Esto establece una comunicación unidireccional del servidor al usuario. La interfaz web muestra el estado de la detección de movimiento y el estado de la conexión.

- Interacción del Usuario (Comunicación Bidireccional):
  - Se añade un botón o interruptor en la página web que permite a los usuarios controlar los actuadores.
  - Específicamente, este botón permite desactivar la alarma (el buzzer), logrando así una comunicación bidireccional entre el usuario y el servidor.

- Configuración del Entorno:
  - Se requiere una tarjeta SD de al menos 8GB para instalar el sistema operativo de la Raspberry Pi.
  - Para acceder remotamente a la Raspberry Pi (por ejemplo, mediante Escritorio Remoto), es necesario habilitar SSH, ya que está deshabilitado por defecto en la Raspberry Pi 3. Esto se hace a través del comando `sudo raspi-config` en la terminal.

Inicialmente, el servidor funciona localmente, lo que significa que los usuarios deben estar conectados a la misma red Wi-Fi que la Raspberry Pi para acceder al sistema. Este proyecto establece la base para futuras secciones del proyecto que explorarán protocolos de comunicación más avanzados como WebSockets y MQTT, así como la seguridad y el despliegue en la nube.

3.3.1. Conexión PIR y Zumbador con Raspberry PI

Aquí se detalla la conexión y el funcionamiento técnico del Sensor PIR y el Zumbador con la Raspberry Pi en el contexto de un dispositivo IoT detector de movimiento, destacando cómo estos componentes interactúan para formar un sistema funcional.

Conexión y Funcionamiento Técnico del Sensor PIR con Raspberry Pi El Sensor PIR (Infrarrojo Pasivo) es el módulo principal de detección de movimiento del proyecto.

- Pines y Conexión a Raspberry Pi:
  - El módulo PIR tiene tres pines principales: Tierra (Ground), VCC (5V) para alimentación, y un pin de Salida (Output).
  - Este pin de salida proporciona un nivel lógico alto si se detecta un objeto y viceversa (nivel bajo si no hay detección).
  - El sensor PIR se conecta directamente a la Raspberry Pi.

- Principio de Detección:
  - Es un sensor piroeléctrico (elemento RE200 BL) que genera energía al exponerse al calor.
  - Detecta el movimiento porque los cuerpos humanos o animales emiten energía calorífica en forma de radiaciones infrarrojas.
  - Es pasivo porque no emite energía, solo detecta la energía emitida por otros objetos.

- Configuraciones Ajustables:
  - Dispone de dos potenciómetros: uno para ajustar la sensibilidad del sensor y otro para ajustar el tiempo en que la señal de entrada permanece en alto (de 0.3 segundos a 5 minutos).
  - Tiene tres pines adicionales con un jumper para seleccionar modos de disparo:
    - Disparador No Repetible: La salida cambia de alta a baja automáticamente una vez que el tiempo de retardo ha terminado, tras una detección.
    - Disparador Repetible: La salida se mantendrá alta todo el tiempo hasta que el objeto detectado esté presente en el rango del sensor.

- Interacción con Raspberry Pi (Software):

- Una vez conectado, se escribe un código Python básico que detecta las señales altas y bajas al detectar movimiento provenientes del sensor. Estas señales son el input clave para el sistema.

Conexión y Funcionamiento Técnico del Zumbador con Raspberry Pi El Zumbador (Buzzer) es el actuador principal para la alarma en el proyecto.

- Pines y Conexión a Raspberry Pi:
  - El zumbador consta de una caja con tres pines: VCC (5V), Tierra (Ground), y Señal (Signal).
  - Se conecta directamente a la Raspberry Pi.
- Principio de Generación de Sonido:
  - Internamente tiene un elemento piezoeléctrico y un disco metálico vibratorio.
  - Cuando se aplica corriente, el disco metálico se contrae y expande, vibrando y produciendo sonido.
- Control del Tono y Generación de Sonido con Raspberry Pi:
  - Si se cambia la frecuencia de la corriente aplicada, la velocidad de vibración del disco cambia, lo que a su vez cambia el tono (pitch) del sonido. Esto permite generar melodías.
  - Para controlarlo, se genera una onda cuadrada. En términos técnicos, esto implica alternar el pin de señal entre un estado alto y bajo por milisegundos y repetir el proceso.
  - Un código Python básico en la Raspberry Pi se encarga de generar esta onda cuadrada para controlar el zumbador.

Funcionamiento Técnico Conjunto y Rol de la Raspberry Pi La Raspberry Pi actúa como el cerebro que integra y controla ambos componentes:

- Detección y Activación: El Sensor PIR detecta el movimiento y envía una señal a la Raspberry Pi.
- Procesamiento y Alarma: La Raspberry Pi, ejecutando código Python, interpreta esta señal del PIR. Si se detecta movimiento, el código instruye a la Raspberry Pi para que active el Zumbador generando una onda cuadrada en su pin de señal. Esto cumple la función de activar una alarma.
- Comunicación Unidireccional y Bidireccional:
  - Esta interacción permite la comunicación unidireccional del servidor a los usuarios (enviando alertas y activando la alarma).
  - Para la comunicación bidireccional, se añade una funcionalidad para que los usuarios puedan desactivar el zumbador (la alarma) a través de una interfaz web. Esto implica que la Raspberry Pi también recibe comandos del usuario para controlar el actuador.
- Servidor Web HTTP Básico: La Raspberry Pi ejecuta un servidor web HTTP básico con Python Flask en la red local. Este servidor es responsable de:
  - Enviar el estado del sensor y los datos al usuario para actualizaciones en tiempo real.
  - Mostrar un botón de interruptor en la página web que permite a los usuarios desactivar el zumbador, demostrando el control de actuadores.

En resumen, la Raspberry Pi sirve como el intermediario inteligente que recibe información del Sensor PIR (entrada de datos del mundo físico), la procesa con lógica programada en Python, y luego usa esa información para controlar un Zumbador (actuador que interactúa con el mundo físico), así como para comunicarse con los usuarios a través de una interfaz web, logrando un sistema IoT funcional.

3.3.2. Código Python para Detección de Movimiento y Control del Zumbador

Aquí se proporcionan detalles sobre el código Python utilizado para la detección de movimiento del Sensor PIR y el control del Zumbador en el contexto del funcionamiento técnico del Dispositivo IoT de Defensa Antirrobo. La Raspberry Pi actúa como el centro de control, ejecutando este código para integrar ambos componentes.

Aquí se detalla este aspecto:

Código Python para la Detección de Movimiento del Sensor PIR

- Conexión con Raspberry Pi:
  - El Sensor PIR se conecta directamente a la Raspberry Pi.
- Detección de Señales:
  - Se escribe un código Python básico que detecta las señales altas y bajas al detectar movimiento provenientes del sensor.
  - El pin de salida del PIR proporciona un nivel lógico alto si se detecta un objeto y un nivel bajo si no hay detección. El código Python monitorea estos cambios de estado.
- Propósito:
  - Este código es fundamental para la funcionalidad principal del dispositivo, que es detectar movimiento y, basándose en ello, activar una alarma.

Código Python para el Control del Zumbador

- Conexión con Raspberry Pi:
  - El Zumbador también se conecta directamente a la Raspberry Pi.
- Generación de Sonido:
  - El zumbador genera sonido cuando se le aplica corriente, causando la contracción y expansión de un disco metálico vibratorio.
- Control mediante Onda Cuadrada:
  - Para controlar el zumbador y generar el sonido de la alarma, se genera una onda cuadrada.
  - El código Python básico para el zumbador implica alternar el pin de señal del zumbador entre un estado alto y un estado bajo durante ciertos milisegundos y repetir el proceso.
  - Se escribirá un código de una sola línea para controlar el zumbador, sugiriendo una implementación concisa.
- Control del Tono (Pitch):
  - Al cambiar la frecuencia de la corriente aplicada (es decir, la velocidad de alternancia entre alto y bajo en la onda cuadrada), el código Python puede cambiar el tono (pitch) del sonido, permitiendo la generación de diferentes melodías.
- Propósito:
  - El código permite que la Raspberry Pi controle los actuadores, siendo el zumbador el principal para activar una alarma.

Funcionamiento Técnico Conjunto e Integración en Python La Raspberry Pi, con su capacidad de ejecutar código Python, es el cerebro que orquesta la interacción entre el sensor y el actuador:

- Flujo del Sistema:
  - El proceso técnico es el siguiente: se conectan el sensor PIR y el zumbador con la Raspberry Pi. Luego, se escribe el código Python básico que primero detecta las señales altas y bajas al detectar movimiento del PIR y, en consecuencia, controla el zumbador.
- Servidor Web con Flask:

- La Raspberry Pi no solo gestiona los componentes físicos, sino que también ejecuta un servidor web HTTP básico con Python Flask en la red local.
- Este servidor web Python es crucial para la comunicación bidireccional: no solo envía el estado del sensor y los datos al usuario (permitiendo actualizaciones en vivo del estado de detección de movimiento), sino que también permite al usuario desactivar el zumbador (la alarma) a través de un botón de interruptor en la página web.
- La funcionalidad de desactivación del zumbador a través de la interfaz web demuestra cómo el código Python en el servidor maneja las interacciones del usuario para controlar el actuador.

- Programación y Lógica:
  - El código Python es el que implementa la lógica del proyecto, desde la lectura de entradas del sensor hasta la activación de salidas en el actuador y la gestión de la interfaz web para la interacción del usuario.

En resumen, el código Python en la Raspberry Pi es el componente clave que permite al sistema IoT detector de movimiento interpretar los datos del Sensor PIR, activar el Zumbador como alarma sonora y gestionar la interacción con el usuario a través de un servidor web Flask para un control bidireccional.

3.3.3. Servidor WEB FLASK en Raspberry PI (Red Local).

El Servidor Web Flask en la Raspberry Pi desempeña un rol crucial en el funcionamiento técnico del dispositivo IoT detector de movimiento, sirviendo como la interfaz principal para la interacción del usuario y la gestión de los componentes del sistema, todo operando dentro de una red local.

A continuación, se dan los detalles de su funcionamiento técnico:

Implementación y Operación en la Red Local

- Tipo de Servidor:
  - Se trata de un servidor web HTTP básico con Python Flask.
- Plataforma de Ejecución:
  - El servidor Flask se ejecuta directamente en la Raspberry Pi.
- Conectividad de Red:
  - Opera dentro de la red Wi-Fi local. Esto implica que los usuarios que desean interactuar con el dispositivo deben estar conectados al mismo router Wi-Fi para acceder al servidor.
- Acceso del Usuario:
  - La Raspberry Pi es asignada una dirección IP local por el router Wi-Fi (por ejemplo, 192.168.1.250). Los usuarios pueden acceder a la página web del servidor a través de sus navegadores utilizando esta dirección IP local.

Funcionalidades para la Comunicación Bidireccional

El servidor Flask facilita tanto la comunicación unidireccional (servidor a usuario) como la bidireccional (usuario a servidor).

- Envío de Datos del Sensor y Estado en Tiempo Real:
  - Una vez que el usuario carga la página web, el navegador envía peticiones keepalive al servidor de la Raspberry Pi periódicamente cada cinco segundos.
  - Estas peticiones actúan como un latido para confirmar que la conexión con el servidor sigue activa.
  - En respuesta a cada petición keepalive, el servidor envía el estado del sensor y los datos al usuario, lo que permite que el usuario reciba actualizaciones en vivo del sensor en la página web.
  - La interfaz web mostrará el estado de la detección de movimiento y el estado de la conexión.
- Recepción de Comandos del Usuario para el Control de Actuadores:

- La página web generada por Flask incluye botones de interruptor que permiten a los usuarios controlar los actuadores conectados a la Raspberry Pi.
- Un ejemplo específico es la funcionalidad para desactivar el zumbador (la alarma), lo que establece una comunicación bidireccional completa desde el usuario hacia el servidor.
- Estas acciones del usuario se traducen en solicitudes que el código JavaScript del navegador envía a la aplicación Flask. Por ejemplo, un botón para aplicar cambios de permisos envía una solicitud con el formato grant - user ID - read state - right stick.
- La aplicación Flask debe tener endpoints configurados para recibir y procesar estas solicitudes (por ejemplo, un endpoint para /grant user ID read and write).

Gestión de Usuarios y Permisos (Características Avanzadas)

- Población Dinámica de la Interfaz: El servidor Flask es responsable de enviar detalles adicionales a la página web principal (index.html), como el user ID del usuario actual y una lista de usuarios en línea.
- Obtención de Datos de Usuarios: Para obtener la lista de usuarios en línea, el servidor llama a una función get all logged in users que devuelve un mapa (online user records) con información de cada usuario (nombre, ID, permisos de lectura y escritura).
- Visualización en Plantillas Jinja: Utiliza plantillas Jinja en el index.html para iterar sobre la lista de online user records y mostrar dinámicamente cada usuario en una fila, con botones de interruptor para otorgar permisos de lectura y escritura. Los estados de los interruptores se determinan por variables checked o unchecked enviadas desde el servidor Python.
- Control de Acceso (Admin/No-Admin): El servidor implementa lógica para asegurar que el panel de control completo solo sea visible para 'usuarios administradores. Esto se logra añadiendo una sentencia if en el código HTML basada en el user ID proporcionado por el servidor.
- Procesamiento de Solicitudes de Permisos: Cuando el servidor Flask recibe una solicitud grant (por ejemplo, de un botón aplicar), verifica si el remitente es un usuario administrador. Si la verificación es exitosa, procede a almacenar los permisos de lectura y escritura del usuario en la base de datos y, si es necesario, a llamar al servidor PubNub para otorgar estos permisos específicos al usuario.

En resumen, el servidor web Flask en la Raspberry Pi es el núcleo técnico que une los sensores y actuadores con la interfaz de usuario, permitiendo la monitorización en tiempo real y el control de los dispositivos IoT a través de una red local, además de gestionar aspectos más complejos como la autenticación y los permisos de usuario.

3.3.4. Solisitud de HTTP del Navegador del Usuario

Las solicitudes HTTP del navegador del usuario son un componente fundamental en el funcionamiento técnico del dispositivo IoT detector de movimiento, ya que constituyen el principal medio de interacción entre el usuario y el servidor web Flask que se ejecuta en la Raspberry Pi. Estas solicitudes permiten la monitorización del estado del sensor y el control de los actuadores del sistema.

Aquí se dan los detalles de este aspecto:

Acceso Inicial a la Página Web

- Petición Inicial: El funcionamiento técnico comienza cuando el usuario introduce la dirección IP local de la Raspberry Pi (por ejemplo, 192.168.1.250) en su navegador. Al hacerlo, el usuario puede ir a esta dirección desde su navegador y solicitar una página web.
- Respuesta del Servidor: Como respuesta a esta solicitud HTTP, el servidor web Flask envía la página web principal (index.html) al navegador del usuario.

Solicitudes Keepalive para Actualizaciones en Vivo

- Comunicación Unidireccional (Servidor a Usuario): Una vez que el usuario ha cargado la página web, el navegador inicia un proceso de envío de peticiones keepalive al servidor web de la Raspberry Pi periódicamente cada cinco segundos.

- Propósito: Estas solicitudes actúan como un latido para asegurar que la conexión con el servidor aún existe (el servidor sigue vivo).
- Envío de Datos: En respuesta a cada petición keepalive, el servidor permite que el servidor envíe el estado del sensor y los datos al usuario, lo que facilita las actualizaciones en vivo del sensor en la página web del usuario. La página mostrará el estado de la detección de movimiento y el estado de la conexión.
- Técnica Subyacente: Este mecanismo de keepalive es una forma de comunicación que se asemeja al long polling de AJAX, mencionada como la técnica inicial del proyecto antes de la posible adopción de PubNub.

Solicitudes de Control para Actuadores

- Comunicación Bidireccional (Usuario a Servidor): El navegador del usuario también envía solicitudes HTTP para permitir la interacción del usuario con los actuadores. Se implementa un botón de interruptor en la página web que permite a los usuarios controlar los actuadores.
- Ejemplo Específico (Desactivar Alarma): Un caso de uso clave es la capacidad de desactivar el zumbador (la alarma). Cuando el usuario interactúa con estos botones en la interfaz web, el código JavaScript del navegador genera y envía una solicitud HTTP al servidor Flask.
- Formato de la Solicitud: Por ejemplo, una solicitud para aplicar cambios de permisos se envía como grant - ID de usuario - estado de lectura - estado de escritura. Estas son solicitudes POST enviadas al servidor.
- Endpoint del Servidor: La aplicación Flask en la Raspberry Pi tiene un endpoint para recibir esta solicitud, por ejemplo, /grant user ID read and write.

Gestión de Permisos (Solicitudes Grant)

- JavaScript a Flask: Específicamente, el archivo main.js en el navegador del usuario contiene una función que escucha los botones de interruptor. Si un botón de interruptor comienza con XS, se extrae el ID del usuario, se lee el estado de los interruptores de lectura y escritura, y se envía la solicitud como grant - ID de usuario - estado de lectura - estado de escritura al servidor Flask.
- Verificación de Admin: Antes de procesar los permisos, la aplicación Flask es capaz de verificar si esta solicitud grant proviene de un usuario administrador.
- Almacenamiento y Notificación: Si el acceso es concedido, el servidor almacena los permisos de lectura y escritura en la base de datos y puede llamar al servidor PubNub para otorgar estos permisos específicos al usuario.

Consideraciones de Seguridad

- Cifrado SSL/TLS: Inicialmente, las solicitudes HTTP del navegador podrían ser inseguras. Sin embargo, se describen un proceso para asegurar nuestro dominio personalizado con certificados SSL/TLS de Let's Encrypt.
- Redirección a HTTPS: Una vez configurado, el servidor está diseñado para redirigir el tráfico HTTP a HTTPS.
- Comunicación Cifrada: Esto significa que cada cliente conectado con este servidor tendrá una comunicación cifrada de extremo a extremo, garantizando la seguridad de los datos enviados entre el navegador del usuario y el servidor IoT.
- Reglas de Seguridad: Se requiere configurar las reglas de seguridad de entrada para HTTPS (puerto 443) en el servidor remoto (AWS EC2 en el ejemplo) para permitir que el navegador se conecte de forma segura.

En resumen, las solicitudes HTTP del navegador del usuario son el mecanismo esencial para la interacción remota con el dispositivo IoT. Permiten desde la carga inicial de la interfaz, la recepción continua de datos del sensor, hasta el envío de comandos de control y la gestión de permisos, todo ello protegido mediante SSL/TLS para garantizar la seguridad de la comunicación.

3.3.5. Solisitudes KEEPALIVE Periódicas (cada 5s)

Las solicitudes Keepalive Periódicas (Cada 5 segundos) son un elemento técnico fundamental en el funcionamiento del dispositivo IoT detector de movimiento, especialmente para la comunicación unidireccional del servidor al usuario y la monitorización en tiempo real.

A continuación, se dan los detalles de su funcionamiento técnico:

Mecanismo y Frecuencia de las Solicitudes Keepalive

- Origen de la Solicitud: Una vez que el usuario ha cargado la página web desde su navegador (accediendo a la dirección IP local de la Raspberry Pi), el 'navegador del usuario ' comienza a enviar estas solicitudes.
- Destino: Las peticiones se envían 'al servidor web de la Raspberry Pi ' .
- Periodicidad: Estas solicitudes se envían 'periódicamente cada cinco segundos ' .
- Propósito Inicial (Latido): La respuesta a cada solicitud keepalive se considera un 'latido para asegurarse de que la conexión con el servidor aún existe ' , confirmando que 'el servidor sigue vivo ' .

Función en la Actualización de Datos en Tiempo Real

- Envío de Datos del Sensor: Más allá de solo verificar la conexión, la función principal de estas solicitudes en el contexto del proyecto es permitir que el servidor 'envíe el estado del sensor y los datos al usuario ' en respuesta a cada keepalive.
- Actualizaciones en Vivo: Este mecanismo asegura que el usuario reciba 'actualizaciones en vivo del sensor ' en la página web. La interfaz web mostrará el 'estado de la detección de movimiento ' y el 'estado de la conexión ' .
- Comunicación Unidireccional: Esto establece una 'comunicación unidireccional del servidor al usuario ' , un objetivo clave del proyecto.

Contexto Técnico y Evolución

- Tecnología Subyacente: Inicialmente, para la comunicación entre el cliente y el servidor, el proyecto utilizaba la 'técnica AJAX '. Las solicitudes keepalive periódicas son una implementación que se asemeja al *long polling* de AJAX para simular la comunicación en tiempo real en la interfaz web.
- Limitaciones y Mejoras: Se sugieren que, aunque este método permite las actualizaciones en vivo, el proyecto evolucionaría para usar protocolos de comunicación más avanzados y ligeros para IoT, como WebSockets y MQTT , y específicamente la plataforma PubNub , en lugar del *long polling* de AJAX. PubNub se usaría como el principal protocolo de comunicación en fases posteriores. Esto implica que, si bien las solicitudes keepalive con AJAX fueron el punto de partida, tecnologías más eficientes se adoptarían para mejorar la latencia y la bidireccionalidad.
- Acceso Local: Es importante recordar que este servidor, en esta etapa del proyecto, está 'funcionando localmente ' . Por lo tanto, los usuarios solo pueden acceder a él 'desde dentro de la red ' , es decir, deben estar 'conectados al mismo router Wi-Fi ' que la Raspberry Pi para que estas solicitudes keepalive y la interacción general funcionen.

En resumen, las solicitudes Keepalive Periódicas cada 5 segundos son un pilar técnico para la monitorización en tiempo real del estado de los sensores en el dispositivo IoT. Permiten al navegador del usuario mantener una conexión activa con el servidor Flask en la Raspberry Pi y recibir actualizaciones constantes y en vivo sobre la detección de movimiento y el estado del sistema, facilitando la comunicación unidireccional inicial del servidor al usuario.

3.3.6. Botón en Página WEB para Controlar Actuadores (Desactivar Zumbador)

El botón en la página web para controlar actuadores , específicamente para desactivar el zumbador (alarma), es un componente técnico crucial que habilita la comunicación bidireccional del usuario al servidor en el dispositivo IoT detector de movimiento. Permite que el usuario interactúe directamente con el sistema para controlar sus funciones.

A continuación, se detalla el funcionamiento técnico: **Propósito y Presentación en la Interfaz de Usuario**

- Comunicación Bidireccional: Uno de los objetivos del proyecto es añadir una funcionalidad en la que los usuarios puedan comunicarse con el servidor, cumpliendo con la comunicación bidireccional. El botón de control es la interfaz para esta interacción.
- Caso de Uso Principal: El uso más destacado de esta funcionalidad es la capacidad de 'desactivar el zumbador ' , que actúa como la alarma del sistema.
- Ubicación en la Web: Una vez que el usuario accede a la página web ('index.html'), se añade un 'botón de interruptor ' o *switch button* en la interfaz para permitir el control de los actuadores. Inicialmente, se describen una etapa con el 'estado de movimiento ' y un botón para controlar el zumbador.
- Control de Permisos (Panel Admin): En una etapa posterior del proyecto, se implementa un panel de control visible solo para usuarios administradores, donde se listan los usuarios en línea y, junto a cada nombre, se encuentran botones de interruptor para 'conceder permisos de lectura y escritura ' . También hay un 'botón para aplicar los cambios '. Estos botones tienen IDs dinámicos que incorporan el ID del usuario (por ejemplo, 'read user ID', 'write user ID', 'access user ID').

Mecanismo de Envío de Solicitudes (Front-end: JavaScript)

- Interacción del Usuario: Cuando el usuario interactúa con estos botones en la página web, el código JavaScript ('main.js') del navegador entra en acción.
- Detección y Procesamiento: El JavaScript tiene un método que 'escucha ' cualquier botón de interruptor en el dashboard. Si la ID de un interruptor comienza con 'XS ' (para el caso de los permisos de acceso), se extrae el ID del usuario de la ID del botón.
- Formación de la Solicitud: Luego, se lee el estado actual del interruptor de lectura y el interruptor de escritura. Finalmente, se 'envía la solicitud como 'grant - ID de usuario - estado de lectura - estado de escritura' ' .
- Tipo de Solicitud: Esta solicitud se envía al servidor como una 'solicitud POST ' utilizando un método como 'send event'.

Procesamiento en el Servidor (Back-end: Flask)

- Endpoint en Flask: La aplicación Flask en la Raspberry Pi está configurada con un 'endpoint para recibir esta solicitud ' . Un ejemplo de este endpoint sería '/grant user ID read and write'.
- Verificación de Admin: Una vez recibida la solicitud, es recomendable 'verificar si esta solicitud grant proviene de un usuario administrador' antes de procesarla. Si no es un administrador, la respuesta puede ser 'acceso denegado '.
- Actualización de la Base de Datos y PubNub: Si la solicitud es válida y proviene de un administrador, el servidor realiza dos acciones principales:
  - 'Almacenar los permisos de lectura y escritura del usuario en la base de datos ' .
  - 'Llamar al servidor PubNub para conceder estos permisos específicos al usuario ' . Esto se hace a través de la funcionalidad de 'Access Manager de PubNub '.
- Respuesta al Cliente: El servidor envía una respuesta al navegador del usuario indicando si el acceso fue 'concedido ' o 'denegado'. Si el acceso es concedido, el JavaScript en el cliente puede 'resetear la suscripción al canal '.

Funcionamiento Técnico General y Seguridad

- Acceso Local: Es fundamental recordar que, en esta etapa del proyecto, el servidor Flask funciona localmente, lo que significa que los usuarios deben estar 'conectados al mismo router Wi-Fi ' que la Raspberry Pi para poder acceder a la página web y utilizar estos botones de control.
- Seguridad (SSL/TLS): Todas las solicitudes HTTP enviadas desde el navegador del usuario, incluyendo las que provienen de los botones de control, están sujetas a las medidas de seguridad implementadas. Una vez que el dominio personalizado se asegura con certificados SSL/TLS de Let's Encrypt y se configura la redirección de HTTP a HTTPS, 'cada cliente conectado con este servidor

tendrá una comunicación cifrada de extremo a extremo ' . Esto garantiza la confidencialidad e integridad de los comandos de control enviados al dispositivo IoT. Se requiere configurar reglas de seguridad de entrada para HTTPS (puerto 443) en el servidor remoto para permitir esta conexión segura.

En síntesis, el botón en la página web para controlar actuadores representa una sofisticada interacción que comienza con una acción del usuario en la interfaz web, se traduce en una solicitud HTTP POST gestionada por JavaScript, es procesada por un servidor Flask que verifica permisos y actualiza la información en la base de datos y en PubNub, todo ello bajo una comunicación cifrada de extremo a extremo para garantizar la seguridad.

3.3.7. Acceso solo Dentro de la Red Local

Se enfatizan que el Acceso Solo Dentro de la Red Local es una característica fundamental del funcionamiento técnico inicial del servidor IoT detector de movimiento, que se ejecuta en la Raspberry Pi. Esta limitación define cómo los usuarios pueden interactuar con el dispositivo en las primeras etapas del proyecto.

Aquí se dan los detalles de este punto en el contexto del funcionamiento técnico:

Naturaleza del Servidor y Requisito de Acceso Local

- Servidor Local en Raspberry Pi: El proyecto comienza con la creación de un servidor web HTTP básico de Python Flask en la Raspberry Pi. Este servidor está diseñado para 'ejecutarse en nuestra red Wi-Fi local ' .
- Acceso Exclusivo Dentro de la Red: Debido a que el servidor 'funciona localmente ' , los usuarios 'solo pueden acceder a él desde dentro de la red ' . Esto significa que, para interactuar con el servidor, los usuarios 'deben estar conectados al mismo router Wi-Fi ' que la Raspberry Pi.
- Acceso por Dirección IP: Para acceder a la página web del servidor, el usuario debe ir a la dirección IP local asignada a la Raspberry Pi por el router Wi-Fi (por ejemplo, '192.168.1.250') desde su navegador.

Implicaciones en las Funcionalidades Técnicas

- Este requisito de acceso local afecta directamente cómo funcionan varios componentes técnicos del sistema:
  - Carga de la Página Web: La interfaz web ('index.html'), que muestra el estado del sensor y los controles, solo puede ser cargada por usuarios que están en la misma red local.
  - Solicitudes Keepalive Periódicas: Las 'solicitudes keepalive periódicas ' que el navegador del usuario envía al servidor cada cinco segundos para verificar la conexión y recibir actualizaciones en tiempo real, solo son posibles si el usuario está conectado al mismo router Wi-Fi [conversación previa].
  - Control de Actuadores (Botón para Desactivar Zumbador): De manera similar, la funcionalidad de usar un botón en la página web para controlar actuadores, como 'desactivar el zumbador ' , también está restringida a usuarios dentro de la misma red local. Las solicitudes POST generadas por JavaScript para controlar estos actuadores solo llegarán al servidor Flask si el cliente está en la red [conversación previa].

Evolución del Proyecto más Allá del Acceso Local

- Aunque el acceso local es la configuración inicial y fundamental, el proyecto y el proyecto están diseñados para evolucionar y superar esta limitación:
  - Despliegue en la Nube y Dominio Personalizado: En secciones posteriores, el proyecto se centra en 'desplegar el servidor IoT en la nube de AWS ' y asegurar un 'nombre de dominio personalizado ' con certificados SSL/TLS. Esto indica una transición de un entorno estrictamente local a una infraestructura accesible a través de Internet.
  - Seguridad Mejorada: Al migrar a un servidor remoto con un dominio seguro (HTTPS), 'cada cliente conectado con este servidor tendrá una comunicación cifrada de extremo a extremo ' . Esto es un contraste con la seguridad implícita (o su ausencia) en una red local inicial, donde el acceso es intrínsecamente más limitado por la geografía.

En resumen, el acceso solo dentro de la red local es una característica definitoria del funcionamiento técnico inicial del servidor IoT, donde la Raspberry Pi actúa como un servidor Flask básico. Requiere que todos los usuarios estén 'conectados al mismo router Wi-Fi ' para interactuar con la interfaz web, recibir datos en tiempo real y controlar actuadores. Si bien esta configuración simplifica el inicio del proyecto, se demuestran que el objetivo final es una solución IoT más robusta y accesible globalmente a través del despliegue en la nube y protocolos seguros.

3.4. Configuración de Raspberry PI

En el contexto más amplio del Primer Proyecto IoT: Detector de Movimiento (etapa 2 del proyecto 'Internet de las Cosas con Python y Raspberry Pi'), la configuración de la Raspberry Pi es fundamental, ya que actúa como el 'cerebro ' central del sistema.

Aquí se detallan los siguientes aspectos clave para la Configuración de la Raspberry Pi:

- Requisitos de Almacenamiento:
  - Se requiere una tarjeta SD de al menos 8GB para instalar el sistema operativo de la Raspberry Pi.
- Instalación del Sistema Operativo:
  - Se recomienda seguir las instrucciones oficiales de configuración de Raspberry Pi disponibles en su sitio web para instalar el sistema operativo.
- Acceso Remoto y SSH:
  - Es posible realizar una conexión de escritorio remoto a la Raspberry Pi, una opción que el instructor utilizará.
  - Para la Raspberry Pi 3 , el protocolo SSH está deshabilitado por defecto .
  - Para habilitar SSH, lo cual es necesario para el acceso remoto por escritorio, los usuarios deben ir a la terminal en la Raspberry Pi, escribir el comando `sudo raspi-config` y seguir las instrucciones para activarlo. El uso de Putty para conectarse al servidor, como se menciona en la etapa 5, implica que se utilizará una conexión SSH.
- Conectividad de Red:
  - La Raspberry Pi alojará un servidor web HTTP básico con Flask en Python que se ejecutará en la red Wi-Fi local .
  - Los usuarios podrán acceder a la página web del dispositivo desde su navegador a través de la dirección IP local asignada a la Raspberry Pi (por ejemplo, 192.168.1.250).
  - Es importante destacar que el servidor funcionará localmente, lo que significa que los usuarios deben estar conectados al mismo router Wi-Fi que la Raspberry Pi para poder acceder al sistema.
- Una vez configurada, la Raspberry Pi se encargará de:
  - Conectar el sensor PIR y el buzzer .
  - Ejecutar el código Python básico para detectar señales del sensor y controlar el buzzer.
  - Alojar la aplicación web que permitirá la monitorización y el control del dispositivo.

Esta configuración sienta las bases para el desarrollo del proyecto, permitiendo la interacción entre el hardware, el software y la interfaz de usuario a través de la red local.

3.4.1. Tarjeta SD de 8GB Mínimo (OS)

Aquí se detalla un requisito específico para la Tarjeta SD de 8GB Mínimo en el contexto de la Configuración de Raspberry Pi .

En concreto, para configurar la Raspberry Pi y comenzar con el proyecto, se establece que:

- Requisito de Tarjeta SD: Se requiere una 'tarjeta SD de 8GB como mínimo ' .

- Propósito (Instalación del Sistema Operativo): El propósito principal de esta tarjeta SD es la 'instalación del sistema operativo' en ella.

Esto significa que la tarjeta SD es un componente esencial para el funcionamiento básico de la Raspberry Pi, ya que alberga el sistema operativo que permite que el dispositivo arranque y ejecute el servidor IoT y los scripts de Python. Sin esta tarjeta SD con el sistema operativo instalado, la Raspberry Pi no podría iniciar ni realizar ninguna de las tareas del proyecto.

Además, la configuración general de la Raspberry Pi mencionada incluye pasos como:

- Seguir las instrucciones oficiales de configuración de Raspberry Pi para instalar el sistema operativo en la tarjeta SD.
- Posiblemente necesitar un monitor HDMI o realizar una conexión de escritorio remoto (Remote Desktop) a la Raspberry Pi para completar la configuración inicial.
- Habilitar SSH si está deshabilitado por defecto (como en Raspberry Pi 3) para permitir el escritorio remoto.

3.4.2. Monitor HDMI O Escritorio Remoto

Para la Configuración de Raspberry Pi en el contexto del proyecto de IoT, se indican que es esencial contar con un método para interactuar con la Raspberry Pi, ya sea a través de un Monitor HDMI o mediante Escritorio Remoto .

Aquí se detalla este aspecto:

- Necesidad de un Método de Interacción:
  - Para la configuración inicial de la Raspberry Pi, que incluye seguir las instrucciones oficiales de configuración, se requiere una forma de visualizar y controlar el sistema.
    - Opción 1: Monitor HDMI: Se menciona que 'puede que necesite un monitor HDMI ' para realizar la configuración. Esto implica una conexión física directa a la Raspberry Pi para acceder a su interfaz gráfica o terminal.
    - Opción 2: Escritorio Remoto: Si no se dispone de un monitor HDMI, el 'escritorio remoto a la Raspberry Pi ' es una alternativa viable que el instructor mismo utilizará. Existen 'instrucciones sencillas en el video de YouTube proporcionado ' para llevar a cabo esta conexión.
- Habilitación de SSH para Escritorio Remoto (Específico para Raspberry Pi 3): Un punto técnico crucial es que, para la Raspberry Pi 3, el protocolo SSH (Secure Shell) 'está deshabilitado por defecto ' . Para poder realizar el escritorio remoto, es 'necesario habilitar el SSH'.
- Procedimiento para Habilitar SSH: Para activar SSH, el usuario debe ir a la terminal en la Raspberry Pi y 'escribir `sudo raspi-config` y seguir las instrucciones para activar el SSH ' .

En resumen, la capacidad de ver y controlar la Raspberry Pi durante su configuración inicial es fundamental, ofreciendo la flexibilidad de elegir entre una conexión física con un Monitor HDMI o una conexión virtual a través de Escritorio Remoto , siendo este último un proceso que en ciertos modelos como la Raspberry Pi 3 requiere la habilitación explícita de SSH .

3.4.3. Habilitar SSH

En el contexto más amplio de la Configuración de Raspberry Pi para el proyecto de IoT, se destacan la importancia de habilitar SSH (`sudo raspi-config`), especialmente cuando se utiliza el 'Escritorio Remoto' como método de interacción.

Aquí se detalla este aspecto:

- Necesidad del Escritorio Remoto: Si un usuario no dispone de un monitor HDMI para configurar su Raspberry Pi, el Escritorio Remoto es una alternativa viable para interactuar con el dispositivo. De hecho, el instructor mismo utilizará este método.
- SSH Deshabilitado por Defecto (**en Raspberry Pi 3**): Un punto técnico crucial es que, para modelos específicos como la Raspberry Pi 3 , el protocolo SSH (Secure Shell) 'está deshabilitado por defecto ' .

- Requisito para el Escritorio Remoto: Para poder realizar la conexión de Escritorio Remoto a la Raspberry Pi, es 'necesario habilitar el SSH '.
- Procedimiento para Habilitar SSH: Se proporcionan instrucciones claras sobre cómo activar SSH:
  - El usuario debe 'ir a la terminal en la Raspberry Pi'.
  - Una vez en la terminal, debe escribir `sudo raspi-config`.
  - Finalmente, debe 'seguir las instrucciones para activar el SSH ' dentro de la interfaz de configuración de 'raspi-config'.

En síntesis, la habilitación de SSH es un paso fundamental en la configuración de la Raspberry Pi , particularmente cuando se opta por el acceso remoto en lugar de una conexión física con un monitor HDMI. Este proceso asegura la capacidad de establecer una conexión segura y de controlar la Raspberry Pi a distancia, siendo un requisito indispensable para ciertos modelos como la Raspberry Pi 3.

Capítulo 4

PROTOCOLOS DE COMUNICACIóN IoT

En el contexto más amplio del sistema **Internet de las Cosas con Python y Raspberry Pi, (*Monitor Remoto*)**, los Protocolos de Comunicación IoT son un pilar fundamental para el desarrollo de sistemas que permiten a los dispositivos interactuar entre sí y con los usuarios. El sistema explora varias tecnologías, desde las más básicas hasta las más avanzadas y seguras, para establecer una comunicación eficaz y robusta. A continuación, se dan los detalles de estos protocolos:

- 1. Comunicación Inicial con AJAX 'Keepalive' (Primer Proyecto IoT)
  - Descripción y Uso: En la etapa 2, durante el 'Primer Proyecto IoT: Detector de Movimiento', se introduce una forma básica de comunicación cliente-servidor utilizando solicitudes AJAX 'keepalive'. Una vez que el usuario carga la página web en su navegador, esta envía solicitudes 'keepalive' periódicamente (cada cinco segundos) al servidor web de la Raspberry Pi.
  - Propósito: Estas solicitudes actúan como un 'latido' para asegurar que la conexión con el servidor (Flask en la Raspberry Pi) está activa. La respuesta a cada solicitud permite al servidor enviar el estado del sensor y otros datos en tiempo real al usuario, logrando así actualizaciones en vivo del sensor. También se utiliza para la comunicación bidireccional, permitiendo al usuario enviar comandos al servidor (por ejemplo, para desactivar una alarma).
  - Contexto y Limitaciones: Esta implementación inicial es efectiva para un servidor que se ejecuta localmente en la misma red Wi-Fi. Sin embargo, se reconoce las 'ventajas y desventajas de la forma en que desarrollamos nuestro último proyecto en la etapa 2', lo que implica que esta técnica de 'long-polling' de AJAX tiene limitaciones, especialmente para aplicaciones IoT a gran escala, en tiempo real y con requisitos de baja latencia.
- 2. WebSockets: Comunicación Interactiva y de Baja Latencia
  - Introducción: En la etapa 3 ('Protocolos de Comunicación y Seguridad IoT'), se profundiza en WebSockets como una de las 'tecnologías más famosas y adoptadas en IoT para protocolos de comunicación'.
  - Características y Funcionamiento: WebSockets permiten abrir una sesión de comunicación interactiva y persistente sobre una única conexión TCP. A diferencia del polling tradicional (como el 'keepalive' de AJAX), los clientes reciben actualizaciones solo cuando ocurren, sin necesidad de preguntar constantemente al servidor. Esto los hace bidireccionales y dúplex completo, donde ambas partes pueden enviar mensajes de forma independiente. Su baja latencia es una razón clave por la que son ampliamente adoptados en aplicaciones IoT.
  - Proceso de Conexión: Un cliente inicialmente realiza una solicitud HTTP para solicitar al servidor que actualice el protocolo a WebSockets. Una vez que el servidor reconoce el handshake, la sesión se mantiene abierta y persistente hasta que una de las partes la cierra. Los paquetes de datos enviados durante esta sesión son muy pequeños, de 2 a 14 bytes.
- 3. MQTT (Message Queuing Telemetry Transport): Ligero y Basado en Publicación/Suscripción
  - Introducción: También en la etapa 3, se presenta MQTT como otro protocolo clave.

- Características: Es un protocolo de transferencia de mensajes ligero y eficiente en ancho de banda, diseñado para la comunicación máquina a máquina (M2M) e Internet de las Cosas. Utiliza solo 2 bytes de sobrecarga.
- Modelo Publicación/Suscripción: MQTT opera mediante un modelo de publicación y suscripción. Los dispositivos publican datos en 'tópicos' específicos, y otros dispositivos suscritos a esos tópicos reciben esa información. Esto permite escenarios de transmisión de datos uno a uno, uno a muchos y muchos a muchos.
- Funcionamiento: El punto central de comunicación es el broker MQTT, que se encarga de despachar todos los mensajes entre remitentes y receptores. Cada cliente que publica un mensaje al broker incluye un 'tópico' que sirve como información de enrutamiento. El broker reenvía el mensaje a los receptores suscritos a ese tópico.
- Relación con WebSockets: Se aclara que MQTT y WebSockets son cosas diferentes, pero pueden usarse juntos: MQTT corre sobre la capa TCP/IP y puede utilizarse sobre la capa WebSocket. Se explica con la analogía de que MQTT es un 'servicio de entrega' (como DHL) y WebSockets proveen las 'carreteras y camiones'. Así, un paquete de datos MQTT se empaqueta dentro de un 'sobre' WebSocket, que a su vez se envuelve en un 'sobre' TCP/IP.
- Escalabilidad y Desventajas: El modelo de publicación/suscripción hace que MQTT sea altamente escalable, ya que los clientes no necesitan conocerse entre sí, solo comunicarse a través de tópicos. Sin embargo, su principal inconveniente es la dependencia del broker central: 'si el broker muere, toda la comunicación se interrumpirá'.

- 4. PubNub: Un Servicio de Comunicación en Tiempo Real para Reemplazar AJAX 'Long-Polling'
  - Introducción y Contexto: En la etapa 4, se decide 'reconstruir nuestro sistema principal para usar PubNub como el protocolo de comunicación principal en lugar de AJAX long-polling'. PubNub es, por lo tanto, una solución en tiempo real que reemplaza la técnica básica utilizada en el primer sistema (Detector de Movimiento).
  - Uso este sistema: PubNub se emplea no solo para la comunicación principal en tiempo real, sino también, y de manera crucial, en la etapa 6 para la gestión de acceso de usuarios. La funcionalidad 'Access Manager' de PubNub permite a los usuarios administradores 'conceder permisos de lectura y escritura en tiempo real a todos los usuarios no administradores y a los dispositivos'.
  - Integración con Flask: El servidor Flask interactúa con PubNub para otorgar estos accesos, almacenando los permisos en una base de datos y llamando al servidor PubNub para actualizar los permisos en tiempo real.
- Conclusión en un Contexto Amplio:
  - La evolución de los protocolos de comunicación a lo largo del desarrollo de las etapas del sistema IoT, refleja un aprendizaje progresivo:
    - 1. Inicio práctico con limitaciones: El proyecto inicial usa AJAX 'keepalive' para una comunicación básica, pero con alcance local y sin la eficiencia de un protocolo en tiempo real.
    - 2. Conceptos teóricos avanzados: Las etapas posteriores introducen WebSockets y MQTT para superar las limitaciones del AJAX, ofreciendo comunicación bidireccional, baja latencia, eficiencia y escalabilidad.
    - 3. Implementación con servicios en la nube: Finalmente, PubNub se utiliza como una solución de comunicación en tiempo real ya implementada, que facilita el despliegue en la nube (AWS) y la gestión de permisos de manera segura y escalable, superando las complejidades de implementar WebSockets/MQTT desde cero en un servidor propio y permitiendo un enfoque en la lógica de negocio y seguridad.

En resumen, se demuestra entonces que, en el desarrollo de sistemas IoT con Python y Raspberry Pi, la elección del protocolo de comunicación es fundamental y debe evolucionar desde soluciones básicas y locales hasta sistemas seguros, escalables y en tiempo real que pueden desplegarse en la nube y gestionar complejos permisos de usuarios y dispositivos.

4.1. Websockets

Se proporciona una discusión detallada sobre WebSockets en el contexto de los Protocolos de Comunicación IoT, destacando su funcionalidad, ventajas e integración con otros protocolos como MQTT y PubNub. Aquí se desglosa lo que se dice sobre WebSockets:

- 1. Definición y Propósito Principal
  - Tecnología Avanzada para Comunicación Interactiva: WebSockets se describe como una tecnología avanzada que permite abrir sesiones de comunicación interactivas mediante la apertura de una única conexión TCP.
  - Actualizaciones Sin Petición Explícita: Permiten que los clientes reciban actualizaciones solo cuando ocurren, sin necesidad de que el cliente esté 'preguntando' o sondeando al servidor (polling). Esto significa que el cliente puede enviar mensajes al servidor y recibir respuestas basadas en eventos sin tener que solicitar actualizaciones constantemente.
  - Comunicación Bi-direccional y Full Duplex: WebSockets son fundamentalmente bi-direccionales y full duplex , lo que significa que ambas partes (cliente y servidor) pueden enviar mensajes de forma independiente entre sí.
- 2. Cómo Funcionan los WebSockets
  - Proceso de Negociación (Handshake): Inicialmente, un cliente realiza una solicitud HTTP al servidor, pidiéndole que actualice el protocolo a WebSockets.
  - Conexión Persistente: Si el servidor reconoce y acepta esta negociación (handshake), la sesión se mantiene abierta y persistente durante todo el tiempo, a menos que alguna de las partes la cierre.
  - Pequeños Paquetes de Datos: Durante una sesión abierta, los paquetes de datos que se envían entre el cliente y el servidor son muy pequeños, con una longitud de trama de 2 a 14 bytes.
- 3. Ventajas y Adopción en IoT
  - Baja Latencia: Debido a su naturaleza bi-direccional y persistente, WebSockets ofrecen baja latencia en la comunicación.
  - Escalabilidad en Aplicaciones en Tiempo Real: Aunque se estandarizaron en 2011 y fueron adoptados inicialmente por aplicaciones como juegos y sistemas de chat, su baja latencia y naturaleza bi-direccional los hacen altamente adoptados en aplicaciones IoT.
  - Componente del ProyectoIoT: El sistema 'Internet de las Cosas con Python y Raspberry Pi' se enfoca en comprender y resolver problemas del mundo real de IoT, y WebSockets es uno de los protocolos de comunicación en tiempo real y ligeros que se estudian en profundidad en la etapa 3.
- 4. Relación con MQTT
  - MQTT sobre WebSockets: Se aclaran que WebSockets y MQTT son cosas diferentes, pero MQTT puede ejecutarse sobre la capa de WebSockets.
  - Analogía de 'Entrega de Servicio': Se utiliza una analogía para explicar su relación. MQTT se considera un 'servicio de entrega' (como DHL), mientras que WebSockets proporcionan la 'infraestructura' o las 'carreteras'.
  - Anidación de Paquetes: Un paquete de datos MQTT se empaqueta dentro de un 'sobre' de WebSocket, que a su vez se envuelve en un 'sobre' TCP/IP antes de ser enviado por Internet. El proceso inverso ocurre al desempaquetarse.

En el contexto más amplio de los Protocolos de Comunicación IoT, WebSockets representa un paso crucial en la evolución de las soluciones de comunicación. Permite el desarrollo de aplicaciones que requieren interactividad en tiempo real y una comunicación eficiente y persistente, superando las limitaciones de los métodos basados en HTTP tradicionales como el polling de AJAX. Su capacidad para actuar como una capa de transporte para otros protocolos, como MQTT, también destaca su flexibilidad y su rol integral en la construcción de ecosistemas IoT robustos y escalables.

4.1.1. Actualizaciones solo Cuando Ocurren (sin sondeo)

Aquí se explica de manera detallada el concepto de 'Actualizaciones Solo Cuando Ocurren (Sin Sondeo)' en el contexto más amplio de WebSockets.

En el ámbito de la comunicación en el Internet de las Cosas (IoT), se destaca que los WebSockets permiten a los clientes recibir actualizaciones 'solo cuando ocurren' y, crucialmente, 'sin que el cliente pregunte al servidor'. Esto contrasta con métodos más tradicionales que requieren que el cliente 'sondee' (o 'polee') al servidor periódicamente para verificar si hay nuevas actualizaciones. Las características clave que permiten este modelo de actualización son:

- 1.Comunicación Interactiva Bidireccional: WebSockets es una tecnología avanzada que posibilita abrir una 'sesión de comunicación interactiva' mediante la apertura de una 'única conexión TCP'. Esta sesión se mantiene 'abierta y persistente' a menos que una de las partes la cierre.
- 2.Respuestas Dirigidas por Eventos: Gracias a esta conexión persistente, el cliente puede 'enviar mensajes al servidor y recibir una respuesta impulsada por eventos sin tener que sondear o preguntar al servidor' por actualizaciones. Esto significa que el servidor envía datos al cliente tan pronto como están disponibles, en lugar de esperar una solicitud del cliente.
- 3.Naturaleza Full-Duplex: La comunicación es 'bidireccional y full-duplex', lo que implica que 'ambas partes pueden enviar mensajes independientemente una de la otra'. Esta capacidad permite un flujo de datos más eficiente y en tiempo real.
- Baja Latencia y Eficiencia: Debido a su 'baja latencia y naturaleza bidireccional', WebSockets es una tecnología 'altamente adoptada en aplicaciones de IoT'. El tamaño de los paquetes de datos enviados durante una sesión abierta es 'muy pequeño', de 2 a 14 bytes, lo que contribuye a su eficiencia.
- 4.Proceso de Conexión: Inicialmente, un cliente realiza una 'solicitud HTTP y pide al servidor que actualice al protocolo WebSockets'. El servidor 'reconoce el handshake como respuesta', y luego la sesión permanece abierta.

En resumen, los WebSockets facilitan las 'actualizaciones solo cuando ocurren (sin sondeo)' al establecer una conexión persistente y bidireccional entre el cliente y el servidor. Esto permite que el servidor envíe datos al cliente en tiempo real, tan pronto como estén disponibles, eliminando la necesidad de que el cliente realice solicitudes periódicas, lo cual es fundamental para aplicaciones de IoT que requieren una comunicación eficiente y de baja latencia.

4.1.2. Sesión de Comunicación Interactiva Persistente (una conexión TCP)

Aquí se explica que la Sesión de Comunicación Interactiva Persistente (Una Conexión TCP) es una característica fundamental de los WebSockets que revoluciona la forma en que los clientes y servidores se comunican, especialmente en el ámbito del Internet de las Cosas (IoT). En el contexto más amplio de los WebSockets, se detallan lo siguiente sobre esta sesión:

- 1.Establecimiento de una Conexión Única y Duradera: WebSockets es una tecnología avanzada que permite establecer una 'sesión de comunicación interactiva 'mediante la apertura de una 'única conexión TCP'. Una vez establecida, esta sesión se 'mantiene abierta y persistente' durante todo el tiempo, a menos que una de las partes decida cerrarla.
- 2.Proceso de Conexión Inicial ('Handshake'): La conexión se inicia cuando un cliente envía una 'solicitud HTTP' al servidor, pidiéndole que 'actualice al protocolo WebSockets'. El servidor, al reconocer esta solicitud, envía una respuesta que se conoce como 'handshake', y a partir de ese momento, la sesión se considera abierta y persistente.
- 3.Comunicación Bidireccional y Full-Duplex: La naturaleza de esta conexión persistente permite una comunicación 'bidireccional y full-duplex'. Esto significa que 'ambas partes pueden enviar mensajes independientemente una de la otra'.
- 4.Actualizaciones Basadas en Eventos (Sin Sondeo): Gracias a esta sesión persistente, los clientes pueden 'enviar mensajes al servidor y recibir una respuesta impulsada por eventos sin tener que sondear o preguntar al servidor' por actualizaciones. Como se señalo previamente, esto se traduce en 'actualizaciones solo cuando ocurren', eliminando la necesidad de que el cliente solicite información periódicamente.

- 5.Beneficios en Aplicaciones IoT: Debido a su 'baja latencia y naturaleza bidireccional, la tecnología WebSocket ha sido 'altamente adoptada en aplicaciones de IoT'. La eficiencia también es un factor clave, ya que los paquetes de datos enviados durante una sesión abierta son 'muy pequeños', con una longitud de trama de 2 a 14 bytes.

En resumen, la sesión de comunicación interactiva persistente sobre una única conexión TCP es el corazón del funcionamiento de WebSockets. Permite una interacción en tiempo real, bidireccional y eficiente entre clientes y servidores, facilitando que las actualizaciones se entreguen solo cuando ocurren y sin necesidad de sondeo, lo cual es vital para aplicaciones como el monitoreo y control en el Internet de las Cosas.

4.1.3. Bidireccional Y Full Duplex

Se describen los conceptos de comunicación bidireccional y full duplex principalmente en el contexto de WebSockets. Los WebSockets son una tecnología avanzada que posibilita abrir una sesión de comunicación interactiva manteniendo una única conexión TCP. Esta característica los hace bidireccionales y full duplex. A continuación, se detalla lo que significan estos términos:

- 1.Bidireccional y Full Duplex: Con WebSockets, ambas partes (el cliente y el servidor) pueden enviar mensajes de forma independiente una de la otra. Esto significa que los clientes pueden recibir actualizaciones solo cuando estas ocurren, sin necesidad de que el cliente le pregunte al servidor. Permiten enviar mensajes al servidor y recibir respuestas impulsadas por eventos sin tener que sondear o preguntar al servidor por actualizaciones. Esta naturaleza de baja latencia y bidireccionalidad ha llevado a su adopción en aplicaciones de IoT.

- 2.Históricamente, antes de su estandarización en 2011, aplicaciones como juegos, sistemas de chat y aplicaciones web ya aprovechaban al máximo el potencial de los WebSockets. En el contexto de IoT, los WebSockets son una de las tecnologías más adoptadas para protocolos de comunicación.

En resumen, los WebSockets establecen una conexión persistente que permite un flujo de comunicación constante y simultáneo en ambas direcciones entre el cliente y el servidor, lo que se define como bidireccional y full duplex.

4.1.4. Baja Latencia

En el contexto de WebSockets, se destacan que una de sus características clave es la baja latencia:

- 1.Específicamente, los WebSockets son una tecnología avanzada que posibilita abrir una sesión de comunicación interactiva manteniendo una única conexión TCP. Permiten que los clientes reciban actualizaciones solo cuando estas ocurren, sin necesidad de que el cliente le pregunte al servidor. Esto significa que tanto el cliente como el servidor pueden enviar mensajes de forma independiente el uno del otro, una cualidad que los hace bidireccionales y full duplex.

- 2.Debido a su naturaleza de baja latencia y bidireccional, los WebSockets han sido ampliamente adoptados en aplicaciones de Internet de las Cosas (IoT). Esta característica es fundamental porque permite una comunicación más rápida y eficiente, donde las actualizaciones se entregan casi instantáneamente en el momento en que suceden, sin retrasos significativos causados por el sondeo constante al servidor.

4.1.5. Funcionamiento

Se proporcionan una descripción clara del funcionamiento de los WebSockets, especialmente en el contexto de la comunicación en el Internet de las Cosas (IoT) y aplicaciones en tiempo real:

- 1. Establecimiento de la Conexión: Inicialmente, un cliente realiza una solicitud HTTP al servidor, pidiéndole que actualice el protocolo a WebSockets.

- 2. Handshake y Persistencia: El servidor reconoce esta solicitud mediante un 'handshake' como respuesta. Una vez que este handshake es exitoso, la sesión se mantiene abierta y persistente durante todo el tiempo, a menos que alguna de las partes la cierre.

- 3. Comunicación Bidireccional y Full Duplex: Una vez establecida la conexión, los WebSockets permiten una comunicación interactiva donde tanto el cliente como el servidor pueden enviar mensajes de forma independiente el uno del otro. Esto significa que los clientes pueden recibir actualizaciones solo cuando ocurren, sin necesidad de consultar o preguntar constantemente al servidor por ellas.
- 4. Paquetes de Datos Ligeros: Durante la sesión, los paquetes de datos que se envían entre el cliente y el servidor son muy pequeños, con una longitud de trama de 2 a 14 bytes. Esta característica contribuye a su baja latencia.

Los WebSockets se estandarizaron en 2011, y su naturaleza de baja latencia y bidireccional los ha hecho muy adoptados en aplicaciones IoT, juegos, sistemas de chat y aplicaciones web. Es importante destacar que, aunque MQTT puede ejecutarse sobre la capa de WebSocket, son protocolos diferentes; los WebSockets proporcionan la 'infraestructura' de comunicación (como las carreteras), mientras que MQTT es un 'servicio de entrega' que opera sobre ella.

CLIENTE SOLICITA ACTUALIZACIóN A PROTOCOLO WEBSOCKETS (HTTP)

En el contexto del funcionamiento de los WebSockets, se explican claramente el rol del cliente en la solicitud de actualización al protocolo WebSocket a través de HTTP. El proceso se describe de la siguiente manera:

- 1.Solicitud Inicial del Cliente: Inicialmente, un cliente realiza una solicitud HTTP y le pide al servidor que actualice al protocolo WebSockets. Esta es la fase de inicio para establecer la comunicación WebSocket.
- 2.Reconocimiento del Servidor (Handshake): El servidor reconoce esta solicitud a través de un 'handshake' como respuesta.
- 3.Conexión Persistente: Una vez que el handshake es exitoso, la sesión se mantiene abierta y persistente durante todo el tiempo, a menos que alguna de las partes la cierre.

Este paso es fundamental porque, a partir de esta solicitud inicial HTTP y el subsiguiente handshake, se establece la conexión TCP única que permite la sesión de comunicación interactiva, bidireccional y full duplex de los WebSockets. Posteriormente, durante esta sesión abierta, los paquetes de datos que se envían entre el cliente y el servidor son muy pequeños, con una longitud de trama de 2 a 14 bytes, lo que contribuye a la baja latencia de los WebSockets.

SERVIDOR RECONOCE HANDSHAKE COMO RESPUESTA

En el contexto más amplio del funcionamiento de los WebSockets, se explican que el servidor reconoce el handshake como un paso crucial para establecer una comunicación persistente y bidireccional. El proceso se desarrolla de la siguiente manera:

- 1. Solicitud Inicial del Cliente: Primero, un cliente realiza una solicitud HTTP al servidor, pidiéndole que 'actualice' el protocolo a WebSockets.
- 2. Reconocimiento del Servidor (Handshake): En respuesta a esta solicitud, el servidor reconoce el 'handshake'. Este reconocimiento es la señal de que la solicitud de actualización ha sido aceptada.
- 3. Establecimiento de la Sesión Persistente: Una vez que el servidor ha reconocido exitosamente el handshake, la sesión se mantiene abierta y persistente durante todo el tiempo, a menos que alguna de las partes decida cerrarla.

Este 'handshake' inicial es fundamental porque transforma una solicitud HTTP convencional en una conexión TCP única y persistente, lo que permite la comunicación interactiva y eficiente que caracteriza a los WebSockets. Después de este establecimiento, la comunicación se vuelve bidireccional y full duplex, permitiendo que tanto el cliente como el servidor envíen mensajes de forma independiente, con paquetes de datos muy pequeños (de 2 a 14 bytes), lo que contribuye a su baja latencia.

SESIóN ABIERTA Y PERSISTENTE

En el contexto más amplio del funcionamiento de los WebSockets, se explican que la característica de una sesión abierta y persistente es un resultado directo del proceso de establecimiento de la conexión. El funcionamiento se describe de la siguiente manera:

- 1. Solicitud Inicial del Cliente: Un cliente inicia el proceso realizando una solicitud HTTP al servidor, pidiéndole que actualice el protocolo a WebSockets.
- 2. Reconocimiento del Servidor (Handshake): El servidor responde a esta solicitud reconociendo el 'handshake'.
- 3. Establecimiento de la Sesión Abierta y Persistente: Una vez que el handshake es exitoso, la sesión se mantiene abierta y persistente durante todo el tiempo, a menos que alguna de las partes la cierre.

Esta sesión persistente es fundamental para las ventajas que ofrecen los WebSockets, como la comunicación bidireccional y full duplex, donde tanto el cliente como el servidor pueden enviar mensajes de forma independiente. Además, durante esta sesión abierta, los paquetes de datos intercambiados son muy pequeños (con una longitud de trama de 2 a 14 bytes), lo que contribuye a la baja latencia de los WebSockets. La capacidad de mantener una conexión única TCP y una sesión interactiva es lo que permite a los clientes recibir actualizaciones solo cuando ocurren, sin necesidad de consultarle al servidor constantemente.

PAQUETES DE DATOS PEQUEñOS

En el contexto más amplio del funcionamiento de los WebSockets, se destacan la característica de los paquetes de datos pequeños (2 a 14 bytes) como un factor importante en su eficiencia. El funcionamiento de los WebSockets se desarrolla de la siguiente manera:

- 1. Solicitud Inicial y Handshake: El proceso comienza cuando un cliente envía una solicitud HTTP al servidor, solicitando una 'actualización' al protocolo WebSockets. El servidor, a su vez, reconoce esta solicitud mediante un 'handshake'.
- 2. Sesión Abierta y Persistente: Una vez que el handshake es exitoso, se establece una sesión que se mantiene abierta y persistente. Esta conexión TCP única permite una comunicación interactiva continua entre el cliente y el servidor.
- 3. Intercambio de Paquetes de Datos Pequeños: Durante esta sesión abierta, los paquetes de datos que se envían entre el cliente y el servidor son muy pequeños, con una longitud de trama de 2 a 14 bytes.

Esta característica de los paquetes de datos ligeros contribuye directamente a la baja latencia de los WebSockets, lo que los hace ideales para aplicaciones donde las actualizaciones deben ser casi instantáneas, como en juegos, sistemas de chat y aplicaciones IoT.

4.2. MQTT (Message Queuing Telemetry Transport)

Se proporcionan una visión detallada de MQTT (Message Queuing Telemetry Transport) en el contexto de los Protocolos de Comunicación IoT, destacando su eficiencia, modelo de funcionamiento y su relación con otras tecnologías.

Aquí se desglosa lo que se dice sobre MQTT:

- 1. Definición y Características Fundamentales
  - Protocolo de Mensajería Ligero: MQTT se describe como un 'protocolo de transferencia de mensajería ligero' (Message Queuing Telemetry Transport) diseñado específicamente para la comunicación máquina a máquina (M2M) e Internet de las Cosas.
  - Eficiencia en Ancho de Banda: Es 'muy eficiente en ancho de banda ' , utilizando solo 2 bytes de sobrecarga, lo que lo hace ideal para entornos con reproyectos limitados.
  - Comunicación en Tiempo Real y Ligera: Junto con WebSockets, MQTT es considerado uno de los 'protocolos de comunicación en tiempo real y ligeros ' más adoptados en aplicaciones IoT.

- 2. Modelo de Funcionamiento: Publicar/Suscribir
  - Diversos Escenarios de Datos: MQTT soporta escenarios de transmisión de datos uno a uno, uno a muchos y muchos a muchos para dispositivos y aplicaciones.
  - Modelo Publicar/Suscribir: Este modelo se logra a través de un esquema de 'publicar y suscribir' . Los datos se envían sobre un 'tópico ' específico, y los dispositivos suscritos a ese tópico pueden recibir los datos.
  - El Broker (Agente):
    - El 'punto central de comunicación' en MQTT es el broker .
    - El broker es el encargado de 'despachar todos los mensajes' entre el emisor y los receptores.
  - Tópicos:
    - Cada cliente que publica un mensaje al broker incluye un 'tópico ' dentro del mensaje.
    - Este tópico sirve como 'información de enrutamiento ' para el broker, permitiéndole reenviar el mensaje a los receptores suscritos a ese tópico.
    - Ejemplo: Un usuario puede publicar la temperatura en un tópico llamado 'temperatura ', y el broker reenviará estos datos a un aire acondicionado que esté suscrito al mismo tópico, para que ajuste la temperatura deseada. Otro ejemplo es un sensor de humedad que p ublica niveles de humedad en un tópico 'jardín ', y una bomba de agua se activa si el nivel es bajo.
- 3. Ventajas y Desventajas
  - Escalabilidad: El protocolo es 'altamente escalable ' porque los clientes no necesitan conocerse entre sí; solo necesitan comunicarse a través del tópico.
  - Dependencia Centralizada: El 'único inconveniente' de este protocolo es su 'entidad central' (el broker). Si el broker falla, 'toda la comunicación se perderá' .
- 4. Relación con WebSockets
  - Capas de Protocolo: MQTT y WebSockets son 'cosas diferentes ' .
  - MQTT sobre WebSockets: MQTT puede ejecutarse 'sobre la capa de WebSockets ' . Esto es posible porque MQTT, a su vez, se ejecuta sobre la red TCP/IP.
  - Analogía de 'Entrega de Servicio ': Para aclarar la relación, se usa una analogía:
    - MQTT se compara con un 'servicio de entrega ' (como DHL).
    - WebSockets proporcionan la 'infraestructura ' o las 'carreteras ' por las que viaja el servicio de entrega.
  - Empaquetamiento de Datos: Un paquete de datos MQTT se 'empaqueta dentro de un sobre de WebSocket' , que a su vez se envuelve en un 'sobre TCP/IP ' antes de ser enviado por Internet. El proceso inverso ocurre al desempaquetarse en el destino.
- 5. Rol en el Proyecto 'Internet de las Cosas con Python y Raspberry Pi '
  - Estudio en Profundidad: En la etapa 3 del proyecto, se estudia 'en profundidad' sobre los protocolos de comunicación en tiempo real y ligeros para Internet de las Cosas, incluyendo MQTT, WebSockets y una demostración práctica con PubNub. Esto subraya su importancia para los participantes del proyecto que buscan comprender y resolver problemas reales de IoT.

En resumen, MQTT es un protocolo esencial en el ecosistema IoT debido a su diseño ligero y eficiente, su modelo publicar/suscribir basado en brokers y tópicos, y su capacidad de coexistir con WebSockets para una comunicación robusta y escalable.

4.2.1. Protocolo de Mensajería Ligero

Se describen a MQTT (Message Queuing Telemetry Transport) como un protocolo ligero de transferencia de mensajes con las siguientes características en el contexto más amplio de su funcionamiento:

- Definición de Protocolo Ligero: MQTT es un protocolo ligero de transferencia de mensajes diseñado para la comunicación máquina a máquina (M2M) y el Internet de las Cosas (IoT).

- Eficiencia de Ancho de Banda: Se destaca por ser muy eficiente en el uso del ancho de banda, requiriendo solo 2 bytes de sobrecarga. Esta mínima sobrecarga es un factor clave que lo clasifica como un protocolo ligero.
- Modelo de Publicación y Suscripción: Opera mediante un modelo de publicación y suscripción. En este modelo, una transmisión de datos específica se envía sobre un 'tema' (topic), y los dispositivos que se 'suscriben' a ese tema pueden recibir los datos. Esto permite escenarios de transmisión de datos uno a uno, uno a muchos y muchos a muchos para dispositivos y aplicaciones.
- Base en TCP/IP: Este protocolo funciona sobre la red TCP/IP, lo que implica que puede ser utilizado incluso sobre la capa de WebSocket. Sin embargo, se aclaran que no se debe confundir MQTT con WebSockets, ya que son conceptos diferentes; MQTT es un 'servicio de entrega' que trabaja sobre WebSockets, que proporcionan la 'infraestructura' (como las carreteras). Un paquete de datos MQTT se encapsula dentro de un 'sobre' WebSocket, que a su vez se envuelve en un 'sobre' TCP/IP antes de ser enviado por internet.
- Broker Central: El punto central de comunicación en MQTT es el broker, que se encarga de despachar todos los mensajes entre remitentes y receptores. Cada cliente que publica un mensaje al broker incluye un tema, que sirve como información de enrutamiento para que el broker reenvíe el mensaje a los receptores suscritos a ese tema.
- Escalabilidad: Esta arquitectura lo hace altamente escalable, ya que los clientes no necesitan conocerse entre sí, solo comunicarse a través del tema.
- Desventaja (Entidad Central): La única desventaja mencionada es la dependencia de una entidad central: si el broker falla, toda la comunicación se interrumpe.

En resumen, la naturaleza de protocolo ligero de MQTT, su eficiencia en el ancho de banda con baja sobrecarga, y su modelo de publicación/suscripción lo hacen una solución muy adoptada en entornos de IoT donde los recursos y el ancho de banda pueden ser limitados.

4.2.2. Máquina a Máquina (M2M) e IoT

Se describen MQTT (Message Queuing Telemetry Transport) como un protocolo ligero de transferencia de mensajes específicamente diseñado para la comunicación máquina a máquina (M2M) y el Internet de las Cosas (IoT). Es una de las tecnologías más reconocidas y adoptadas en IoT para protocolos de comunicación. En el contexto más amplio de su funcionamiento para M2M e IoT, se destacan las siguientes características clave:

- Eficiencia de Ancho de Banda y Ligereza: MQTT es 'muy eficiente en el uso del ancho de banda'. Su diseño ligero se evidencia en que requiere solo 2 bytes de sobrecarga, lo cual es crucial para los dispositivos IoT que a menudo tienen recursos limitados y operan en entornos con restricciones de ancho de banda.
- Modelo de Publicación y Suscripción: Este protocolo opera mediante un modelo de publicación y suscripción, el cual es fundamental para los escenarios de comunicación M2M e IoT.
  - Una transmisión de datos específica se envía sobre un 'tema' (topic).
  - Los dispositivos que se 'suscriben' a ese tema pueden recibir los datos.
  - Este modelo permite escenarios de transmisión de datos uno a uno, uno a muchos y muchos a muchos entre dispositivos y aplicaciones.
  - Por ejemplo, un usuario puede publicar la temperatura en un tema llamado 'temperatura', y un aire acondicionado, suscrito al mismo tema, recibirá esos datos para ajustar la temperatura deseada. Otro ejemplo es un sensor de humedad que publica niveles de humedad en un tema llamado 'jardín', y una bomba de agua se activa si el nivel de humedad es bajo.
- Broker Central y Escalabilidad: El broker es el punto central de comunicación en MQTT y se encarga de despachar todos los mensajes entre los remitentes y los receptores. Cada cliente que publica un mensaje al broker incluye un tema, que actúa como información de enrutamiento para que el broker reenvíe el mensaje a los receptores suscritos a ese tema. Esta arquitectura hace que el protocolo sea altamente escalable, ya que los clientes no necesitan conocerse entre sí, solo comunicarse a través del tema. Esto es esencial para el crecimiento y la gestión de un gran número de dispositivos en un ecosistema IoT.

- Funcionamiento sobre TCP/IP y WebSockets: MQTT funciona sobre la red TCP/IP y puede ser utilizado sobre la capa de WebSocket. Se aclaran que MQTT y WebSockets son conceptos diferentes: MQTT se considera un 'servicio de entrega' que trabaja sobre la 'infraestructura' proporcionada por WebSockets. Un paquete de datos MQTT se encapsula dentro de un 'sobre' WebSocket, que a su vez se envuelve en un 'sobre' TCP/IP antes de ser enviado por internet.
- Desventaja: La única desventaja mencionada es su dependencia de una entidad central (el broker); si el broker falla, toda la comunicación se interrumpe.

4.2.3. Muy Eficiente en Ancho de Banda (2 Bytes de Sobre Carga)

Se destacan que MQTT (Message Queuing Telemetry Transport) es un protocolo altamente eficiente en el uso del ancho de banda, lo cual es una de sus características definitorias en el contexto más amplio de su aplicación en el Internet de las Cosas (IoT) y la comunicación máquina a máquina (M2M). Específicamente, se mencionan:

- Baja Sobrecarga (2 Bytes de Overhead): MQTT es 'muy eficiente en el uso del ancho de banda' y utiliza solo 2 bytes de sobrecarga. Esta mínima cantidad de datos adicionales es crucial y un factor clave que lo clasifica como un protocolo ligero de transferencia de mensajes.
- Idoneidad para M2M e IoT: Esta eficiencia lo hace especialmente adecuado para dispositivos IoT y escenarios M2M, donde a menudo los recursos son limitados y el ancho de banda puede ser una preocupación. Permite la transmisión de datos sin consumir excesivos recursos de red, lo que es vital para la operación de miles de dispositivos conectados.
- Funcionamiento sobre TCP/IP: A pesar de su ligereza, MQTT funciona sobre la red TCP/IP, e incluso puede ser utilizado sobre la capa de WebSocket. Un paquete de datos MQTT se encapsula dentro de un 'sobre' WebSocket, que a su vez se envuelve en un 'sobre' TCP/IP antes de ser enviado por internet, y se desempaqueta en el orden inverso al recibirlo. Esta arquitectura demuestra que puede mantener su eficiencia incluso cuando se anida dentro de otras capas de comunicación.

En resumen, la característica de ser eficiente en ancho de banda con solo 2 bytes de sobrecarga es fundamental para la identidad de MQTT como un protocolo ligero, haciendo posible su amplia adopción en el ecosistema IoT donde la optimización de recursos es primordial.

4.2.4. Escenario de Datos Uno a Uno, Uno a Muchos y Muchos a Muchos

Se explican que MQTT (Message Queuing Telemetry Transport) está diseñado para facilitar diversos escenarios de streaming de datos, especialmente en el contexto de la comunicación máquina a máquina (M2M) y el Internet de las Cosas (IoT). En el contexto más amplio de MQTT, los escenarios de streaming de datos se logran principalmente a través de su modelo de publicación y suscripción:

- Tipos de Escenarios: MQTT proporciona escenarios de streaming de datos de uno a uno, uno a muchos y muchos a muchos para dispositivos y aplicaciones.
- Modelo de Publicación y Suscripción: Esto se consigue mediante un modelo en el que una transmisión de datos específica se envía a través de un 'tema' (topic), y los dispositivos que se 'suscriben' a ese tema pueden recibir los datos.
- Rol del Broker y los Temas: El broker es el punto central de comunicación y se encarga de despachar todos los mensajes. Cada cliente que publica un mensaje al broker incluye un tema, que sirve como información de enrutamiento para que el broker reenvíe el mensaje a los receptores suscritos a ese tema.
- Se proporcionan ejemplos concretos de como se manifiestan estos escenarios de streaming de datos en la práctica:
  - Un usuario puede publicar la temperatura en un tema llamado 'temperatura'. El broker, al recibir este mensaje, reenvía los datos a un aire acondicionado que está suscrito al mismo tema, permitiendo que el aire acondicionado establezca la temperatura deseada.
  - Un sensor de humedad puede medir el nivel de humedad y publicarlo en un tema llamado 'jardín'. Una bomba de agua, que está suscrita a ese tema, se activará si el nivel de humedad detectado es bajo.

- Este diseño de publicación/suscripción, junto con el rol central del broker, hace que el protocolo sea altamente escalable para estos escenarios de streaming, ya que los clientes no necesitan conocerse entre sí, solo comunicarse a través de los temas.
- Además, la naturaleza de MQTT como un protocolo ligero de transferencia de mensajes y su eficiencia en el uso del ancho de banda (con solo 2 bytes de sobrecarga) lo hacen particularmente adecuado para estos escenarios de streaming en dispositivos IoT, donde los recursos y el ancho de banda pueden ser limitados.

Uno a Uno

Se indican que MQTT (Message Queuing Telemetry Transport) está diseñado para soportar una variedad de escenarios de streaming de datos, y explícitamente mencionan el uno a uno como una de estas capacidades. En el contexto más amplio de los escenarios de streaming de datos que ofrece MQTT, se destaca lo siguiente:

- Tipos de Escenarios: Se afirman que MQTT proporciona escenarios de streaming de datos de uno a uno, uno a muchos y muchos a muchos para dispositivos y aplicaciones. Esta declaración subraya la flexibilidad del protocolo para cubrir diversas necesidades de comunicación en el ámbito del Internet de las Cosas (IoT) y la comunicación máquina a máquina (M2M).
- Modelo de Publicación y Suscripción: La capacidad de MQTT para facilitar estos diferentes tipos de streaming, incluyendo el uno a uno, se logra a través de su modelo de publicación y suscripción. En este modelo, una transmisión de datos específica se envía sobre un 'tema' (topic), y los dispositivos que se 'suscriben' a ese tema pueden recibir los datos.
- Rol del Broker: El broker actúa como el punto central de comunicación en MQTT, encargado de despachar todos los mensajes entre remitentes y receptores. Un cliente que publica un mensaje al broker incluye un tema, que sirve como información de enrutamiento para que el broker reenvíe el mensaje a los receptores suscritos a ese tema.

Aunque no se ofrecen un ejemplo específico que ilustre de forma exclusiva un escenario 'uno a uno' donde solo un suscriptor recibe el mensaje (los ejemplos provistos, como el sensor de temperatura y el aire acondicionado, o el sensor de humedad y la bomba de agua, podrían configurarse como uno a uno si solo hay un suscriptor único para un tema específico), la mención explícita de 'uno a uno' confirma que MQTT tiene la capacidad inherente de gestionar este tipo de comunicación directa entre un publicador y un suscriptor específico, lo cual se logra mediante el uso de temas dedicados o configuraciones particulares dentro del modelo de publicación/suscripción.

Uno a Muchos

Se indican que MQTT (Message Queuing Telemetry Transport) está diseñado para soportar una variedad de escenarios de streaming de datos, y explícitamente menciona el uno a muchos como una de estas capacidades. En el contexto más amplio de los escenarios de streaming de datos que ofrece MQTT, se destaca lo siguiente:

- Tipos de Escenarios: Se afirman que MQTT proporciona escenarios de streaming de datos de uno a uno, uno a muchos y muchos a muchos para dispositivos y aplicaciones. Esta capacidad demuestra la flexibilidad del protocolo para manejar diversas configuraciones de comunicación en el Internet de las Cosas (IoT) y la comunicación máquina a máquina (M2M).
- Modelo de Publicación y Suscripción: La funcionalidad de 'uno a muchos' se logra a través del modelo de publicación y suscripción de MQTT. En este modelo, una transmisión de datos específica se envía sobre un 'tema' (topic). Múltiples dispositivos o aplicaciones pueden 'suscribirse' a ese mismo tema y, por lo tanto, recibir los datos que se publican en él.
- Rol del Broker y los Temas: El broker es el punto central de comunicación en MQTT, encargado de despachar todos los mensajes entre remitentes y receptores. Cuando un cliente publica un mensaje, incluye un tema dentro del mensaje. Este tema actúa como información de enrutamiento para que el broker reenvíe el mensaje a todos los receptores que estén suscritos a ese tema coincidente.

- Escalabilidad: Esta arquitectura hace que el protocolo sea altamente escalable para escenarios de streaming de datos como el 'uno a muchos', ya que los clientes no necesitan conocerse entre sí; solo necesitan comunicarse a través del tema. Esto es fundamental para entornos IoT donde un solo sensor podría enviar datos a múltiples aplicaciones o dispositivos que requieren esa información.

Aunque los ejemplos directos (temperatura a un aire acondicionado, sensor de humedad a una bomba de agua) se presentan de forma implícitamente 'uno a uno' en su descripción, el modelo subyacente de publicación/suscripción con el broker permite intrínsecamente que un publicador envíe datos a múltiples suscriptores simultáneamente, lo cual es la esencia del escenario 'uno a muchos' que el protocolo declara soportar.

Muchos a Muchos

Se indican que MQTT (Message Queuing Telemetry Transport) está diseñado para soportar una variedad de escenarios de streaming de datos, y explícitamente menciona el muchos a muchos como una de estas capacidades. En el contexto más amplio de los escenarios de streaming de datos que ofrece MQTT, se destaca lo siguiente:

- Tipos de Escenarios: Se afirman que MQTT proporciona escenarios de streaming de datos de uno a uno, uno a muchos y muchos a muchos para dispositivos y aplicaciones. Esta capacidad demuestra la flexibilidad del protocolo para manejar complejas configuraciones de comunicación en el Internet de las Cosas (IoT) y la comunicación máquina a máquina (M2M).
- Modelo de Publicación y Suscripción: La funcionalidad de 'muchos a muchos' se logra a través del modelo de publicación y suscripción de MQTT. En este modelo, múltiples dispositivos o aplicaciones pueden publicar transmisiones de datos sobre diferentes 'temas' (topics), y a su vez, múltiples dispositivos o aplicaciones pueden suscribirse a uno o varios de esos temas para recibir los datos.
- Rol del Broker y los Temas: El broker es el punto central de comunicación en MQTT, encargado de despachar todos los mensajes entre remitentes y receptores. Cuando un cliente publica un mensaje, incluye un tema dentro del mismo. Este tema actúa como información de enrutamiento para que el broker reenvíe el mensaje a todos los receptores que estén suscritos a ese tema coincidente.
- Escalabilidad: Esta arquitectura hace que el protocolo sea altamente escalable para escenarios de streaming de datos como el 'muchos a muchos', ya que los clientes no necesitan conocerse entre sí; solo necesitan comunicarse a través de los temas. Esto es fundamental para entornos IoT donde un gran número de dispositivos pueden estar generando datos y un gran número de aplicaciones o usuarios pueden estar interesados en diferentes flujos de datos simultáneamente.

Aunque no se proporciona un ejemplo específico que ilustre un escenario 'muchos a muchos' de manera explícita, la capacidad de múltiples publicadores para enviar a un broker y de múltiples suscriptores para recibir de ese mismo broker, organizada por temas, es la base de esta funcionalidad declarada.

4.2.5. Modelo Publicar/Suscribir

Se explican que el Modelo Publicar/Suscribir (Publish/Subscribe Model) es una característica central y fundamental del funcionamiento de MQTT (Message Queuing Telemetry Transport), especialmente en el contexto de la comunicación áquina a máquina (M2M) y el Internet de las Cosas (IoT). A continuación, se detalla lo que se dice sobre este modelo:

- Fundamento del Modelo: El modelo de publicación y suscripción es la forma en que MQTT logra la transferencia de mensajes. En este modelo, una transmisión de datos específica se envía sobre un 'tema' (topic), y los dispositivos o aplicaciones que se 'suscriben' a ese tema pueden recibir los datos.
- Roles Clave:
  - Publicador (Publisher): Es el cliente que envía mensajes. Cada cliente que publica un mensaje al broker incluye un tema dentro del mensaje.
  - Suscriptor (Subscriber): Es el cliente que recibe mensajes. Los dispositivos se suscriben a temas específicos para recibir los datos asociados a ellos.

    - Broker: Es el punto central de comunicación en MQTT. El broker es el encargado de despachar todos los mensajes entre los remitentes (publicadores) y los receptores (suscriptores). El tema incluido por el publicador sirve como información de enrutamiento para que el broker reenvíe el mensaje a todos los receptores que estén suscritos a ese tema.

- Escenarios de Streaming de Datos: Gracias a este modelo, MQTT proporciona diversos escenarios de streaming de datos, incluyendo:
    - Uno a uno.
    - Uno a muchos.
    - Muchos a muchos para dispositivos y aplicaciones.

- Ejemplos de Aplicación:
    - Un usuario puede publicar la temperatura en un tema llamado 'temperatura'. El broker, al recibir este mensaje, reenvía los datos a un aire acondicionado que está suscrito al mismo tema, lo que le permite establecer la temperatura deseada.
    - Un sensor de humedad puede medir el nivel de humedad y publicarlo en un tema llamado 'jardín'. Una bomba de agua, suscrita a ese tema, se activará si el nivel de humedad es bajo.

- Escalabilidad: El modelo de publicación y suscripción, con el broker como intermediario, hace que el protocolo MQTT sea altamente escalable. Esto se debe a que los clientes no necesitan conocerse directamente entre sí; solo tienen que comunicarse a través del tema y el broker se encarga del enrutamiento. Esta característica es fundamental para gestionar un gran número de dispositivos en un ecosistema IoT.

- Integración con WebSockets: Un paquete de datos MQTT se encapsula dentro de un 'sobre' WebSocket, que a su vez se envuelve en un 'sobre' TCP/IP antes de ser enviado por internet. Esto demuestra cómo MQTT, con su modelo publicar/suscribir, puede operar sobre diferentes capas de red manteniendo su funcionalidad.

En resumen, el Modelo Publicar/Suscribir es la columna vertebral de MQTT, permitiendo una comunicación eficiente, flexible y escalable en entornos M2M e IoT al desacoplar a los publicadores de los suscriptores mediante el uso de temas y un broker centralizado

subsubsectionStream de Datos Enviado sobre un 'Tópico' Se explican que el concepto de Stream de Datos Enviado sobre un 'Tópico' (Topic) es fundamental para el Modelo Publicar/Suscribir de MQTT (Message Queuing Telemetry Transport), especialmente en la comunicación máquina a máquina (M2M) y el Internet de las Cosas (IoT). En el contexto más amplio del Modelo Publicar/Suscribir:

- MQTT como Protocolo de Transferencia de Mensajes Ligero:
    - MQTT (Message Queuing Telemetry Transport) es un protocolo de transferencia de mensajes ligero diseñado para la comunicación de máquina a máquina (M2M) y el Internet de las Cosas (IoT).
    - Es muy eficiente en cuanto al ancho de banda, utilizando solo 2 bytes de sobrecarga.
    - Funciona sobre una red TCP/IP, lo que significa que también puede usarse sobre una capa WebSocket.

- El Modelo Publicar/Suscribir (Publish/Subscribe):
    - MQTT se basa en un modelo Publicar/Suscribir para escenarios de transmisión de datos uno a uno, uno a muchos y muchos a muchos, lo que lo hace ideal para dispositivos y aplicaciones IoT.
    - En este modelo, una corriente de datos específica se envía sobre un 'tópico' determinado.
    - Los dispositivos o aplicaciones que se 'suscriben' a ese tópico pueden recibir los datos.

- El Rol del Broker y los Tópicos:
    - El punto central de comunicación en MQTT es el broker.
    - El broker es el responsable de despachar todos los mensajes entre el emisor (publisher) y los receptores (subscribers).

    - Cada cliente que publica un mensaje al broker incluye un tópico dentro del mensaje.
    - Este tópico actúa como información de enrutamiento para el broker, permitiéndole reenviar el mensaje a los receptores que están suscritos a ese tópico específico.
- Flujo de Comunicación y Escalabilidad:
    - Los clientes (publishers) no necesitan conocerse entre sí; solo necesitan comunicarse a través del tópico, lo que hace que el protocolo sea altamente escalable.
    - Un ejemplo dado es un usuario publicando la temperatura en un tópico llamado 'temperatura', y el broker reenvía estos datos a un aire acondicionado suscrito al mismo tópico para ajustar la temperatura deseada.
    - Otro ejemplo menciona un sensor de humedad midiendo el nivel de humedad y publicando en el tópico 'jardín', lo que hace que la bomba de agua se encienda si el nivel de humedad es bajo.
- Distinción con WebSockets:
    - Es importante no confundir MQTT con WebSockets, ya que son cosas diferentes.
    - Se puede considerar a MQTT como un 'servicio de entrega' que funciona sobre WebSockets, de la misma manera que DHL utiliza las carreteras y vías proporcionadas por WebSockets.
    - Un paquete de datos MQTT se 'empaqueta' dentro de un 'sobre' de WebSocket, que a su vez se 'envuelve' en un sobre TCP/IP y se envía por Internet, desempaquetándose en orden inverso al recibirse.
- Inconvenientes:
    - La principal desventaja de este protocolo es la entidad central (el broker); si el broker falla, toda la comunicación se interrumpe.

En resumen, se explican que en el modelo Publicar/Suscribir de MQTT, los flujos de datos se asocian a 'tópicos' específicos. Un broker central distribuye estos mensajes, enrutándolos a los suscriptores basándose en los tópicos a los que están suscritos. Esto permite una comunicación eficiente y escalable para aplicaciones IoT.

Dispositivos se Suscriben a Tópicos para Recibir Datos

Aquí se ofrecen una explicación detallada sobre cómo los dispositivos se suscriben a tópicos para recibir datos en el contexto del modelo Publicar/Suscribir, específicamente a través del protocolo MQTT. Aquí se desglosa este aspecto:

- El Modelo Publicar/Suscribir en MQTT:
    - MQTT (Message Queuing Telemetry Transport) es un protocolo de transferencia de mensajes ligero diseñado para la comunicación máquina a máquina (M2M) e Internet de las Cosas (IoT).
    - Se basa en un modelo Publicar/Suscribir que permite escenarios de transmisión de datos uno a uno, uno a muchos y muchos a muchos, lo que lo hace ideal para dispositivos y aplicaciones IoT.
    - En este modelo, una corriente de datos específica se envía sobre un 'tópico' determinado, y los dispositivos que se 'suscriben' a ese tópico pueden recibir los datos.
- El Rol del Broker y los Tópicos como Enrutamiento:
    - El punto central de comunicación en MQTT es el broker.
    - El broker es el encargado de despachar todos los mensajes entre el emisor (publisher) y los receptores (subscribers).
    - Cada cliente que publica un mensaje al broker incluye un tópico dentro del mensaje.
    - Este tópico sirve como información de enrutamiento para el broker, permitiéndole reenviar el mensaje a los receptores que están suscritos a ese tópico específico.
- Proceso de Suscripción y Recepción de Datos:

- Los clientes (dispositivos o aplicaciones) no necesitan conocerse entre sí; solo necesitan comunicarse a través del tópico.
- Cuando un dispositivo se 'suscribe' a un tópico, le está indicando al broker que desea recibir todos los mensajes publicados en ese tópico.
- El broker, al recibir un mensaje con un tópico específico de un publisher, busca a todos los suscriptores de ese mismo tópico y les entrega el mensaje.
- Esta arquitectura hace que el protocolo sea altamente escalable.

- Ejemplos de Suscripción de Dispositivos:
  - Un ejemplo proporcionado es un usuario que publica la temperatura en un tópico llamado 'temperatura'; el broker reenvía estos datos a un aire acondicionado que está suscrito al mismo tópico, permitiéndole establecer la temperatura deseada.
  - Otro ejemplo involucra un sensor de humedad que mide el nivel de humedad y publica en el tópico 'jardín'. Si el nivel de humedad es bajo, la bomba de agua, que está suscrita a ese tópico, se enciende.

- Funcionamiento sobre Redes y Comparación con WebSockets:
  - MQTT funciona sobre una red TCP/IP, lo que significa que también puede usarse sobre una capa WebSocket.
  - Se aclaran que MQTT y WebSockets son distintos; se puede pensar en MQTT como un 'servicio de entrega' que opera sobre WebSockets. Un paquete de datos MQTT se 'empaqueta' dentro de un 'sobre' de WebSocket, que a su vez se 'envuelve' en un sobre TCP/IP y se envía por Internet.

- Consideraciones:
  - La principal desventaja de este protocolo es la entidad central (el broker); si el broker falla, toda la comunicación se interrumpe.

En conclusión, se enfatizan que los dispositivos se suscriben a tópicos en el modelo Publicar/Suscribir de MQTT para recibir datos relevantes. El broker actúa como un intermediario inteligente, utilizando los tópicos como mecanismo de enrutamiento para asegurar que los mensajes lleguen a los dispositivos suscritos, lo que permite una comunicación eficiente y escalable en entornos IoT.

4.2.6. Corre Sobre TCP/IP y Puede Correr Sobre Websockets

Aquí se explica claramente que MQTT (Message Queuing Telemetry Transport) funciona sobre TCP/IP y puede utilizarse sobre una capa de WebSockets, estableciendo una distinción y una relación de capas entre estos protocolos. Aquí se detalla este aspecto:

- MQTT Corre sobre TCP/IP:
  - MQTT es un protocolo ligero de transferencia de mensajes diseñado para la comunicación máquina a máquina (M2M) y el Internet de las Cosas (IoT).
  - Una característica fundamental es que este protocolo 'corre sobre la red TCP/IP'.
  - Es un protocolo muy eficiente en cuanto al ancho de banda, utilizando solo 2 bytes de sobrecarga.

- Capacidad de Uso sobre WebSockets:
  - Debido a que MQTT opera sobre TCP/IP, se indican que 'podemos usar este protocolo sobre la capa WebSocket también'.
  - Esto sugiere una flexibilidad en la implementación de MQTT, permitiéndole aprovechar las capacidades de WebSockets.

- Distinción y Relación entre MQTT y WebSockets:
  - Es crucial no confundir MQTT con WebSockets, ya que 'ambos son cosas diferentes'.

- Se ofrece una analogía para clarificar su relación: se puede 'considerar a MQTT como un servicio de entrega que funciona sobre WebSockets'. En esta analogía, MQTT sería como 'DHL', mientras que 'las carreteras y las vías son proporcionadas por el WebSocket'.
- Esta analogía ilustra que WebSocket actúa como una capa de transporte subyacente para MQTT.
- El proceso de encapsulación se describe de la siguiente manera: un 'paquete de datos MQTT se empaqueta dentro de un sobre de WebSocket, el cual es luego envuelto dentro de un sobre TCP/IP y enviado a través de Internet'. El desempaquetado ocurre en orden inverso al recibirse.

- Contexto de WebSockets:
  - Las WebSockets, por sí mismas, son una tecnología avanzada que permite abrir una sesión de comunicación interactiva persistente sobre una única conexión TCP.
  - Habilitan la comunicación bidireccional y dúplex completo, lo que significa que ambas partes (cliente y servidor) pueden enviar mensajes de forma independiente.
  - Debido a su baja latencia y naturaleza bidireccional, WebSockets son 'altamente adoptados en aplicaciones IoT'.

En resumen, se establece que MQTT está inherentemente ligado a la capa de transporte TCP/IP y que esta base permite que se utilice eficientemente 'sobre la capa WebSocket'. Aunque son protocolos distintos, WebSockets proporcionan una infraestructura (como 'carreteras') sobre la cual los paquetes de datos de MQTT pueden ser transportados, aprovechando así la eficiencia y las capacidades de comunicación bidireccional en tiempo real para las aplicaciones IoT.

4.2.7. No Confundir con Websockets (MQTT es un 'Servicio de Entrega' que Corre Encima de Websockets)

Se enfatiza la importancia de no confundir MQTT con WebSockets, a pesar de que MQTT puede operar sobre una capa de WebSockets. Se explica esta relación y distinción de la siguiente manera:

- Distinción Explícita y Analógica:
  - Se establece claramente que 'no confunda MQTT con WebSockets, ambos son cosas diferentes'.
  - Para ilustrar esta diferencia y su relación, se propone una analogía: 'considere MQTT como un servicio de entrega que funciona sobre WebSockets'.
  - En esta analogía, MQTT sería como 'DHL', y 'las carreteras y las vías son proporcionadas por el WebSocket'. Esto significa que WebSockets proporcionan la infraestructura o el medio de transporte subyacente, mientras que MQTT es el protocolo que define cómo se entregan los mensajes específicos.

- Funcionamiento y Encapsulación:
  - MQTT es un protocolo ligero de transferencia de mensajes diseñado para la comunicación máquina a máquina (M2M) y el Internet de las Cosas (IoT). Es muy eficiente en cuanto al ancho de banda, utilizando solo 2 bytes de sobrecarga.
  - Se especifica que MQTT 'corre sobre la red TCP/IP'.
  - Debido a esta base sobre TCP/IP, se indican que es posible 'usar este protocolo sobre la capa WebSocket también'.
  - El proceso técnico de cómo MQTT utiliza WebSockets se describe como una encapsulación: un 'paquete de datos MQTT se empaqueta dentro de un sobre de WebSocket, el cual es luego envuelto dentro de un sobre TCP/IP y enviado a través de Internet'. Al recibirse, el paquete se desempaqueta en el orden inverso.

- Características de WebSockets como Capa Subyacente:
  - Las WebSockets son una tecnología avanzada que permite establecer una sesión de comunicación interactiva persistente y bidireccional sobre una única conexión TCP.
  - Permiten que los clientes reciban actualizaciones solo cuando ocurren, sin necesidad de que el cliente esté 'preguntando' constantemente al servidor (polling).

- Debido a su baja latencia y naturaleza bidireccional y dúplex completo (donde ambas partes pueden enviar mensajes independientemente), las WebSockets son 'altamente adoptadas en aplicaciones IoT'.

En resumen, se recalcan que MQTT y WebSockets son protocolos distintos. Si bien MQTT es el protocolo de mensajería ligero ideal para IoT con su modelo Publicar/Suscribir, WebSockets actúan como una capa de transporte eficiente que permite a MQTT aprovechar sus capacidades de comunicación bidireccional y de baja latencia sobre una conexión TCP/IP, funcionando como un 'servicio de entrega' que utiliza las 'carreteras' que WebSockets proporcionan.

4.2.8. Funcionamiento

Aquí se ofrece una descripción detallada del funcionamiento de MQTT (Message Queuing Telemetry Transport), destacando su arquitectura, modelo de comunicación y cómo interactúa con otras tecnologías de red. Aquí se explica cómo funciona MQTT:

- 1. Naturaleza y Propósito de MQTT:
  - MQTT es un protocolo de transferencia de mensajes ligero diseñado específicamente para la comunicación de máquina a máquina (M2M) y para el Internet de las Cosas (IoT).
  - Es muy eficiente en cuanto al ancho de banda, utilizando solo 2 bytes de sobrecarga.
  - Proporciona escenarios de transmisión de datos uno a uno, uno a muchos y muchos a muchos para dispositivos y aplicaciones.
- 2. El Modelo Publicar/Suscribir (Publish/Subscribe):
  - El funcionamiento central de MQTT se basa en un modelo de publicar y suscribir.
  - En este modelo, una corriente de datos específica se envía sobre un 'tópico' determinado.
  - Los dispositivos o aplicaciones que se 'suscriben' a ese tópico pueden recibir los datos.
- 3. El Rol Central del Broker:
  - El punto central de comunicación en MQTT es el 'broker'.
  - El broker es el encargado de despachar todos los mensajes entre el emisor (publisher) y los receptores (subscribers).
  - Cada cliente que publica un mensaje al broker incluye un tópico dentro del mensaje.
  - Este tópico actúa como información de enrutamiento para el broker, permitiéndole reenviar el mensaje a los receptores que están suscritos a ese tópico específico.
- 4. Proceso de Publicación y Suscripción:
  - Cuando un 'publisher' (dispositivo o aplicación) tiene datos para enviar, los publica en un tópico específico al broker.
  - Simultáneamente, los 'subscribers' (otros dispositivos o aplicaciones) que desean recibir ciertos datos, se 'suscriben' a tópicos específicos con el broker.
  - Cuando el broker recibe un mensaje de un publisher para un tópico determinado, lo reenvía a todos los suscriptores que se han registrado para ese mismo tópico.
  - Un ejemplo dado es un usuario que publica la temperatura en un tópico llamado 'temperatura', y el broker reenvía estos datos a un aire acondicionado suscrito al mismo tópico para ajustar la temperatura deseada. Otro ejemplo es un sensor de humedad que publica en el tópico 'jardín', haciendo que una bomba de agua suscrita se encienda si el nivel es bajo.
- 5. Ventajas y Desventajas:
- Este modelo hace que el protocolo sea altamente escalable, ya que los clientes (publishers y subscribers) no necesitan conocerse entre sí; solo tienen que comunicarse a través del tópico con el broker.
- La principal desventaja de este protocolo es la entidad central (el broker); si el broker falla, toda la comunicación se interrumpe.

- 6. Funcionamiento sobre Redes (TCP/IP y WebSockets):
  - MQTT 'corre sobre la red TCP/IP'.
  - Esto significa que también puede 'usarse este protocolo sobre la capa WebSocket'.
  - Se enfatiza que MQTT y WebSockets son 'cosas diferentes'. Se utiliza la analogía de que MQTT es como un 'servicio de entrega' (DHL) que 'funciona sobre WebSockets', siendo WebSockets las 'carreteras y las vías'.
  - Técnicamente, un paquete de datos MQTT se 'empaqueta dentro de un sobre de WebSocket', el cual a su vez es 'envuelto dentro de un sobre TCP/IP' y enviado a través de Internet. El desempaquetado ocurre en orden inverso al recibirse.
  - WebSockets, por su parte, permiten una comunicación interactiva, bidireccional y dúplex completo, lo que las hace adecuadas para aplicaciones IoT debido a su baja latencia

En resumen, el funcionamiento de MQTT se centra en un broker que orquesta la comunicación entre publishers y subscribers a través de tópicos. Este modelo publicar/suscribir es altamente escalable y eficiente, operando sobre TCP/IP y pudiendo aprovechar las capacidades de WebSockets como una capa de transporte subyacente.

4.2.9. Escalabilidad: Clientes no Necesitan Conocerse Entre SÍ, Solo Comunicarse a Través del Tópico

Aquí se resalta que el diseño de MQTT (Message Queuing Telemetry Transport) y su modelo Publicar/Suscribir contribuyen significativamente a su escalabilidad, particularmente porque los clientes no necesitan conocerse entre sí. A continuación, se detalla este aspecto:

- Modelo Publicar/Suscribir como Base:
  - MQTT es un protocolo ligero de transferencia de mensajes diseñado para la comunicación de máquina a máquina (M2M) y el Internet de las Cosas (IoT).
  - Este protocolo proporciona escenarios de transmisión de datos uno a uno, uno a muchos y muchos a muchos, lográndolo a través de un modelo de publicar y suscribir.
  - En este modelo, una corriente de datos específica se envía sobre un 'tópico' determinado, y los dispositivos que se 'suscriben' a ese tópico pueden recibir los datos.
- El Broker como Intermediario Central:
  - El punto central de comunicación en MQTT es el broker.
  - El broker es el encargado de despachar todos los mensajes entre el emisor (publisher) y los receptores (subscribers).
  - Cada cliente que publica un mensaje al broker incluye un tópico dentro del mensaje, el cual sirve como información de enrutamiento para el broker. Esto permite al broker reenviar el mensaje a los receptores que están suscritos a ese tópico específico.
- La Clave de la Escalabilidad: Desconocimiento Mutuo de los Clientes:
  - La arquitectura descrita hace que el protocolo sea 'altamente escalable'.
  - La razón principal de esta escalabilidad es que 'los clientes no tienen que conocerse entre sí'.
  - En cambio, los clientes 'solo tienen que comunicarse sobre el tópico' con el broker.
- Ejemplos Ilustrativos:
  - Un usuario puede publicar la temperatura en un tópico llamado 'temperatura', y el broker reenvía estos datos a un aire acondicionado que está suscrito al mismo tópico. El aire acondicionado no necesita saber quién publicó el mensaje; solo reacciona a los datos del tópico 'temperatura'.
  - Otro ejemplo es un sensor de humedad que mide el nivel y publica en el tópico 'jardín'. Una bomba de agua, suscrita a ese tópico, se enciende si el nivel es bajo, sin tener conocimiento directo del sensor.

- Advertencia sobre la Entidad Central:
    - A pesar de la alta escalabilidad que ofrece el modelo de desconocimiento mutuo entre clientes, la principal desventaja de este protocolo es la entidad central (el broker). Si el broker falla, toda la comunicación se interrumpe.

En resumen, se enfatizan que la escalabilidad de MQTT radica en su modelo Publicar/Suscribir, donde el broker central actúa como un intermediario que enruta los mensajes basados en tópicos, eliminando la necesidad de que los clientes (publishers y subscribers) tengan conocimiento directo el uno del otro. Esta desvinculación hace que el sistema sea fácil de expandir y gestionar, aunque la dependencia de un broker centralizado representa una vulnerabilidad potencial.

4.2.10. Inconveniente: Entidad Central (BROKER), Si Flla, Toda la Comunicación se Pierde

Aquí se identifica un inconveniente clave en el funcionamiento de MQTT (Message Queuing Telemetry Transport), que se relaciona directamente con la entidad central, el BROKER. Aquí se detalla este aspecto:

- El Broker como Punto Central de Comunicación:
    - En el modelo Publicar/Suscribir de MQTT, el broker es el punto central de comunicación.
    - Es el encargado de despachar todos los mensajes entre el emisor (publisher) y los receptores (subscribers).
    - El broker utiliza los 'tópicos' incluidos en los mensajes para enrutar la información a los suscriptores correspondientes.
- La Desventaja de la Entidad Centralizada:
    - A pesar de las ventajas de escalabilidad y eficiencia que ofrece MQTT, se señalan una desventaja importante: 'el único inconveniente de este protocolo es la entidad central; si el broker muere, toda la comunicación se va a caer'.

En resumen, se explican que si bien el broker es fundamental para el funcionamiento y la escalabilidad de MQTT al facilitar la comunicación entre publishers y subscribers, su naturaleza centralizada lo convierte en un punto único de fallo. Si este broker deja de funcionar, toda la red de comunicación de los dispositivos y aplicaciones que dependen de él se interrumpirá.

4.2.11. Ejemplos

Se ofrecen varios ejemplos concretos del funcionamiento de MQTT (Message Queuing Telemetry Transport) en el contexto de sus características principales, como el modelo Publicar/Suscribir y su aplicación en IoT:

- 1. Control de Aire Acondicionado por Temperatura:
    - Escenario: Un usuario desea controlar un aire acondicionado.
    - Funcionamiento con MQTT: El usuario puede publicar la temperatura deseada en un tópico específico, por ejemplo, 'temperatura'. El broker MQTT recibe este mensaje y lo reenvía al aire acondicionado que está suscrito a ese mismo tópico. De esta manera, el aire acondicionado recibe la instrucción y ajusta la temperatura deseada.
- 2. Activación de Bomba de Agua por Nivel de Humedad:
    - Escenario: Mantener el nivel de humedad adecuado en un jardín.
    - Funcionamiento con MQTT: Un sensor de humedad mide constantemente el nivel de humedad y publica estos datos en un tópico como 'jardín'. Si el nivel de humedad es bajo (detectado por el sensor), la bomba de agua, que está suscrita a ese mismo tópico, recibe el mensaje y se enciende automáticamente para regar el jardín.

Estos ejemplos ilustran cómo MQTT, a través de su modelo Publicar/Suscribir y el uso de tópicos enrutados por un broker central, facilita la comunicación eficiente y en tiempo real entre dispositivos en un entorno IoT, permitiendo la automatización y el control de manera escalable, ya que los dispositivos no necesitan conocerse directamente entre sí.

Usuario Publica Temperatura->Aire Acondicionado se Suscribe

Se ofrece el ejemplo específico de un usuario publicando la temperatura y un aire acondicionado suscribiéndose para ilustrar el funcionamiento de MQTT (Message Queuing Telemetry Transport) en el contexto de su modelo Publicar/Suscribir y sus aplicaciones en el Internet de las Cosas (IoT). A continuación, se detalla lo que se dice sobre este ejemplo:

- El Escenario del Ejemplo:
  - Se describe directamente este caso: 'un usuario puede publicar la temperatura en un tópico llamado 'temperatura' y el broker reenvía estos datos a un aire acondicionado que está suscrito en el mismo tópico y establece la temperatura deseada'.
- Conceptos de MQTT Ilustrados:
  - Publicador (Publisher): En este ejemplo, el usuario es el publicador. El usuario (o un dispositivo en su nombre, como un termostato inteligente) genera el dato de la temperatura y lo envía al sistema.
  - Tópico (Topic): La temperatura se publica en un tópico específico, nombrado 'temperatura'. Este tópico actúa como una categoría o dirección lógica que clasifica el tipo de datos que se están enviando.
  - Broker: El broker MQTT es la entidad central que recibe el mensaje del usuario. Su función es 'reenviar estos datos' a los suscriptores apropiados.
  - Suscriptor (Subscriber): El aire acondicionado es el suscriptor. Este dispositivo ha manifestado su interés en recibir mensajes relacionados con el tópico 'temperatura', por lo que el broker le entrega los datos.
  - Acción Basada en Datos: Una vez que el aire acondicionado recibe los datos de temperatura del tópico, puede 'establecer la temperatura deseada', lo que demuestra la capacidad de los dispositivos IoT para reaccionar y actuar en función de la información recibida en tiempo real.
- Implicaciones para la Escalabilidad y la Comunicación:
  - Este ejemplo también subraya la alta escalabilidad del protocolo, ya que el usuario (publisher) y el aire acondicionado (subscriber) 'no tienen que conocerse entre sí'. Su única interacción es a través del tópico con el broker, lo que simplifica la adición o eliminación de dispositivos sin afectar a otros componentes del sistema.
  - Esto es un claro ejemplo de comunicación uno a muchos o uno a uno para el control de dispositivos en entornos IoT.

En resumen, el ejemplo del usuario publicando la temperatura y el aire acondicionado suscribiéndose ilustra de manera efectiva cómo MQTT facilita la comunicación eficiente y desvinculada entre usuarios/dispositivos y actuadores en un sistema IoT, utilizando el modelo Publicar/Suscribir, tópicos y un broker central para el enrutamiento de mensajes.

Capítulo 5

SERVIDOR IoT SEGURO Y LOGIN DE USUARIO

En el contexto más amplio del proyecto 'Internet de las Cosas con Python y Raspberry Pi ' , el Servidor IoT Seguro y el Login de Usuario constituyen la etapa 5, donde se abordan aspectos cruciales para construir una plataforma IoT robusta, segura y multiusuario. El objetivo general es crear una plataforma en la nube sin satélites donde múltiples usuarios puedan iniciar sesión de forma segura, y controlar y monitorear sus dispositivos autorizados en tiempo real.

A continuación se detallan los siguientes aspectos sobre la seguridad del servidor y la gestión de usuarios:

- 1. Asegurar el Dominio Personalizado con Certificados SSL/TLS (HTTPS)
- La primera etapa para un servidor IoT seguro es asegurar la comunicación, lo cual se logra mediante certificados SSL/TLS de Let's Encrypt.
  - Autoridad de Certificación: Let's Encrypt es una autoridad de certificación gratuita, automatizada y de código abierto, respaldada por importantes patrocinadores y ampliamente utilizada por desarrolladores y empresas.
  - Proceso de Instalación y Configuración:
  - **Software de Terceros**: El primer paso es instalar el software de terceros 'certbot' en el servidor. Esto implica agregar el repositorio PPA para 'certbot' y 'apache' (`sudo add-apt-repository ppa:certbot/cer` actualizar la lista de paquetes (`sudo apt-get update`) e instalar 'python-certbot-apache'.
  - Generación del Certificado: Se ejecuta `sudo certbot --apache -d [nombre-de-dominio] -d www.[nombre-d` para configurar el certificado SSL para Apache. Durante este proceso, se solicita un correo electrónico para la recuperación de la clave.
  - Redirección a HTTPS: Es fundamental redirigir el tráfico HTTP a HTTPS , lo cual se selecciona durante la configuración 'Type 2 to enter').
  - Ubicación de Certificados: Los archivos de certificado generados se encuentran en '/etc/letsencrypt/live/'.
  - Reglas de Seguridad Entrantes: Un paso crítico, y un error común, es olvidar asignar las reglas de seguridad entrantes para HTTPS (puerto 443) en el servidor remoto (por ejemplo, en AWS EC2). Sin esto, la conexión al servidor puede fallar. Se debe añadir una regla para HTTPS con el puerto 443 en la configuración de las instancias EC2.
  - Verificación: Se puede verificar el estado del certificado SSL en 'SSL labs.com/ssltest'.
  - Validez y Renovación: Los certificados de Let's Encrypt son válidos por aproximadamente tres meses y deben renovarse al expirar.
  - Resultado: Una vez configurado correctamente, el servidor redirigirá de HTTP a HTTPS, mostrando un candado verde en el navegador, lo que indica una comunicación cifrada de extremo a extremo para cada cliente conectado al servidor.
- 2. Funcionalidad de Login de Usuario y Gestión de Acceso

- El proyecto también aborda la implementación de una funcionalidad de login de usuario segura y el almacenamiento de los detalles del usuario en una base de datos integrada. Además, se crean reglas para usuarios administradores y no administradores para gestionar el acceso a los dispositivos y funcionalidades.
    - Roles de Usuario (Admin y No-Admin): Se establece un sistema donde los usuarios pueden tener roles de administrador o no administrador.
    - Panel de Control del Administrador:
        - Los usuarios administradores tienen acceso a un panel de control que muestra una lista de todos los usuarios en línea .
        - Frente al nombre de cada usuario en línea, hay botones de conmutación para otorgar permisos de lectura y escritura .
        - También hay un botón 'Apply ' (aplicar) para guardar los cambios en los permisos.
    - Implementación en el Servidor (Python/Flask):
        - Para poblar la lista de usuarios en el dashboard, el servidor envía detalles adicionales a la página web principal, como el `user_ID` de la sesión y una lista de `online_user_records`.
        - `online_user_records` es un mapa que contiene el nombre del usuario, el ID del usuario, y el estado de acceso de lectura y escritura (1 para 'checked'/marcado, 0 para 'unchecked '/desmarcado).
        - La función 'get-all-logged-in-users' se encarga de retornar este mapa.
    - Implementación en el Cliente (HTML/Jinja2):
        - La página 'index.html' utiliza plantillas Jinja para iterar sobre la lista de `online_user_records` y crear filas en una tabla para cada usuario. Cada fila muestra el nombre del usuario, y los ID y estados (checked/unchecked) de los botones de conmutación para lectura y escritura.
    - Visibilidad Condicional: El panel de control completo (para gestionar permisos) solo es visible para los usuarios administradores. Esto se logra mediante una declaración 'if' en el código HTML que compara el `user_ID` del usuario actual con el `user_ID` del administrador (previamente codificado o verificado). Un usuario no administrador (como 'Anam Chaudhary ' en el ejemplo) no tendrá acceso a este panel de control.
    - Otorgar Permisos en Tiempo Real:
        - Lado del Cliente (JavaScript): Un método en 'main.js' escucha los eventos de los botones de conmutación. Cuando se activa un botón de acceso, extrae el ID del usuario, el estado de lectura y el estado de escritura. Luego, envía una solicitud POST al servidor (por ejemplo, `grant_user_ID_read_state_write_state`).
        - Lado del Servidor (Aplicación Flask): El servidor tiene un endpoint ('/grant') para recibir estas solicitudes. Primero, verifica si la solicitud proviene de un usuario administrador. Si no, deniega el acceso. Si es un administrador, almacena los permisos de lectura y escritura del usuario en la base de datos y llama al servidor PubNub para otorgar estos permisos al usuario específico.
        - PubNub Access Manager: Se utiliza la funcionalidad PubNub Access Manager para que los usuarios administradores puedan otorgar permisos de lectura y escritura en tiempo real a usuarios no administradores y dispositivos. El primer paso es generar una clave de autorización para el usuario y almacenarla en la base de datos, seguido de la concesión de permisos.

Este enfoque integral en la seguridad del servidor y la gestión de usuarios sienta las bases para una plataforma IoT robusta, capaz de manejar múltiples dispositivos y usuarios de manera controlada y protegida.

5.0.1. Asegurar Dominio Personalizado con SSL/TLS

Se detallan exhaustivamente el proceso de asegurar un dominio personalizado con SSL/TLS en el contexto de la creación de un Servidor IoT Seguro y Login de Usuario , un componente crucial para establecer una plataforma IoT robusta y fiable. Este es el enfoque principal de la etapa 5 del proyecto.

Aquí se desglosa la información relevante:

- 1. Propósito y Contexto General del Curso
  - El proyecto 'Internet de las Cosas con Python y Raspberry Pi ' se centra en construir una plataforma IoT basada en la nube donde múltiples usuarios pueden iniciar sesión de forma segura y controlar y monitorear sus dispositivos autorizados en tiempo real.
  - La etapa 5 está dedicada al desarrollo de terminologías de seguridad , incluyendo la obtención de un nombre de dominio personalizado y su aseguramiento con la autoridad de certificación Let's Encrypt. También se implementará una funcionalidad de login de usuario seguro y el almacenamiento de los detalles del usuario en una base de datos integrada.
  - El objetivo final es asegurar que cada cliente conectado al servidor tenga una comunicación cifrada de extremo a extremo .
- 2. Uso de Let's Encrypt para Certificados SSL/TLS
  - El primer paso para asegurar el sitio web es utilizar la autoridad de certificación Let's Encrypt .
  - Características de Let's Encrypt: Es una autoridad de certificación gratuita, automatizada y de código abierto . Es ampliamente utilizada por desarrolladores y empresas, y cuenta con el respaldo de importantes patrocinadores.
  - Período de Validez y Renovación: Los certificados de Let's Encrypt son válidos por aproximadamente tres meses y requieren ser renovados después de su vencimiento.
- 3. Proceso de Aseguramiento del Dominio
- El proceso implica una serie de pasos técnicos detallados:
  - Instalación de Certbot:
    - El primer paso es instalar un software de terceros (Certbot) en el servidor.
    - Esto se hace agregando el repositorio de Certbot (PPA) y luego actualizando la lista de paquetes.
    - Finalmente, se instala Certbot para Apache mediante el comando `sudo apt-get install python-certbot`
  - Configuración del Certificado SSL para Apache:
    - Una vez instalado Certbot, se ejecuta el comando `sudo certbot --apache -D [nombre-de-dominio] -D` para configurar el certificado SSL.
    - Se solicitará al usuario que proporcione una dirección de correo electrónico para la recuperación de la clave en caso de pérdida.
    - Se ofrecerá la opción de redirigir el tráfico HTTP a HTTPS , lo cual se recomienda aceptar.
    - Los archivos de certificado generados se pueden encontrar en `/etc/letsencrypt/live`.
  - Configuración de Reglas de Seguridad Inbound (AWS):
    - Es crucial asegurarse de que las reglas de seguridad inbound para HTTPS (puerto 443) estén asignadas en el servidor remoto, especialmente si se utiliza AWS EC2. La omisión de este paso puede causar fallos en la conexión al servidor durante la verificación del certificado.
    - Para corregirlo, se deben editar las reglas de entrada y agregar la regla HTTPS .
- 4. Verificación y Resultado
  - Verificación del Estado del Certificado: El estado del certificado SSL se puede verificar utilizando herramientas como SSL Labs (ssl.com/ssltest).
  - Confirmación de Seguridad: Una vez completado el proceso, el certificado aparecerá como válido, firmado por Let's Encrypt para el servidor (ej. 'pact IOT server ').
  - Redirección y Comunicación Segura: El sitio web se redirigirá automáticamente de HTTP a HTTPS , mostrando un 'candado verde ' en el navegador, lo que indica una comunicación segura . Esto garantiza una comunicación cifrada de extremo a extremo para todos los clientes conectados al servidor.

En resumen, la seguridad del dominio personalizado con SSL/TLS utilizando Let's Encrypt es un pilar fundamental en la creación de un servidor IoT seguro , facilitando una comunicación cifrada y protegiendo el login de usuario y los datos intercambiados entre los dispositivos IoT y la plataforma en la nube.

5.0.2. Autoridad de Certificación: Let's Encrypt

Se detalla el papel crucial de la Autoridad de Certificación Let's Encrypt en el contexto más amplio de asegurar un dominio personalizado con SSL/TLS. Esto es este aspecto:

- ¿Qué es Let's Encrypt?
  - Let's Encrypt es una autoridad de certificación gratuita, automatizada y de código abierto.
  - Es ampliamente utilizada por muchos desarrolladores y empresas, y está respaldada por importantes patrocinadores.
  - Su objetivo principal es permitir la obtención de certificados SSL/TLS (Secure Sockets Layer/Transport Layer Security).
- Proceso de Uso para Asegurar un Dominio Personalizado:
  - 1. Instalación de certbot: El primer paso para usar Let's Encrypt es instalar un software de terceros llamado certbot en el servidor.
    - Esto implica añadir el repositorio de certbot (PPA para paquetes preparados por el equipo de Let's Encrypt de Debian y respaldados para Ubuntu).
    - Luego, se actualiza la lista de paquetes para incorporar el nuevo repositorio.
    - Finalmente, se instala certbot específicamente para Apache, utilizando el comando `sudo apt-get install p`
  - 2. Configuración del Certificado SSL: Una vez instalado certbot, se utiliza para configurar el certificado SSL para Apache.
    - Esto se logra ejecutando `sudo certbot --apache -d [nombre_de_dominio]` y también añadiendo el dominio con 'www' (`-d www.nombre_de_dominio`).
    - Esto se logra ejecutando `sudo certbot --apache -d [nombre_de_dominio]` y también añadiendo el dominio con 'www' (`-d www.nombre_de_dominio`).
    - Durante este proceso, se solicita una dirección de correo electrónico para la recuperación de la clave en caso de pérdida.
    - También se pregunta si se desea redirigir el tráfico HTTP a HTTPS, una acción que se confirma para asegurar la comunicación.
  - 3. Generación y Ubicación del Certificado: Una vez finalizada la instalación, los archivos del certificado generado se pueden encontrar en la ruta /etc/letsencrypt/live.
- Verificación y Seguridad Resultante:
  - Verificación del Estado: El estado del certificado SSL puede verificarse utilizando herramientas como SSL labs.com/ssltest.
  - Reglas de Seguridad Inbound (AWS): Para que la evaluación sea exitosa, es fundamental asegurarse de que las reglas de seguridad de entrada (inbound security rules) del servidor remoto (por ejemplo, en AWS EC2) permitan el tráfico HTTPS en el puerto predeterminado 443. Inicialmente, si solo HTTP y SSH están permitidos, se debe añadir explícitamente la regla HTTPS.
  - Resultados de la Certificación: Tras una configuración correcta, el certificado SSL será válido, firmado específicamente para el servidor con la autoridad emisora Let's Encrypt, y tendrá una validez inicial de aproximadamente tres meses.
  - Renovación: Dado que es una autoridad de certificación gratuita, los certificados de Let's Encrypt expiran cada tres meses y deben renovarse.
  - Redirección y Comunicación Segura: Una vez implementado, cualquier acceso al servidor se redirigirá automáticamente de HTTP a HTTPS, mostrando un 'candado verde' en el navegador que indica que la conexión es segura. Esto garantiza que 'todo cliente conectado con este servidor tendrá una comunicación encriptada de extremo a extremo'.

En el contexto más amplio de asegurar un dominio personalizado con SSL/TLS para un servidor IoT, Let's Encrypt es presentado como la solución preferida y gratuita para obtener los certificados necesarios que permiten la comunicación segura y encriptada (HTTPS) entre el servidor y todos sus clientes, siendo un paso esencial en el desarrollo de la seguridad de la infraestructura IoT.

Gratuito

Se enfatiza claramente que Let's Encrypt es una autoridad de certificación gratuita en el contexto más amplio de asegurar un dominio personalizado con SSL/TLS. Esto es este aspecto:

- Naturaleza Gratuita, Automatizada y de Código Abierto:
  - Let's Encrypt se describe como una 'autoridad de certificación gratuita, automatizada y de código abierto'.
  - Es ampliamente utilizada por muchos desarrolladores y empresas, y cuenta con el respaldo de importantes patrocinadores.
  - Su objetivo es permitir la obtención de certificados SSL/TLS.
- Implicación de la Gratuidad: Renovación Frecuente:
  - Una consecuencia directa de su naturaleza gratuita es que los certificados de Let's Encrypt 'expiran cada tres meses y deben renovarse'. Esto se menciona explícitamente como una característica de esta 'autoridad de certificación gratuita'.

En resumen, se recalcan que Let's Encrypt ofrece una solución sin costo para la obtención de certificados SSL/TLS, lo que la convierte en una opción muy accesible para asegurar la comunicación en servidores, incluyendo aquellos utilizados en proyectos IoT. La contrapartida de ser gratuito es la necesidad de renovaciones periódicas cada tres meses.

Automatizado

Se destaca que Let's Encrypt es una autoridad de certificación automatizada en el contexto más amplio de asegurar un dominio personalizado con SSL/TLS. Esto es este aspecto:

- Naturaleza Automatizada de Let's Encrypt:
  - Let's Encrypt se describe explícitamente como una 'autoridad de certificación gratuita, automatizada y de código abierto'. Es ampliamente utilizada por muchos desarrolladores y empresas, y cuenta con el respaldo de importantes patrocinadores.
  - Su propósito es facilitar la obtención de certificados SSL/TLS.
- Implementación a Través de certbot:
  - La naturaleza automatizada de Let's Encrypt se materializa mediante el uso de software de terceros como certbot.
  - El proceso para utilizarlo implica:
    - 1. Instalar certbot en el servidor, añadiendo su repositorio y luego instalando el paquete `python-certbot-apache`.
    - 2. Configurar el certificado SSL ejecutando comandos específicos de certbot (ej. `sudo certbot --apache -d` Este proceso automatizado guía al usuario a través de la configuración, solicitando un correo electrónico para recuperación de clave y preguntando si se desea redirigir el tráfico HTTP a HTTPS.
- Implicaciones de la Automatización:
  - La automatización simplifica significativamente el proceso de obtención y configuración de certificados SSL/TLS, haciendo que la seguridad HTTPS sea más accesible.
  - A pesar de ser automatizado, los certificados de Let's Encrypt 'expiran cada tres meses y deben renovarse'. Aunque no se detallan el comando exacto para la renovación automática, el hecho de ser una CA 'automatizada' implica que este proceso puede ser gestionado de forma programada por herramientas como certbot, reduciendo la intervención manual necesaria.

En resumen, se enfatizan que la capacidad de Let's Encrypt para ser una autoridad de certificación automatizada es una de sus características fundamentales, lo que simplifica en gran medida la tarea de obtener, configurar y gestionar certificados SSL/TLS para asegurar la comunicación en servidores, incluyendo aquellos en el ámbito del IoT, aunque estos certificados requieran renovaciones periódicas.

Código Abierto

Se destaca que Let's Encrypt es una autoridad de certificación de código abierto en el contexto más amplio de asegurar un dominio personalizado con SSL/TLS. Esto es este aspecto:

- Naturaleza de Código Abierto, Gratuita y Automatizada:
  - Let's Encrypt se describe explícitamente como una 'autoridad de certificación gratuita, automatizada y de código abierto'.
  - Se menciona que es ampliamente utilizada por muchos desarrolladores y empresas, y que cuenta con el respaldo de importantes patrocinadores.
  - Su objetivo es permitir la obtención de certificados SSL/TLS.

En resumen, se indican que la característica de ser de código abierto es fundamental para Let's Encrypt, junto con ser gratuita y automatizada. Esto contribuye a su adopción generalizada y al respaldo que recibe, facilitando así la obtención de certificados SSL/TLS para asegurar la comunicación en servidores, como los utilizados en proyectos IoT.

Respaldado por Grandes Patrocinadores

Aquí se indica claramente que Let's Encrypt cuenta con el respaldo de grandes patrocinadores en el contexto más amplio de su rol como Autoridad de Certificación. Esto es este aspecto:

- Respaldo de Patrocinadores Importantes:
  - Let's Encrypt se describe como una 'autoridad de certificación gratuita, automatizada y de código abierto'.
  - Se utiliza ampliamente por 'muchos desarrolladores y empresas' y, crucialmente, está 'respaldada por grandes patrocinadores'. Aunque no se lista explícitamente los nombres de los patrocinadores, menciona 'like all of these', sugiriendo que son numerosos y significativos.

En resumen, se enfatizan que el hecho de que Let's Encrypt esté respaldada por grandes patrocinadores es una de sus características clave, junto con ser gratuita, automatizada y de código abierto. Este respaldo contribuye a su credibilidad y capacidad para ofrecer certificados SSL/TLS ampliamente adoptados para asegurar dominios personalizados, incluyendo aquellos en el ámbito del Internet de las Cosas (IoT).

5.0.3. Instalación de CERTBOT (Software de Terceros)

Se proporciona información detallada sobre la instalación de software de terceros, específicamente Certbot, en el contexto de asegurar un dominio personalizado con certificados SSL/TLS de Let's Encrypt.

- Contexto General: Asegurar un Dominio Personalizado con SSL/TLS.
- El objetivo principal de este proceso es asegurar un dominio personalizado utilizando certificados SSL/TLS de Let's Encrypt para habilitar HTTPS. Esto es parte de la etapa 5 del curso 'Internet of Things using Python and Raspberry Pi', que se enfoca en proteger un servidor HTTP IoT y la autenticación de usuarios. Se menciona también que la seguridad en Internet y la criptografía son temas clave en la etapa 3, incluyendo protocolos SSL/TLS (HTTPs). Después de asegurar el dominio con Let's Encrypt, el servidor redirigirá el tráfico de HTTP a HTTPS, mostrando un candado verde que indica que la conexión es segura, y toda comunicación entre el cliente y el servidor estará encriptada de extremo a extremo.
- Instalación y Uso de Certbot:
  - 1. Autoridad de Certificación: Para obtener un certificado SSL/TLS, se utiliza Let's Encrypt, una autoridad de certificación gratuita, automatizada y de código abierto, respaldada por importantes patrocinadores y utilizada por muchos desarrolladores y empresas.
  - 2. Software de Terceros (Certbot): El primer paso para usar Let's Encrypt es instalar software de terceros en el servidor. Este software es Certbot, el cliente de Let's Encrypt.
  - 3. Pasos de Instalación:
    - Conexión al Servidor: Se debe conectar al servidor, por ejemplo, usando PuTTY.

    - Añadir el Repositorio: Se añade el repositorio de Certbot utilizando el comando `sudo add-apt-repository` Este es un PPA (Package Archive) preparado por el equipo de Debian Let's Encrypt y de respaldo para Ubuntu.
    - Actualizar Lista de Paquetes: Después de añadir el repositorio, se actualiza la lista de paquetes con `sudo apt-get update` para que el nuevo repositorio sea reconocido.
    - Instalar Certbot: Finalmente, se instala Certbot desde el nuevo repositorio con el comando `sudo apt-get install python-certbot-apache`.
- 4. Configuración del Certificado SSL con Certbot: Una vez instalado, Certbot se usa para configurar el certificado SSL para Apache:
    - Se ejecuta el comando `sudo certbot --apache -d [nombre_de_dominio] -d www.[nombre_de_dominio`
    - Información Requerida: Se pedirá una dirección de correo electrónico para la recuperación de la clave en caso de pérdida.
    - Redirección HTTPS: Se preguntará si se desea redirigir el tráfico HTTP a HTTPS, lo cual es altamente recomendable.
    - Ubicación de los Certificados: Una vez finalizada la instalación, los archivos del certificado generado se pueden encontrar en /etc/letsencrypt/live.
- 5. Verificación del Certificado:
    - Se puede verificar el estado del certificado SSL en SSL labs.com/ssltest proporcionando el nombre de dominio.
    - Reglas de Seguridad Inbound: Si la evaluación falla ('unable to connect to the server'), puede ser debido a la falta de reglas de seguridad inbound para HTTPS en el servidor remoto (por ejemplo, en AWS). Es necesario editar las reglas de inbound para añadir HTTPS (puerto 443) y luego guardar.
    - Una vez configurado correctamente, la verificación mostrará los detalles del certificado SSL, indicando que está firmado para el servidor (ej. 'packt IOT server'), emitido por Let's Encrypt, y con una validez de aproximadamente tres meses. Dado que es una autoridad de certificación gratuita, los certificados expiran cada tres meses y deben ser renovados.

En resumen, Certbot es una herramienta esencial y de código abierto que facilita la obtención y configuración de certificados SSL/TLS de Let's Encrypt para asegurar un dominio, lo que permite la comunicación encriptada (HTTPS) entre los clientes y el servidor.

CONECTAR AL SERVIDOR (Putty)

En el contexto de la instalación de software de terceros (Certbot) para asegurar un dominio personalizado con SSL/TLS, se indican que la conexión al servidor es el primer paso fundamental, y PuTTY es la herramienta específica mencionada para llevar a cabo esta conexión. Aquí se detalla lo que se dice sobre 'Conectar al Servidor (PuTTY)':

- Paso Inicial para la Instalación de Certbot: Antes de proceder con la instalación de Certbot, es necesario establecer una conexión con el servidor. Se especifica que el primer paso para utilizar Let's Encrypt y obtener un certificado SSL es instalar software de terceros en el servidor, y para ello, primero se conecta al servidor usando PuTTY.
- Secuencia de Comandos: Una vez conectado al servidor mediante PuTTY, el siguiente paso inmediato es añadir el repositorio de Certbot utilizando el comando `sudo add-apt-repository ppa:certbot/certbot`. Esto establece claramente la posición de la conexión al servidor como un requisito previo a cualquier comando de instalación.
- Contexto de Servidores Remotos: Aunque no se menciona directamente PuTTY en todos los contextos de conexión, la necesidad de acceder a un servidor remoto (como una instancia EC2 de AWS o una Raspberry Pi) para realizar configuraciones es recurrente. Por ejemplo, en el caso de una Raspberry Pi, se menciona que el 'escritorio remoto' y SSH pueden requerir que SSH esté habilitado por defecto si está deshabilitado, y para ello se debe usar `sudo raspi-config` en el terminal de la Raspberry Pi. PuTTY es una herramienta comúnmente utilizada en Windows para establecer conexiones SSH a servidores remotos, lo que alinea su uso con la gestión de estos entornos de servidor.

En resumen, PuTTY se presenta como la interfaz de conexión esencial para iniciar el proceso de instalación de Certbot en el servidor, permitiendo ejecutar los comandos necesarios para configurar los certificados SSL/TLS.

AGREGAR REPOSITORIO CERTBOT

En el contexto más amplio de la instalación de software de terceros (Certbot) para asegurar un dominio personalizado con certificados SSL/TLS, se detallan que agregar el repositorio de Certbot (`sudo add-apt-repository ppa:certbot/certbot`) es el primer comando crucial que se ejecuta en el servidor, después de haber establecido la conexión. A continuación, se detalla este aspecto:

- 1. El Primer Paso para la Instalación de Certbot: Se establece que, para utilizar la autoridad de certificación Let's Encrypt y obtener un certificado SSL, el primer paso es instalar software de terceros en el servidor. Inmediatamente después de conectarse al servidor (por ejemplo, usando PuTTY, como se mencionó en nuestra conversación anterior), la primera acción es añadir el repositorio de Certbot.
- 2. El Comando Específico: El comando exacto proporcionado es `sudo add-apt-repository ppa:certbot/certbot`.
- 3. Propósito del Repositorio PPA:
  - Se explica que este es un PPA (Package Archive).
  - Este PPA ha sido preparado por el equipo de Debian Let's Encrypt y de respaldo para Ubuntu.
  - La adición de este repositorio es fundamental porque permite que el sistema operativo del servidor sepa dónde encontrar los paquetes de instalación más recientes y correctos para Certbot.
- 4. Secuencia Post-Adición del Repositorio: Una vez que el repositorio ha sido añadido, el siguiente paso es actualizar la lista de paquetes con `sudo apt-get update`. Esta actualización es necesaria para que el sistema reconozca el nuevo repositorio que se acaba de añadir y pueda ver los paquetes disponibles en él. Solo después de esta actualización se puede proceder a instalar Certbot (`sudo apt-get install python-certbot-apache`) desde el nuevo repositorio.

En resumen, el comando `sudo add-apt-repository ppa:certbot/certbot` es una instrucción indispensable que prepara el entorno del servidor para la instalación de Certbot, asegurando que el sistema operativo tenga acceso a las versiones necesarias del software, lo que a su vez es un paso fundamental para obtener y configurar los certificados SSL/TLS de Let's Encrypt y, en última instancia, asegurar el servidor HTTP IoT y permitir la comunicación encriptada de extremo a extremo.

ACTUALIZAR LISTA DE PAQUETES

En el contexto más amplio de la instalación de software de terceros (Certbot) para asegurar un dominio personalizado con certificados SSL/TLS, se indican que 'Actualizar Lista de Paquetes (`sudo apt-get update`)' es un paso intermedio esencial después de añadir el repositorio de Certbot y antes de su instalación final. A continuación, se detalla este aspecto:

- 1. Propósito del Comando: Se explica explícitamente que después de añadir el nuevo repositorio (el PPA de Certbot), es necesario actualizar la lista de paquetes para 'recoger' (pick up) el nuevo repositorio que se acaba de añadir. Esto significa que el sistema operativo necesita ser informado sobre los nuevos orígenes de software disponibles.
- 2. El Comando Específico: El comando proporcionado para esta acción es `sudo apt-get update`.
- 3. Lugar en la Secuencia de Instalación de Certbot:
  - Este comando se ejecuta después de haber añadido el repositorio de Certbot con `sudo add-apt-repository ppa`
  - Se ejecuta antes de instalar Certbot con `sudo apt-get install python-certbot-apache`.
  - La secuencia de pasos que se sigue es: conectarse al servidor (por ejemplo, con PuTTY), añadir el repositorio de Certbot, actualizar la lista de paquetes e, inmediatamente después, instalar Certbot.

En resumen, `sudo apt-get update` es un paso crítico para asegurar que el sistema del servidor esté al tanto de los paquetes de software disponibles en el repositorio de Certbot recién agregado. Sin esta actualización, el sistema no podría encontrar ni instalar el cliente de Let's Encrypt (Certbot), lo que detendría el proceso de asegurar el dominio personalizado con certificados SSL/TLS y habilitar la comunicación HTTPS encriptada de extremo a extremo.

INSTALAR CERTBOT PARA APACHE

En el contexto más amplio de la instalación de software de terceros (Certbot) para asegurar un dominio personalizado con certificados SSL/TLS, se indican que el comando `sudo apt-get install python-certbot-apache` es el paso final para la instalación del cliente Certbot en el servidor, haciéndolo operativo y listo para configurar los certificados. A continuación, se detalla este aspecto:

- 1. Propósito del Comando: Este comando instala el cliente de Let's Encrypt, Certbot, junto con su plugin específico para el servidor web Apache. Se afirma que después de ejecutarlo, 'el cliente de Let's Encrypt, Certbot, ya está listo para ser usado'.
- 2. Lugar en la Secuencia de Instalación de Certbot:
  - Este comando se ejecuta después de haber añadido el repositorio de Certbot (`sudo add-apt-repository ppa:ce`
  - También se ejecuta después de haber actualizado la lista de paquetes (`sudo apt-get update`) para que el sistema reconozca el nuevo repositorio.
  - Es el último paso en la secuencia de preparación del sistema para la configuración del certificado, siguiendo la conexión al servidor (ej. PuTTY), la adición del repositorio y la actualización de la lista de paquetes.
- 3. Habilitación de la Configuración SSL: Una vez que `sudo apt-get install python-certbot-apache` ha sido ejecutado con éxito, el cliente Certbot está preparado para configurar el certificado SSL para Apache. El siguiente paso directo es ejecutar `sudo certbot --apache -d [nombre_de_dominio] -d www.[nombre_` para iniciar la configuración del certificado en sí.
- 4. Contexto de Asegurar el Dominio: La instalación de este software de terceros (Certbot) es el primer paso crucial para utilizar Let's Encrypt como autoridad de certificación. Al instalar python-certbot-apache, se obtiene la herramienta necesaria para automatizar la obtención y configuración de los certificados SSL/TLS, que es fundamental para asegurar el servidor HTTP IoT y permitir la comunicación encriptada de extremo a extremo (HTTPS).

En síntesis, `sudo apt-get install python-certbot-apache` es la culminación de la fase de preparación del software Certbot en el servidor, permitiendo que el cliente de Let's Encrypt esté disponible para configurar y gestionar los certificados SSL/TLS que asegurarán el dominio personalizado y habilitarán HTTPS.

5.0.4. Configuración del Certificado SSL para APACHE

Se detalla que 'Configurar Certificado SSL para Apache' es el paso crucial que sigue a la instalación del cliente Certbot y es el que finalmente permite asegurar un dominio personalizado con SSL/TLS de Let's Encrypt. Este proceso es fundamental para la etapa 5 del curso 'Internet of Things using Python and Raspberry Pi', que se centra en proteger un servidor HTTP IoT y la autenticación de usuarios. Aquí se desglosa lo que se dice sobre este proceso:

- 1. Propósito y Contexto Amplio de Asegurar el Dominio: El objetivo es asegurar un dominio personalizado con certificados SSL/TLS para habilitar HTTPS. Una vez que el dominio está asegurado con Let's Encrypt, el servidor redirigirá automáticamente el tráfico de HTTP a HTTPS, mostrando un candado verde en el navegador que indica que la conexión es segura, y toda la comunicación entre el cliente y el servidor estará encriptada de extremo a extremo. Esto se basa en la autoridad de certificación gratuita, automatizada y de código abierto Let's Encrypt.
- 2. Ejecución del Cliente Certbot para Configurar el Certificado SSL: Después de haber instalado Certbot para Apache (mediante `sudo apt-get install python-certbot-apache`), el cliente de Let's Encrypt ya está listo para ser usado. El siguiente paso es configurar el certificado SSL para Apache.

- Comando Específico: Para ello, se ejecuta el comando `sudo certbot --apache -d [nombre_de_dominio] -d w` Este comando instruye a Certbot para que genere y configure el certificado SSL para el dominio especificado, incluyendo su versión con 'www'.

- 3. Interacciones y Opciones Durante la Configuración: Durante la ejecución del comando, Certbot solicitará cierta información y ofrecerá opciones:
  - Dirección de Correo Electrónico: Se pedirá una dirección de correo electrónico, la cual se utilizará para la recuperación de la clave en caso de pérdida.
  - Redirección HTTP a HTTPS: Se preguntará si se desea redirigir el tráfico HTTP a HTTPS. Se enfatiza que definitivamente se desea hacer esto, indicando que se debe seleccionar la opción de redirigir.
- 4. Resultado de la Configuración: Una vez finalizada la instalación y configuración, los archivos del certificado generado se pueden encontrar en la ubicación /etc/letsencrypt/live.
- 5. Verificación del Estado del Certificado SSL: Es fundamental verificar que el certificado se haya configurado correctamente:
  - Herramienta de Verificación: Se puede verificar el estado del certificado SSL y su configuración yendo a SSL labs.com/ssltest y proporcionando el nombre del DNS.
  - Solución de Problemas (Reglas de Seguridad Inbound): Si la evaluación inicial falla, indicando 'unable to connect to the server' (incapaz de conectar al servidor), la razón común es la falta de reglas de seguridad inbound para HTTPS en el servidor remoto (por ejemplo, en AWS). Es necesario editar las reglas de inbound para añadir HTTPS (el puerto predeterminado para HTTPS es 443) y luego guardar los cambios.
  - Confirmación Exitosa: Después de ajustar las reglas de seguridad, la verificación mostrará los detalles del certificado SSL, confirmando que está firmado para el servidor (ej. 'packt IOT server'), emitido por Let's Encrypt, y con una validez de aproximadamente tres meses. Es importante recordar que, al ser una autoridad de certificación gratuita, los certificados expiran cada tres meses y deben renovarse.

En conclusión, la configuración del certificado SSL para Apache utilizando Certbot es el paso que materializa la seguridad del dominio. Implica la ejecución de un comando específico, la interacción con algunas preguntas para definir preferencias (como la redirección HTTPS), y la verificación posterior para asegurar que el certificado esté activo y que las reglas de seguridad del servidor permitan la comunicación segura, logrando así una comunicación encriptada de extremo a extremo.

EJECUTAR CERBOT

En el contexto más amplio de la configuración de un Certificado SSL para Apache, se indican que el comando `sudo certbot --apache -d [nombre_de_dominio] -d www.[nombre_de_dominio]` es la instrucción clave que se ejecuta para activar el cliente de Let's Encrypt y proceder con la obtención y configuración del certificado SSL/TLS para el servidor web Apache. Este es un paso fundamental para asegurar el dominio personalizado y habilitar la comunicación HTTPS. A continuación, se detalla lo que se dice sobre la ejecución de este comando:

- 1. Propósito del Comando:
  - Una vez que el cliente de Let's Encrypt (Certbot) ha sido instalado y está listo para usarse, este comando se utiliza específicamente para configurar el certificado SSL para Apache.
  - El comando especifica el servidor web (`--apache`) y los dominios para los cuales se desea el certificado (`-d [nombre_de_dominio] -d www.[nombre_de_dominio]`), asegurando que tanto el dominio principal como su versión con 'www' estén cubiertos.
- 2. Interacciones Durante la Ejecución:
  - Cuando se ejecuta el comando, Certbot solicita una dirección de correo electrónico. Esta dirección se utilizará para la recuperación de la clave en caso de que se pierda.

- También se le preguntará al usuario si desea redirigir el tráfico HTTP a HTTPS. Se enfatiza que 'definitivamente queremos eso' y se debe seleccionar la opción de redirigir, lo que significa que el servidor automáticamente reenviará todas las solicitudes HTTP al puerto seguro HTTPS (puerto 443).

- 3. Resultado de la Ejecución Exitosa:
  - Una vez que la instalación y configuración han finalizado, los archivos del certificado generado se pueden encontrar en la ubicación /etc/letsencrypt/live.
  - El servidor se configurará para redirigir de HTTP a HTTPS, y se mostrará un candado verde en el navegador, indicando una conexión segura y comunicación encriptada de extremo a extremo entre el cliente y el servidor.
- 4. Verificación Post-Configuración:
  - Se sugiere verificar el estado del certificado SSL y su configuración utilizando una herramienta como SSL labs.com/ssltest, donde se debe proporcionar el nombre de dominio.
  - Si la evaluación inicial falla con un mensaje como 'unable to connect to the server', se señala que la causa más probable es la falta de reglas de seguridad inbound para HTTPS (puerto 443) en el servidor remoto (por ejemplo, en AWS). Es necesario editar estas reglas para añadir el puerto HTTPS y luego guardar.
  - Una vez corregidas las reglas, la verificación exitosa mostrará los detalles del certificado SSL, confirmando que está firmado para el servidor (ej. 'packt IOT server'), emitido por Let's Encrypt, y válido por aproximadamente tres meses. Se recuerda que, al ser una autoridad de certificación gratuita, los certificados deben renovarse cada tres meses.

En resumen, el comando `sudo certbot --apache -d [nombre_de_dominio] -d www.[nombre_de_dominio]` es el corazón del proceso de configuración SSL para Apache, ya que orquesta la obtención, instalación y ajuste de los certificados de Let's Encrypt. Es un paso crítico para transformar una conexión HTTP en una conexión HTTPS segura, protegiendo así la comunicación de un servidor IoT.

PROPORCIONAR CORREO ELECTRóNICO (RECUPERACIóN DE CLAVE)

En el contexto más amplio de la Configuración de Certificado SSL para Apache, se indican que proporcionar una dirección de correo electrónico es un paso interactivo y obligatorio durante la ejecución del comando certbot, y su propósito es fundamental para la recuperación de la clave del certificado. Aquí se detalla este aspecto:

- 1. Momento de la Solicitud: Después de ejecutar el comando `sudo certbot --apache -d [nombre_de_dominio] -d` para configurar el certificado SSL para Apache, Certbot solicitará información específica al usuario.
- 2. Información Solicitada: Una de las solicitudes es 'proporcionar un correo electrónico'.
- 3. Propósito: Recuperación de Clave: La razón explícita para solicitar este correo electrónico es que 'este correo electrónico se utilizará para la recuperación en caso de que perdamos la clave'. Esto asegura que si se pierde el acceso a la clave privada del certificado SSL, hay un mecanismo para recuperarla, manteniendo así la continuidad de la seguridad del dominio.
- 4. Contexto General de Seguridad: Esta interacción es parte integral del proceso de asegurar un dominio personalizado con certificados SSL/TLS de Let's Encrypt. La capacidad de recuperar la clave es un aspecto importante de la gestión de certificados y la seguridad a largo plazo de un servidor HTTP IoT, asegurando que el servidor pueda mantener su comunicación encriptada de extremo a extremo.

En resumen, cuando se está configurando el certificado SSL para Apache utilizando Certbot, se requiere que el usuario proporcione una dirección de correo electrónico, cuya función principal es servir como método de recuperación en caso de que la clave del certificado SSL se pierda, garantizando así la capacidad de mantener el dominio seguro con HTTPS.

REDIRECCIóN DE HTTP A HTTPS (TYPE 2 TO ENTER)

En el contexto más amplio de la Configuración del Certificado SSL para Apache, se destacan que la redirección de HTTP a HTTPS, específicamente al seleccionar la opción '2', es una elección crucial que se presenta durante la ejecución de Certbot y es altamente recomendada para garantizar la seguridad total del dominio. Aquí se detalla este aspecto:

- 1. Momento de la Solicitud: Después de ejecutar el comando `sudo certbot --apache -d [nombre_de_dominio] -d` para configurar el certificado SSL para Apache, Certbot interactúa con el usuario solicitando información y preferencias. Una de las preguntas que se formula es 'si queremos redirigir nuestro tráfico HTTP a HTTPS'.
- 2. La Opción Específica (Seleccionar 2): Se menciona que se debe 'escribir 2 para entrar' para seleccionar esta opción. Esto indica que '2' es la opción numérica correspondiente a la acción de redirigir el tráfico.
- 3. Recomendación Enfatizada: La instrucción es clara: 'definitivamente queremos eso'. Esto subraya la importancia de esta redirección, haciendo entender que es una práctica estándar y altamente deseable en la configuración de un servidor seguro.
- 4. Propósito y Beneficios de la Redirección:
  - La redirección asegura que todo el tráfico que inicialmente llegue a través del protocolo HTTP inseguro sea automáticamente reenviado al protocolo HTTPS seguro.
  - Esto es fundamental para la seguridad de extremo a extremo (end-to-end encrypted communication) entre el cliente y el servidor.
  - Al activar esta redirección, el servidor exhibirá un 'candado verde' en el navegador, lo que indica visualmente a los usuarios que la conexión es segura.
  - Este proceso es parte integral de la etapa 5 del curso, que se enfoca en proteger un servidor HTTP IoT y la autenticación de usuarios, donde la seguridad en Internet y la criptografía, incluidos los protocolos SSL/TLS (HTTPS), son temas clave.

En resumen, seleccionar la opción '2' para redirigir el tráfico de HTTP a HTTPS durante la configuración del certificado SSL con Certbot es un paso interactivo que se recomiendan enfáticamente. Es esencial para garantizar que todas las comunicaciones con el dominio personalizado sean encriptadas, proporcionando una conexión segura y una mejor experiencia para el usuario al mostrar un candado verde en el navegador.

5.0.5. Ubicación de Certificados Generados: /etc/letsencrypt/live/

En el contexto más amplio de Asegurar un Dominio Personalizado con SSL/TLS, se indican que la ubicación de los certificados generados, específicamente en /etc/letsencrypt/live, es el resultado final y tangible del proceso de configuración exitosa con Certbot. Aquí se detalla este aspecto:

- 1. Resultado de la Instalación y Configuración: Una vez que la instalación y configuración del certificado SSL para Apache utilizando Certbot ha finalizado, el sistema almacenará los archivos correspondientes. Se afirma claramente que, después de completar este proceso, 'deberíamos poder encontrar nuestros archivos de certificado generados en /etc/letsencrypt/live'.
- 2. Lugar en el Proceso General: Esta ubicación se menciona inmediatamente después de los pasos interactivos de Certbot, como proporcionar una dirección de correo electrónico para la recuperación de la clave y seleccionar la opción de redirigir el tráfico HTTP a HTTPS. Es decir, una vez que Certbot ha ejecutado su función y completado la configuración, estos archivos se almacenan en la ruta especificada.
- 3. Importancia en la Seguridad del Dominio: La presencia de estos archivos en esta ubicación significa que el dominio personalizado ha sido exitosamente asegurado con certificados SSL/TLS de Let's Encrypt. Estos certificados son los que permiten que el servidor redirija el tráfico de HTTP a HTTPS, muestre un 'candado verde' en el navegador y establezca una comunicación encriptada de extremo a extremo entre el cliente y el servidor.

En resumen, /etc/letsencrypt/live es el directorio estándar donde Certbot almacena los certificados SSL/TLS emitidos por Let's Encrypt para un dominio. Su existencia confirma que el proceso de asegurar el dominio personalizado ha sido completado con éxito, lo que es fundamental para la seguridad del servidor HTTP IoT y la autenticación de usuarios.

5.0.6. Verificar Estado del Certificado: SSL LABS (ssltest.com)

En el contexto más amplio de Asegurar un Dominio Personalizado con SSL/TLS, se indican que la verificación del estado del certificado SSL utilizando SSL Labs (ssltest.com) es un paso esencial y final para confirmar que la configuración del certificado se ha realizado correctamente y que el dominio está efectivamente protegido. A continuación, se detalla este aspecto:

- 1. Propósito de la Verificación: Una vez que se ha instalado Certbot y se ha configurado el certificado SSL para Apache, es fundamental verificar su estado. Se afirma que 'ahora también podemos verificar el estado de nuestro certificado SSL yendo a SSL labs.com/ssltest'. Esta verificación asegura que el proceso fue exitoso.
- 2. Cómo Realizar la Verificación: Para utilizar la herramienta, se debe proporcionar el nombre de DNS (nombre de dominio) en SSL Labs.
- 3. Posibles Resultados y Solución de Problemas:
  - Fallo en la Evaluación: Si la evaluación inicial falla, mostrando un mensaje como 'unable to connect to the server' (incapaz de conectar al servidor), se identifican una razón común para esto: la falta de reglas de seguridad inbound para HTTPS en el servidor remoto (ej. en AWS).
  - Corrección Necesaria: Para solucionar este problema, es necesario editar las reglas de seguridad de entrada (inbound security rules) del servidor y añadir HTTPS, cuyo puerto predeterminado es 443, y luego guardar los cambios.
- 4. Confirmación Exitosa de la Verificación:
  - Una vez que las reglas de seguridad se han configurado correctamente, la verificación en SSL Labs mostrará los detalles del certificado SSL.
  - Estos detalles incluyen que el certificado está específicamente firmado para el servidor (ej. 'packt IOT server'), que la autoridad de emisión es Let's Encrypt, y que es válido desde la fecha actual hasta aproximadamente tres meses. Se hace una observación importante sobre que, al ser una autoridad de certificación gratuita, los certificados de Let's Encrypt expiran cada t res meses y deben renovarse.
- 5. Relevancia en el Contexto General: La verificación exitosa en SSL Labs es la confirmación de que el dominio está completamente asegurado. Esto significa que el servidor redirigirá el tráfico de HTTP a HTTPS, mostrando el 'candado verde' en el navegador, y que toda la comunicación entre el cliente y el servidor estará encriptada de extremo a extremo. Este logro es central para la seguridad del servidor HTTP IoT y la autenticación de usuarios, temas clave en la etapa 5 del curso.

En resumen, SSL Labs (ssltest.com) es una herramienta de diagnóstico externa fundamental para validar la correcta implementación del certificado SSL/TLS en un dominio personalizado. Permite al usuario confirmar que el certificado está activo y bien configurado, y, en caso de problemas, ayuda a identificar y corregir fallos como las reglas de seguridad del servidor.

PROBLEMA INICIAL: FALTAN REGLAS DE SEGURIDAD DE ENTRADA HTTPS EN AWS EC2

En el contexto más amplio de la Verificación del Estado del Certificado SSL mediante SSL Labs (ssltest.com), se identifican un 'Problema Inicial: Faltan Reglas de Seguridad de Entrada HTTPS en AWS EC2' como una causa común de falla en la evaluación del certificado, lo que impide que el dominio sea reconocido como seguro. Aquí se detalla este aspecto:

- 1. Momento de la Detección del Problema: Después de haber instalado Certbot y configurado el certificado SSL para Apache, se sugieren verificar el estado del certificado en SSL labs.com/ssltest proporcionando el nombre de DNS. Sin embargo, la evaluación inicial 'ha fallado', mostrando el mensaje 'unable to connect to the server' (incapaz de conectar al servidor).
- 2. Identificación de la Causa Raíz (AWS EC2 y Reglas de Seguridad Inbound): Se identifica inmediatamente la razón de este fallo: 'no hemos asignado reglas de seguridad de entrada (inbound security rules) para HTTPS en nuestro servidor remoto de AWS'. Esto indica que el servidor (en este caso, una instancia EC2 de AWS) no tiene configuradas las reglas necesarias para permitir el tráfico seguro a través de HTTPS.

- 3. Solución al Problema:
  - Para corregirlo, se accede a la configuración de la instancia EC2 (por ejemplo, a través de 'ec2 instance launch wizard two inbounds').
  - Se observa que inicialmente solo se tienen reglas para HTTP y SSH.
  - Se debe editar estas reglas de seguridad y 'añadir regla HTTPS'.
  - El puerto predeterminado para HTTPS es 443, y se debe guardar esta configuración.
- 4. Confirmación de la Solución y Verificación Exitosa:
  - Una vez que las reglas de seguridad han sido modificadas para permitir el tráfico HTTPS, se reintenta la verificación en SSL Labs.
  - Tras un breve período, la evaluación se completa con éxito, mostrando los detalles del certificado SSL, incluyendo que está firmado para el servidor (ej. 'packt IOT server'), emitido por Let's Encrypt, y válido por aproximadamente tres meses.
  - Esto confirma que el servidor ahora es accesible a través de HTTPS y que el certificado está correctamente configurado.

En resumen, la falta de reglas de seguridad de entrada para HTTPS en un servidor remoto como AWS EC2 es un obstáculo significativo que puede hacer que la verificación del certificado SSL falle. Se resalta la importancia de configurar el puerto 443 para HTTPS en las reglas de seguridad de entrada como un paso crítico para asegurar que el certificado SSL pueda ser correctamente validado y que el dominio personalizado logre una comunicación encriptada de extremo a extremo.

SOLUCIóN: AGREGAR REGLAS HTTPS (PUERTO 443) EN AWS INBOUND RULES

En el contexto más amplio de la Verificación del Estado del Certificado SSL mediante SSL Labs (ssltest.com), se detallan que la 'Solución: Agregar Regla HTTPS (Puerto 443) en AWS Inbound Rules' es la acción correctiva directa para resolver el problema de conexión (unable to connect to the server) que se presenta cuando el servidor remoto (específicamente en AWS) no tiene configuradas las reglas de seguridad necesarias para permitir el tráfico HTTPS. A continuación, se detalla este aspecto:

- 1. Detección del Problema y Necesidad de la Solución: Después de intentar verificar el certificado SSL en SSL labs.com/ssltest, el 'assessment ha fallado' con el mensaje 'unable to connect to the server'. La razón identificada para este fallo es que 'no hemos asignado reglas de seguridad de entrada (inbound security rules) para HTTPS en nuestro servidor remoto de AWS'. Esto significa que, aunque el certificado podría estar instalado, el firewall del servidor está bloqueando el acceso a través del puerto seguro.
- 2. Implementación de la Solución (Pasos en AWS EC2):
  - Para corregir esto, se debe acceder a la configuración de la instancia EC2.
  - Las reglas de seguridad existentes (inbound rules) típicamente solo incluyen HTTP y SSH.
  - La solución consiste en 'editar esto [las reglas de seguridad] y añadir regla HTTPS'.
  - Se especifica claramente que el 'puerto predeterminado para HTTP[S] es 443'. Por lo tanto, se debe configurar esta regla para el puerto 443.
  - Finalmente, es necesario guardar los cambios realizados en las reglas de seguridad.
- 3. Resultado de la Aplicación de la Solución:
  - Una vez que se añaden las reglas de seguridad para HTTPS y se guardan, se puede intentar 'refresh this' (actualizar) la verificación en SSL Labs.
  - Después de un corto período de tiempo ('takes a bit time to complete'), la evaluación se completará con éxito.
  - La verificación exitosa mostrará los detalles del certificado SSL, confirmando que está firmado para el servidor (ej. 'packt IOT server'), emitido por Let's Encrypt, y válido por aproximadamente tres meses.

- Esto es la confirmación de que el servidor ahora es accesible a través de HTTPS y que el dominio personalizado está efectivamente asegurado, permitiendo una comunicación encriptada de extremo a extremo.

En síntesis, la acción de agregar la regla HTTPS (puerto 443) a las reglas de seguridad de entrada en AWS EC2 es la solución directa y necesaria para permitir que SSL Labs (y, por extensión, cualquier cliente web) se conecte al servidor a través del protocolo seguro. Este paso es indispensable para validar la correcta implementación del certificado SSL/TLS y garantizar que el dominio esté verdaderamente protegido con HTTPS.

5.0.7. Detalles del Certificado: Firmado Para 'pact IOT server', Emitido por Let's Encrypt. Válido por 3 meses.

En el contexto más amplio de Asegurar un Dominio Personalizado con SSL/TLS, se detallan que los 'Detalles del Certificado: Firmado para 'pact IOT server', Emitido por Let's Encrypt, Válido por 3 Meses' son la confirmación final y tangible del éxito en la configuración del certificado, lo que indica que el dominio está debidamente protegido. Estos detalles se observan después de una verificación exitosa del estado del certificado, por ejemplo, utilizando SSL Labs (ssltest.com) y tras haber resuelto posibles problemas como la falta de reglas de seguridad de entrada HTTPS en el servidor remoto. A continuación, se desglosa lo que se dice sobre estos detalles:

- 1. Firmado para 'pact IOT server':
  - Se indica que el certificado SSL está 'específicamente firmado para nuestro propio servidor, pact IOT server'.
  - Al verificar el certificado, se muestra que está 'emitido a pact IOT server'.
  - Esto significa que el certificado ha sido generado y es válido para el dominio asociado con el 'pact IOT server', confirmando que la seguridad se aplica correctamente a ese servidor específico.
- 2. Emitido por Let's Encrypt:
  - La 'autoridad emisora es Let's Encrypt'.
  - Let's Encrypt se describe como una autoridad de certificación (CA) gratuita, automatizada y de código abierto, utilizada por muchos desarrolladores y empresas, y respaldada por importantes patrocinadores.
  - La decisión de utilizar Let's Encrypt como CA es el primer paso para obtener un certificado SSL y es parte de la estrategia para asegurar el dominio personalizado. El curso se enfoca en asegurar el dominio utilizando esta autoridad de certificación.
- 3. Válido por 3 Meses:
  - El certificado es 'válido desde hoy hasta casi tres meses'.
  - Se explica que, dado que Let's Encrypt es una autoridad de certificación gratuita, los certificados expiran cada tres meses.
  - Esto implica que los usuarios 'tienen que renovarlo después de que expire'. La renovación es un aspecto importante de la gestión continua de la seguridad del dominio.

En el contexto más amplio de asegurar el dominio personalizado con SSL/TLS, la obtención de un certificado con estas características es la culminación de los pasos de instalación y configuración de Certbot. Una vez que se confirman estos detalles, significa que:

- El servidor 'será redirigido de HTTP a HTTPS'.
- Se mostrará un 'candado verde' en el navegador, indicando que la conexión es segura.
- Toda la comunicación entre el cliente y el servidor tendrá una 'comunicación encriptada de extremo a extremo'.

Estos detalles son cruciales para el cumplimiento de los objetivos de la etapa 5 del curso, que se centra en proteger un servidor HTTP IoT y la autenticación de usuarios, donde la seguridad en Internet y la criptografía, incluyendo los protocolos SSL/TLS (HTTPS), son temas clave.

RESULTADO: REDIRECCIóN A HTTPS CON CANDADO VERDE (COMUNICACIóN ENCRIPTADA DE EXTREMO A EXTREMO)

En el contexto más amplio de Asegurar un Dominio Personalizado con SSL/TLS, se detallan que el 'Resultado: Redirección a HTTPS con Candado Verde (Comunicación Encriptada de Extremo a Extremo)' es la culminación exitosa de todo el proceso de configuración del certificado SSL/TLS con Certbot y Let's Encrypt. Este resultado indica que el dominio está completamente protegido y ofrece una experiencia segura al usuario. A continuación, se desglosa lo que se dice sobre este resultado:

- 1. Redirección de HTTP a HTTPS:
  - Una vez que el dominio está asegurado con Let's Encrypt, el servidor 'será redirigido de HTTP a HTTPS'. Esto ocurre automáticamente después de la configuración exitosa del certificado.
  - Durante el proceso de configuración con Certbot, se le pregunta al usuario si desea 'redirigir nuestro tráfico HTTP a HTTPS'. Se enfatiza que 'definitivamente queremos eso' y se debe seleccionar la opción correspondiente (escribiendo '2'). Esta elección es crucial para asegurar que todas las solicitudes sean manejadas por el protocolo seguro.
- 2. Candado Verde en el Navegador:
  - Como resultado de la redirección a HTTPS, el navegador mostrará un 'candado verde'.
  - Este candado verde es una indicación visual para el usuario de que la conexión es 'segura'. Es un símbolo universalmente reconocido de confianza y seguridad en la navegación web.
- 3. Comunicación Encriptada de Extremo a Extremo:
  - Se afirma que, una vez que el sitio web ha sido completamente asegurado, 'todo cliente conectado con este servidor tendrá una comunicación encriptada de extremo a extremo'.
  - Esto significa que todos los datos intercambiados entre el navegador del cliente y el servidor están protegidos mediante criptografía, impidiendo que terceros intercepten o lean la información. La seguridad en Internet y la criptografía, incluyendo los protocolos SSL/TLS (HTTPS), son temas clave del curso y esenciales para proteger un servidor HTTP IoT.

En resumen, la redirección automática de HTTP a HTTPS, la aparición del candado verde en el navegador y la garantía de una comunicación encriptada de extremo a extremo son los indicadores más importantes del éxito en el proceso de asegurar un dominio personalizado con SSL/TLS. Estos resultados no solo validan la configuración técnica realizada, sino que también establece un canal de comunicación seguro y confiable, fundamental para la seguridad de un servidor IoT y la autenticación de usuarios.

Capítulo 6

REGLAS PARA USUARIOS ADMINISTRADORES Y NO ADMINISTRADORES

En el contexto del proyecto 'Internet de las Cosas con Python y Raspberry Pi ', la creación de Reglas para Usuarios Administradores y No Administradores es una parte fundamental para desarrollar una plataforma IoT en la nube robusta, segura y multiusuario. Esta funcionalidad, abordada en la etapa 5 y 6 del proyecto, permite que múltiples usuarios inicien sesión de forma segura y controlen o monitoreen sus dispositivos autorizados en tiempo real.

Se detalla cómo se implementan estas reglas y la gestión de acceso:

- 1. Concepto y Objetivo de los Roles de Usuario
- El objetivo principal es establecer un sistema donde los usuarios puedan tener roles de administrador o no administrador , con diferentes niveles de acceso y control sobre la plataforma y los dispositivos.
    - Plataforma Multi-Usuario: El proyecto se enfoca en construir una plataforma en la nube sin satélites donde múltiples usuarios pueden interactuar de forma segura con sus dispositivos IoT.
    - Control y Monitoreo Autorizado: Los usuarios pueden controlar y monitorear sus dispositivos, pero solo aquellos a los que están autorizados.
    - Gestión de Permisos en Tiempo Real: Los administradores pueden otorgar permisos de lectura y escritura en tiempo real a usuarios no administradores y dispositivos.
- 2. Panel de Control del Administrador
- Los usuarios con rol de administrador tienen acceso a un panel de control específico que les permite gestionar otros usuarios y sus permisos.
    - Lista de Usuarios en Línea: El panel de control del administrador muestra una lista de todos los usuarios que están actualmente en línea .
    - Botones de Conmutación para Permisos: Junto al nombre de cada usuario en línea, el panel incluye botones de conmutación (switch buttons) para otorgar permisos de lectura y escritura .
    - Botón 'Apply ' (Aplicar): Hay un botón para aplicar los cambios realizados en los permisos de los usuarios.
- 3. Implementación de la Lógica en el Servidor (Python/Flask)
- La gestión de usuarios y la preparación de los datos para el panel de control se manejan en el servidor Flask.
    - Envío de Detalles Adicionales: Al retornar la página web principal al usuario, el servidor también envía detalles adicionales como el userID de la sesión y una lista de 'online-user-records'.
    - 'online-user-records': Esta variable es un mapa (o lista de listas) que se rellena en una función como 'get-all-logged-in-users'. Cada entrada contiene:

        - El nombre del usuario (índice 0).
        - El ID del usuario (índice 1).
        - El estado de acceso de lectura (1 para marcado, 0 para desmarcado) (índice 2).
        - El estado de acceso de escritura (1 para marcado, 0 para desmarcado) (índice 3).
    - Conversión a 'checked' o 'unchecked ': Los valores numéricos (1 o 0) para los permisos de lectura y escritura se convierten en las cadenas 'checked' o 'unchecked ' respectivamente, ya que el código HTML las lee para establecer el estado de los botones de conmutación.

- 4. Implementación en el Cliente (HTML/Jinja2) y Visibilidad Condicional
- La interfaz de usuario para la gestión de permisos se construye utilizando plantillas HTML y Jinja2, con una lógica para controlar quién puede verla.
    - Bucle para Usuarios: La página 'index.html' utiliza un bucle 'for' (plantilla Jinja2) para iterar sobre la lista 'online-user-records' y crear una fila de tabla ('<li>') para cada usuario en línea.
    - Display de Datos: Cada fila muestra el nombre del usuario y los botones de conmutación para lectura y escritura, cuyos estados (checked/unchecked) se establecen dinámicamente según los datos recibidos del servidor.
    - Panel de Control Solo para Administradores: El panel de control completo para gestionar permisos solo es visible para los usuarios administradores . Esto se logra mediante una declaración 'if' en el código HTML que compara el 'userID' del usuario actual con el 'userID' de un administrador (previamente codificado o verificado). Un usuario no administrador, como 'Anam Chaudhary ' en el ejemplo, no verá este panel.
- 5. Otorgamiento de Permisos en Tiempo Real
- El proceso para que un administrador otorgue o revoque permisos se realiza en tiempo real, involucrando el cliente y el servidor, y utilizando un gestor de acceso.
    - Lado del Cliente (JavaScript en 'main.js'):
        - Un método en 'main.js' escucha los eventos de los botones de conmutación.
        - Cuando se activa un botón, extrae el ID del usuario, el estado de lectura y el estado de escritura.
        - Luego, envía una solicitud POST al servidor con el formato 'grant-user ID-read state-write state'.
    - Lado del Servidor (Aplicación Flask):
        - El servidor tiene un endpoint ('/grant') para recibir estas solicitudes POST.
        - Verificación de Administrador: Primero, el servidor verifica si la solicitud proviene de un usuario administrador . Si no es así, deniega el acceso.
        - Almacenamiento en Base de Datos: Si la solicitud es válida, el servidor almacena los permisos de lectura y escritura del usuario en la base de datos.
        - PubNub Access Manager: Finalmente, se realiza una llamada al servidor PubNub para otorgar estos permisos al usuario específico .
    - Generación de Clave de Autorización: Es importante destacar que el primer paso antes de conceder permisos de lectura y escritura es generar una clave de autorización para ese usuario específico y almacenarla en la base de datos .

En resumen, se describen un sistema de seguridad y gestión de usuarios robusto que utiliza certificados SSL/TLS para asegurar la comunicación, y una arquitectura de roles (administrador/no administrador) para controlar el acceso y los permisos a dispositivos, implementada con Flask en el servidor, HTML/Jinja2 y JavaScript en el cliente, y la gestión de acceso en tiempo real a través de PubNub.

6.1. Mejoras del Servidor IoT

Se describe un conjunto de mejoras significativas para el servidor IoT , centradas en la implementación de reglas robustas para usuarios administradores y no administradores . Estas mejoras constituyen una parte esencial de la construcción de una plataforma IoT segura y escalable, permitiendo a los administradores gestionar los permisos de otros usuarios y dispositivos en tiempo real.

Aquí se dan los detalles de estas mejoras y la gestión de usuarios:

- 1. Contexto General y Objetivos de las Mejoras
  - La etapa 6 del proyecto se dedica a implementar una 'manera segura ' en la que usuarios y dispositivos IoT pueden conectarse al servidor.
  - El objetivo es añadir funcionalidades mejoradas al servidor IoT, comenzando por la creación de reglas para usuarios administradores y no administradores.
  - Se busca desarrollar un 'ecosistema IoT fuerte, seguro, en tiempo real y escalable ' .
- 2. Dashboard de Administración y Gestión de Usuarios
  - Se introduce una nueva etapa en el dashboard (panel de control) del servidor donde se listan todos los usuarios en línea .
  - Al lado del nombre de cada usuario en línea, el dashboard presenta botones para conceder permisos de lectura y escritura, junto con un botón para 'aplicar los cambios '.
  - El dashboard para usuarios administradores mostrará una lista de todos los usuarios en línea con estos botones de control para conceder o denegar permisos.
- 3. Implementación de Permisos de Lectura y Escritura
  - Inicialmente, el código HTML puede 'hard-codear ' (codificar de forma rígida) la información del usuario, pero se implementa una funcionalidad para poblar dinámicamente esta lista desde el servidor.
  - El servidor envía detalles adicionales a la página web principal, como el 'user ID' y una lista de usuarios en línea .
  - Para cada usuario, el servidor devuelve un registro que incluye el nombre, el 'user ID', y el estado de los permisos de lectura ('read') y escritura ('write').
  - Estos estados de lectura y escritura se convierten a 'checked' o 'unchecked ' (marcado o desmarcado) para reflejarse correctamente en los botones de conmutación (switch buttons) en la interfaz HTML.
  - La página 'index.html' utiliza plantillas Jinja para iterar sobre la lista de usuarios en línea ('online users record') y mostrar múltiples filas, cada una con el nombre del usuario, su ID, y los estados de los permisos de lectura y escritura.
- 4. Restricción del Panel de Control para Usuarios Administradores
  - Una mejora crucial es asegurar que el panel de control completo (que permite cambiar permisos) sea visible 'únicamente para los usuarios administradores' .
  - Esto se logra añadiendo una sentencia 'if' en el código HTML antes de que comience el panel de control, verificando si el 'user ID' del usuario actual coincide con un 'user ID' de administrador codificado.
  - Se demuestra que un usuario no administrador (por ejemplo, 'Anam Chaudhary ') no tiene acceso al panel de control, mientras que el usuario administrador sí puede ver la lista completa de usuarios en línea y sus controles.
- 5. Lógica del Servidor para la Concesión de Permisos
  - Cuando el botón 'aplicar ' es presionado en el dashboard, una solicitud es enviada desde el código JavaScript del cliente a la aplicación Flask en el servidor.
  - El código JavaScript ('main.js') detecta la pulsación de cualquier botón de conmutación cuyo ID comience con 'XS ' (acceso), extrae el ID del usuario y el estado de los interruptores de lectura y escritura.
  - Esta información se envía al servidor como una solicitud 'POST' con el formato 'grant-[user ID]-[read state]-[write state]'.
  - En la aplicación Flask del servidor, se añade un punto final (endpoint) para recibir esta solicitud.
  - Es fundamental verificar que la solicitud provenga de un usuario administrador antes de procesarla. Si no es así, el servidor envía una respuesta de 'acceso denegado '.

- Si la solicitud es válida y proviene de un administrador, el servidor procede a:
  - 1. Almacenar los permisos de lectura y escritura del usuario en la base de datos.
  - 2. Llamar al servidor PubNub para otorgar acceso de lectura y escritura en tiempo real a ese usuario específico.
- Esta concesión de permisos es el segundo paso en un proceso que incluye primero la generación de una clave de autorización para el usuario y su almacenamiento en la base de datos.
- El cliente recibe la respuesta del servidor, extrae el JSON y, si el 'acceso es concedido ', se vuelve a suscribir al canal, y se modifica el método de suscripción para confirmar el éxito de la operación.

En resumen, las mejoras del servidor IoT en el contexto de reglas para usuarios administradores y no administradores se centran en la creación de un panel de control robusto y seguro . Este panel permite a los administradores visualizar y gestionar los permisos de lectura y escritura de otros usuarios y dispositivos en tiempo real, con una validación estricta en el lado del servidor para garantizar que solo los administradores puedan realizar cambios y que la comunicación sea segura de extremo a extremo.

6.2. Iinterfaz de Usuario (index.html)

La interfaz de usuario, específicamente el archivo 'index.html' , juega un papel fundamental en la implementación de las mejoras del servidor IoT y en la aplicación de reglas para usuarios administradores y no administradores . Este archivo es la base del dashboard que permite visualizar y gestionar los permisos de los usuarios en la plataforma IoT.

Aquí se dan los detalles de 'index.html' en este contexto:

- 1. Diseño del Dashboard para la Gestión de Usuarios
  - El 'index.html' se mejora para incluir una nueva etapa debajo de la existente (que controla el movimiento y el zumbador).
  - Esta nueva etapa está dedicada a mostrar una lista de todos los usuarios en línea .
  - Para cada usuario en línea, el 'index.html' presenta:
    - El nombre del usuario .
    - Dos botones tipo 'switch ' (conmutadores), uno para permisos de lectura y otro para permisos de escritura . Estos botones tienen IDs como 'read user ID' y 'write user ID', donde 'user ID' se reemplazará por el ID real del usuario.
    - Un botón 'apply ' (aplicar) para guardar los cambios en los permisos. Este también tendrá un ID que incluye el 'user ID' del usuario correspondiente, como 'access user ID'.
  - Inicialmente, los botones de conmutación se establecen como 'checked' (activados) en el código HTML.
- 2. Población Dinámica de la Lista de Usuarios
  - Aunque inicialmente se puede 'hard-codear ' (codificar de forma rígida) un nombre de usuario de ejemplo en el HTML, el objetivo es poblar dinámicamente esta lista desde el servidor.
  - El servidor envía detalles adicionales a la página web principal ('index.html'), incluyendo el 'user ID' del usuario actual y una lista de registros de usuarios en línea ('online users record').
  - Cada registro de usuario incluye el nombre del usuario (índice 0), su 'user ID' (índice 1), y el estado de los permisos de lectura (índice 2) y escritura (índice 3).
  - El servidor convierte el estado numérico de los permisos de lectura y escritura (por ejemplo, 1 para permitido) a las cadenas 'checked' o 'unchecked ', que el HTML interpreta para configurar el estado de los botones de conmutación.
  - El 'index.html' utiliza plantillas Jinja con un bucle 'for' ('for n in online users record') para iterar sobre la lista de usuarios en línea y generar múltiples filas , una para cada usuario. Cada fila muestra el nombre del usuario, su ID y los estados correctos de los permisos de lectura y escritura.
- 3. Restricción del Panel de Control para Usuarios Administradores

- Una mejora crucial es asegurar que el panel de control completo (que contiene la lista de usuarios y los botones de permisos) sea visible 'únicamente para los usuarios administradores ' .
- Esto se implementa añadiendo una sentencia 'if' directamente en el código HTML del 'index.html' antes de que comience el panel de control. Esta condición verifica si el 'user ID' del usuario actual coincide con un 'user ID' de administrador predefinido.
- De esta manera, un usuario no administrador (como 'Anam Chaudhary ') no tendrá acceso visual a este panel de control, mientras que el administrador sí lo verá y podrá gestionar los permisos.

- 4. Interacción del Usuario y Envío de Solicitudes
  - Cuando un usuario administrador interactúa con los botones de conmutación o presiona el botón 'apply ' en el 'index.html', el código JavaScript ('main.js') del cliente detecta estos eventos.
  - El JavaScript extrae el ID del usuario afectado y el estado actual (activado/desactivado) de los interruptores de lectura y escritura.
  - Esta información se envía al servidor como una solicitud 'POST' en un formato específico: 'grant-[user ID]-[read state]-[write state]'.
  - Esta interacción es fundamental para que los administradores puedan otorgar acceso de lectura y escritura en tiempo real a usuarios específicos, lo que luego se procesa en el servidor y se comunica a servicios como PubNub.

En síntesis, el archivo 'index.html' es el componente visual y de interacción clave del dashboard de administración, diseñado para ser dinámico, seguro y funcional. Permite a los administradores gestionar los permisos de otros usuarios de manera intuitiva, mientras que las reglas integradas en su estructura (mediante sentencias 'if' y Jinja) garantizan que solo los usuarios autorizados puedan acceder y manipular estas funcionalidades críticas en el servidor IoT.

6.2.1. etapa para Usuarios ONLINE

En el contexto más amplio de la Interfaz de Usuario (index.html), se detallan que la 'etapa para Usuarios Online' es una característica clave del panel de control del servidor IoT. Esta etapa está diseñada para mostrar una lista de los usuarios actualmente conectados y permitir a los administradores gestionar sus permisos en tiempo real. A continuación, se desglosa lo que se dice sobre esta etapa:

- 1. Propósito y Ubicación en el Dashboard:
  - Esta etapa es una nueva adición al dashboard.
  - Su objetivo es mostrar una lista de todos los usuarios online.
  - Es parte de las características mejoradas que se están desarrollando para el servidor IoT, específicamente para crear reglas para usuarios administradores y no administradores.

- 2. Estructura y Contenido Visual (index.html):
  - Se implementa en el archivo index.html, debajo de una etapa existente (como la del detector de movimiento y el botón para controlar el zumbador).
  - Se utiliza un panel primario de Bootstrap con el encabezado 'online users' (usuarios online).
  - Cada fila de la tabla de usuarios se define como un elemento de lista (<li>).
  - A la izquierda de cada fila, se muestra el nombre del usuario.
  - A la derecha, se incluyen dos botones tipo 'switch' y un botón de 'aplicar' (apply):
    - Un switch para otorgar permisos de lectura (read permissions).
    - Otro switch para otorgar permisos de escritura (write permissions).
    - Un botón 'apply' para aplicar los cambios.
  - Los switches tienen IDs como read user ID, write user ID, y el botón de aplicar tiene un ID como access user ID. Estos user ID se reemplazan dinámicamente con los IDs de usuario reales proporcionados por el servidor.

- Inicialmente, ambos switches de lectura y escritura están configurados como 'checked' (activados).

- 3. Población de Datos desde el Servidor:
  - Se requiere una funcionalidad en el lado del servidor para proporcionar el nombre de usuario y el ID de usuario para poblar correctamente esta lista.
  - La función get all logged in users en el servidor devuelve un mapa o lista (online user records) que contiene detalles de los usuarios online.
  - Cada registro incluye el nombre de usuario (índice 0), el ID de usuario (índice 1), el estado de lectura (índice 2) y el estado de escritura (índice 3).
  - El servidor convierte el estado de acceso (1 o 0) a 'checked' o 'unchecked' para que el código HTML lo interprete y establezca el estado de los botones switch.
  - En index.html, se utiliza Jinja templates y un bucle for (for n in online users record) para iterar sobre la lista de usuarios online y generar dinámicamente múltiples filas (<li>).

- 4. Control de Visibilidad (Solo para Administradores):
  - Se enfatiza que este 'panel de control' (incluyendo la etapa de usuarios online) debe ser visible solo para usuarios administradores.
  - Esto se implementa añadiendo una sentencia if en el código HTML que verifica si el user ID del usuario actual (obtenido de session user ID) coincide con el ID de usuario del administrador.
  - Se realiza una prueba donde un usuario no administrador (Anam Chaudhary) no tiene acceso a este panel, mientras que el administrador sí lo ve y puede observar a otros usuarios en la lista.

En resumen, la etapa para Usuarios Online en index.html es una característica dinámica y controlada por el lado del servidor, diseñada para proporcionar a los administradores una vista en tiempo real de los usuarios conectados y la capacidad de gestionar sus permisos de lectura y escritura.

6.2.2. Lista de Usuarios ON-LINE

En el contexto más amplio de la Interfaz de Usuario (index.html), se describen la 'Lista de Usuarios Online' como una característica mejorada y crucial del panel de control del servidor IoT. Esta lista está diseñada para mostrar a los usuarios actualmente conectados y, específicamente para los administradores, permitirles gestionar los permisos de lectura y escritura de esos usuarios en tiempo real. A continuación, se detalla lo que se dice sobre la Lista de Usuarios Online:

- 1. Propósito y Ubicación:
  - Es una nueva etapa añadida al dashboard del servidor IoT.
  - Su objetivo es mostrar una lista de todos los usuarios online.
  - Forma parte del desarrollo de características mejoradas para crear reglas para usuarios administradores y no administradores.
  - Para los usuarios administradores, el dashboard mostrará una lista de todos los usuarios online con botones de interruptor para otorgar permisos de lectura y escritura.

- 2. Implementación en index.html:
  - Se añade debajo de una etapa existente (como la del detector de movimiento y el botón para controlar el zumbador) en el archivo index.html.
  - Visualmente, utiliza un panel primario de Bootstrap con el encabezado 'online users'.
  - Cada usuario online se representa como una fila en una tabla, definida como un elemento de lista (<li>) dentro de una clase de lista de Bootstrap.
  - Contenido de cada fila:
    - En el extremo izquierdo, se muestra el nombre del usuario.
    - En el extremo derecho, hay dos botones tipo 'switch' y un botón de 'aplicar' (apply).
    - Los switches son para otorgar permisos de lectura (read permissions) y permisos de escritura (write permissions).

    - Los IDs de los switches (por ejemplo, read user ID, write user ID) y del botón 'apply' (access user ID) son dinámicamente reemplazados por el ID de usuario real proporcionado por el servidor para ese usuario online particular.
    - Inicialmente, ambos switches de lectura y escritura están configurados como 'checked' (activados).

- 3. Población Dinámica de Datos desde el Servidor:
    - Para poblar correctamente esta lista, el servidor necesita proporcionar el nombre de usuario y el ID de usuario.
    - Una función get all logged in users en el lado del servidor se encarga de esto, devolviendo un mapa o lista (online user records) que contiene detalles de los usuarios online.
    - Cada registro en esta lista incluye: el nombre de usuario (índice 0), el ID de usuario (índice 1), el estado de lectura (índice 2) y el estado de escritura (índice 3).
    - El servidor convierte el estado de acceso (1 o 0) en 'checked' o 'unchecked' para que el código HTML lo interprete y establezca el estado inicial de los switches de lectura y escritura.
    - En index.html, se utilizan Jinja templates y un bucle for (for n in online users record) para iterar sobre la lista de usuarios online y generar dinámicamente las múltiples filas (<li>).
- 4. Control de Visibilidad para Administradores:
    - Es fundamental que este panel de control, incluyendo la lista de usuarios online, sea visible solo para usuarios administradores.
    - Esto se implementa añadiendo una sentencia if en el código HTML antes de que comience el panel de control. Esta condición verifica si el user ID del usuario actual (obtenido de session user ID) coincide con el ID de usuario del administrador (que puede ser codificado en el HTML).
    - Se demuestra con una prueba: un usuario no administrador (Anam Chaudhary) no tiene acceso a este panel, mientras que un administrador sí lo ve y puede observar a otros usuarios en la lista.
- 5. Funcionalidad del Botón 'Aplicar':
    - Se añade un endpoint para el botón 'apply' que enviará una solicitud desde el código JavaScript a la aplicación Flask.
    - Un método en main.js escucha los cambios en los switches, extrae el ID de usuario, y lee el estado de los switches de lectura y escritura. Luego, envía una solicitud de tipo POST al servidor con el formato grant - user ID - read state - write state.
    - En la aplicación Flask, se añade un endpoint para recibir esta solicitud, se verifica si el remitente es un usuario administrador y, si todo va bien, se almacena los permisos de lectura y escritura en la base de datos y se llama al servidor PubNub para conceder acceso a ese usuario específico. La generación de la clave de autorización para el usuario específico y su almacenamiento en la base de datos es un paso previo a la concesión de permisos.

En conclusión, la 'Lista de Usuarios Online' en index.html es una característica interactiva y dinámica que permite a los administradores de un servidor IoT visualizar y modificar en tiempo real los permisos de lectura y escritura de los usuarios conectados, asegurando que la gestión de accesos sea eficiente y controlada.

6.2.3. Botones Tipo 'SWITCH ' Para Permisos de Lectura y Escritura

En el contexto más amplio de la Interfaz de Usuario (index.html), se describen los 'Botones de Permisos de Lectura/Escritura por Usuario' como una característica interactiva y esencial dentro de la etapa de usuarios online del panel de control del servidor IoT. Estos botones están diseñados para permitir a los usuarios administradores gestionar en tiempo real los niveles de acceso de otros usuarios conectados. A continuación, se detalla lo que se dice sobre estos botones:

- 1. Propósito y Ubicación en la Interfaz:

- Estos botones son parte de una nueva etapa en el dashboard del servidor IoT, específicamente en la 'Lista de Usuarios Online'.
- Su objetivo es permitir a los administradores otorgar permisos de lectura y escritura a cada usuario online.
- Para cada usuario listado, los botones se ubican a la derecha del nombre del usuario.
- Esta funcionalidad es una característica mejorada para crear reglas específicas para usuarios administradores y no administradores.

- 2. Descripción y Estructura en index.html:
  - Los botones son visualmente 'botones tipo switch' (interruptores).
  - Se implementan en el archivo index.html dentro de un panel primario de Bootstrap.
  - Para cada usuario online, se genera un elemento de lista (<li>) que contiene:
    - Un switch para permisos de lectura (read permissions).
    - Otro switch para permisos de escritura (write permissions).
    - Un botón adicional de 'aplicar' (apply) para guardar los cambios.
  - Los switches y el botón de aplicar tienen IDs dinámicos (por ejemplo, read user ID, write user ID, access user ID), donde user ID se reemplaza con el ID real del usuario proporcionado por el servidor.
  - Inicialmente, ambos switches de lectura y escritura están configurados como 'checked' (activados).
- 3. Población Dinámica y Estado Inicial:
  - El servidor juega un papel crucial al proporcionar los datos para estos botones.
  - Una función get all logged in users en el servidor devuelve una lista o mapa (online user records) que incluye el estado de lectura (índice 2) y el estado de escritura (índice 3) para cada usuario.
  - El servidor convierte internamente estos estados (1 o 0) en las cadenas 'checked' o 'unchecked' para que el código HTML (usando Jinja templates) pueda interpretarlos y establecer el estado inicial de los switches correctamente.
  - Mediante un bucle for (for n in online users record) en index.html, se generan dinámicamente las filas de usuarios con sus respectivos botones y estados iniciales.
- 4. Funcionalidad del 'Botón de Aplicar' y Lógica del Servidor:
  - Existe un endpoint específico para el botón 'apply'.
  - Cuando un switch es modificado y se presiona 'apply', una función en main.js escucha estos cambios, extrae el ID de usuario y el estado de los switches de lectura y escritura.
  - Luego, envía una solicitud POST al servidor con el formato grant - user ID - read state - write state.
  - En la aplicación Flask, un endpoint recibe esta solicitud y realiza varias acciones:
  - Verifica si la solicitud proviene de un usuario administrador.
  - Almacena los permisos de lectura y escritura del usuario en la base de datos.
  - Llama al servidor PubNub para conceder acceso a ese usuario específico.
  - Antes de conceder los permisos, se debe generar una clave de autorización para el usuario específico y almacenarla en la base de datos.
- 5. Control de Visibilidad (Solo para Administradores):
  - Es fundamental que este panel de control y sus botones de permisos sean visibles solo para usuarios administradores.
  - Esto se implementa con una sentencia if en index.html que verifica si el user ID del usuario actual (obtenido de session user ID) coincide con el ID del administrador. Los usuarios no administradores no verán estos controles.

En conclusión, los botones de permisos de lectura/escritura por usuario en index.html son una parte interactiva y programática del panel de control del servidor IoT. Su existencia permite a los administradores una gestión granular y en tiempo real de los accesos de los usuarios, lo cual es fundamental para la seguridad y el control de un ecosistema IoT escalable.

6.2.4. Botón 'APPLY' Para Aplicar Cambios

En el contexto de la Interfaz de Usuario, específicamente el archivo index.html, el botón 'Aplicar Cambios' es un elemento crucial dentro del panel de control para usuarios administradores. Aquí se detalla lo que se dice sobre este botón:

- Contexto en index.html y la Interfaz de Usuario:
  - El panel de control, visible solo para usuarios administradores, incluye una etapa para listar a los usuarios en línea.
  - Para cada usuario en línea, se muestran dos botones de tipo interruptor: uno para otorgar permisos de lectura y otro para permisos de escritura.
  - Junto a estos interruptores, se encuentra un tercer botón, descrito como un 'botón adjunto' o 'botón táctil' con un ID como access user ID, cuya función es aplicar los cambios realizados en los permisos de ese usuario específico.
  - La estructura del index.html para esta etapa utiliza plantillas Jinja para poblar dinámicamente la lista de usuarios, mostrando el nombre del usuario (índice 0), el ID del usuario (índice 1), y los estados de lectura y escritura (índices 2 y 3, respectivamente) que determinan si los interruptores están 'checked' o 'unchecked'.
- Funcionalidad del Botón 'Aplicar Cambios':
  - Cuando se presiona el botón, el código JavaScript en el lado del cliente envía una solicitud a la aplicación Flask en el servidor.
  - Esta solicitud, que se realiza como una petición POST (similar a send event), incluye el ID del usuario, el estado de los permisos de lectura y el estado de los permisos de escritura que se desean aplicar (por ejemplo, `grant - user ID - read state - write stick`).
  - En el lado del servidor, la aplicación Flask recibe esta solicitud a través de un endpoint específico. Es fundamental que la aplicación verifique que la solicitud proviene de un usuario administrador antes de proceder.
  - Si la verificación es exitosa, el servidor realiza dos acciones principales:
    - 1. Almacena los nuevos permisos de lectura y escritura del usuario en la base de datos.
    - 2. Llama al servidor PubNub para otorgar acceso de lectura y escritura a ese usuario específico en tiempo real.
  - Cabe mencionar que un paso previo a la concesión de estos permisos es la generación de una clave de autorización para el usuario en cuestión, la cual se almacena en la base de datos.
  - El panel permite a los administradores 'otorgar acceso de lectura y escritura en tiempo real a todos los usuarios que no son administradores y a los dispositivos'. Al seleccionar el botón 'Aplicar', 'los permisos de acceso de lectura y escritura de ese usuario específico se cambiarán en consecuencia en tiempo real'.

6.2.5. Extructura del Código HTML (usando Bootstrap, divs)

En el contexto más amplio de la Interfaz de Usuario, el archivo index.html es fundamental para el panel de control del servidor IoT, especialmente para los usuarios administradores. Las fuentes detallan una estructura de código HTML que utiliza divs y componentes de Bootstrap para organizar y presentar la información y las funcionalidades. Aquí se desglosa la estructura del código HTML, con énfasis en el uso de divs y Bootstrap:

- 1. Estructura General y Diseño:
  - El dashboard busca tener diferentes secciones, una de las cuales es para la lista de usuarios en línea.
  - Se menciona la replicación de una 'etapa de caja negra' existente, lo que sugiere un diseño modular donde diferentes funcionalidades están encapsuladas en bloques.
  - Se utilizan divs para organizar el contenido, incluyendo un div completo para una etapa, un 'inner div' para definir una fila, y otro div para centrar los elementos ('define everything as a central block').

- 2. Uso de Bootstrap para el Panel de Usuarios en Línea:
  - Para la etapa que lista a los usuarios en línea, se crea un 'Bootstrap primary panel'. Esto indica que se están utilizando las clases y estilos predefinidos de Bootstrap para dar una apariencia coherente y profesional al panel.
  - Dentro de este panel, se añade un encabezado para identificar la etapa como 'online users'.
- 3. Visualización de Usuarios y Permisos:
  - La lista de usuarios se presenta mediante una tabla cuya clase es list group, y cada fila de la tabla se define como un elemento li (lista). Esto sugiere el uso de componentes de lista de Bootstrap, que estilizan elementos ul y li para crear listas de elementos conectados.
  - Cada fila (li) está diseñada para mostrar:
    - El nombre del usuario en el extremo izquierdo.
    - Dos botones de tipo interruptor ('switch buttons') en el extremo derecho para otorgar permisos de lectura (read user ID) y escritura (write user ID). Estos botones se inicializan como 'checked' (activos).
    - Un tercer botón adyacente (conocido como 'touch button' o 'botón táctil') con un ID como access user ID, cuya función es aplicar los cambios de permisos para ese usuario específico.
- 4. Población Dinámica con Plantillas Jinja:
  - El index.html utiliza plantillas Jinja para poblar dinámicamente la lista de usuarios en línea.
  - Esto se logra mediante un bucle for (for n in online users record) que itera sobre una lista de registros de usuarios recibida del servidor.
  - Dentro de cada iteración del bucle, se accede a los datos de cada usuario:
    - El nombre del usuario se encuentra en el índice 0 de la lista.
    - El ID del usuario se encuentra en el índice 1.
    - Los estados de lectura y escritura (que determinan si los interruptores están 'checked' o 'unchecked') se encuentran en los índices 2 y 3, respectivamente.
- 5. Visibilidad Condicional para Administradores:
  - Para asegurar que el panel de control completo (que incluye la lista de usuarios en línea y la funcionalidad del botón 'Aplicar Cambios') sea visible solo para los usuarios administradores, se implementa una sentencia if en el código HTML. Esta condición verifica si el user ID del usuario actual coincide con el ID de un usuario administrador.

En resumen, el index.html se estructura como un dashboard interactivo y dinámico para administradores, utilizando una combinación de divs para la organización modular y Bootstrap para el estilado de paneles y listas. La integración de Jinja permite la presentación de datos en tiempo real de los usuarios en línea y sus permisos, mientras que la lógica condicional en el HTML controla la visibilidad de estos elementos solo para usuarios autorizados.

6.2.6. Elementos de la Fila de Usuario

En el contexto de la Interfaz de Usuario (UI), específicamente dentro del archivo index.html y como parte del panel de control para administradores, los elementos de la fila de usuario son componentes diseñados para mostrar la información de los usuarios en línea y permitir a los administradores gestionar sus permisos en tiempo real. Aquí se detallan los elementos que conforman cada fila de usuario:

- 1. Estructura General de la Fila:
  - Las filas de usuario forman parte de un 'Bootstrap primary panel' con el encabezado 'online users'.
  - Cada fila en la tabla se define como un elemento li (lista), utilizando la clase list group de Bootstrap.
  - La población de estas filas es dinámica, utilizando plantillas Jinja con un bucle for n in online users record, donde n representa cada registro de usuario en línea.

- 2. Nombre del Usuario:
  - En el extremo izquierdo de la fila, se muestra el nombre del usuario.
  - Este dato se obtiene del índice 0 de la lista n que representa al usuario actual en el bucle Jinja.
- 3. Botones de Tipo Interruptor para Permisos:
  - En el extremo derecho de la fila, se encuentran dos botones de tipo interruptor ('switch buttons').
  - Uno es para otorgar permisos de lectura, con un ID como read user ID.
  - El otro es para otorgar permisos de escritura, con un ID como write user ID.
  - Ambos interruptores se configuran inicialmente como 'checked' (activos).
  - El estado ('checked' o 'unchecked') de estos interruptores se determina a partir del índice 2 para lectura y el índice 3 para escritura del registro de usuario (n).
- 4. Botón 'Aplicar Cambios':
  - Junto a los interruptores, hay un tercer botón, descrito como un 'touch button' (botón táctil) con un ID como access user ID.
  - La función principal de este botón es aplicar los cambios realizados en los permisos de lectura y escritura para ese usuario específico.
  - Cuando se presiona, el código JavaScript envía una solicitud POST a la aplicación Flask en el servidor. Esta solicitud incluye el ID del usuario, el estado de los permisos de lectura y el estado de los permisos de escritura.
  - El ID del usuario para estos botones se extrae del índice 1 del registro de usuario (n) y reemplaza el placeholder user ID en los IDs de los botones.
  - Una vez que el servidor procesa la solicitud, almacena los nuevos permisos en la base de datos y llama al servidor PubNub para otorgar o revocar el acceso en tiempo real. Este proceso también implica la generación de una clave de autorización para el usuario.
- 5. Visibilidad Condicional:
  - Es importante destacar que todo el panel de control, que incluye estas filas de usuario y la funcionalidad asociada, es visible únicamente para los usuarios administradores. Esta restricción se implementa mediante una declaración if en el código HTML que verifica si el ID del usuario actual coincide con el ID de un administrador.

En resumen, cada fila de usuario en el index.html del panel de control representa una unidad interactiva que permite a los administradores visualizar la información de los usuarios en línea y gestionar sus permisos de lectura y escritura a través de interruptores y un botón de aplicación de cambios, todo ello con una estructura dinámica impulsada por Jinja y estilizada con Bootstrap.

NOMBRE DE USUARIO

En el contexto más amplio de los Elementos de la Fila de Usuario dentro de la Interfaz de Usuario (UI) en el index.html, el nombre del usuario es un componente clave para identificar a cada usuario en línea en el panel de control para administradores. Aquí se detalla lo que se dice sobre el nombre de usuario:

- 1. Visibilidad y Contexto:
  - El nombre del usuario es parte de la lista de usuarios en línea que se muestra en un 'Bootstrap primary panel' dentro del panel de control.
  - Este panel completo es visible exclusivamente para usuarios administradores. Una sentencia if en el código HTML asegura que solo los administradores (aquellos cuyo ID de usuario coincide con el de un administrador) puedan ver esta etapa.
- 2. Posición en la Fila de Usuario:
  - Para cada fila que representa un usuario en línea, el nombre del usuario se muestra en el extremo izquierdo.

- Las filas están estructuradas como elementos li de una tabla con la clase list group.

- 3. Población Dinámica con Jinja:
  - Inicialmente, el nombre del usuario se podría 'hardcodear' (codificar directamente) en el HTML, por ejemplo, con el nombre del desarrollador.
  - Sin embargo, la implementación final utiliza plantillas Jinja para poblar dinámicamente esta lista.
  - El servidor Flask devuelve una lista de online user records (registros de usuarios en línea) a la página index.html.
  - Dentro de un bucle for n in online users record de Jinja, cada n representa un usuario. El nombre del usuario se encuentra en el índice 0 de esta lista n.

- 4. Generación de Datos en el Servidor:
  - En el lado del servidor, la función get all logged in users (obtener todos los usuarios conectados) es responsable de crear la lista online user records.
  - Esta lista es un mapa donde el nombre del usuario se agrega en el índice 0. También incluye el ID del usuario (índice 1), el estado de lectura (índice 2) y el estado de escritura (índice 3).

En resumen, el nombre de usuario es el identificador visual principal para cada entrada en la lista de usuarios en línea del panel de control de administrador. Su presentación se realiza de manera dinámica a través de plantillas Jinja, asegurando que la información sea actualizada y que solo los usuarios con los permisos adecuados puedan acceder a ella.

BOTóN DE LECTURA

En el contexto más amplio de los Elementos de la Fila de Usuario dentro de la Interfaz de Usuario (UI) del index.html, el Botón de Lectura (`ID: read user ID`) es un componente esencial que permite a los usuarios administradores controlar los permisos de acceso de los usuarios en línea. Aquí se detalla lo que se dice sobre este botón:

- 1. Ubicación y Apariencia en la Interfaz de Usuario:
  - El botón de lectura es uno de los dos botones de tipo interruptor ('switch buttons') que se encuentran en el extremo derecho de cada fila de usuario en la lista de usuarios en línea.
  - El otro botón interruptor es para los permisos de escritura.
  - Inicialmente, ambos interruptores (lectura y escritura) se configuran como 'checked' (activos).
  - Estas filas de usuario, que contienen el botón de lectura, son parte de un panel de control más amplio visible exclusivamente para usuarios administradores. Una sentencia if en el código HTML asegura que solo los administradores puedan ver esta etapa.

- 2. Identificación Dinámica (ID: read user ID):
  - El ID del botón de lectura es read user ID.
  - La parte user ID en este ID se reemplaza dinámicamente por el ID real del usuario en línea al que corresponde la fila. Este ID de usuario es proporcionado por el servidor.

- 3. Estado y Población Dinámica:
  - El estado inicial del interruptor (si está 'checked' o 'unchecked') se determina en el lado del servidor y se envía a la plantilla Jinja en index.html.
  - En el servidor, si el read access es 1, la variable de lectura se establece como 'checked'; de lo contrario, se establece como 'unchecked'.
  - En el lado del cliente, las plantillas Jinja leen este estado para configurar el atributo 'checked' del botón interruptor.
  - Los datos de los usuarios en línea se proporcionan a la página index.html en una lista, y el estado de lectura de cada usuario se encuentra en el índice 2 de su respectivo registro.

- 4. Funcionalidad para Control de Permisos:
  - La función principal del botón de lectura es permitir a los administradores otorgar o revocar permisos de lectura a usuarios o dispositivos específicos en tiempo real.
  - Cuando se presiona el botón 'Aplicar Cambios' (adyacente a los interruptores) para un usuario específico, el código JavaScript lee el estado actual del botón de lectura (y escritura).
  - Esta información se envía como parte de una solicitud POST a la aplicación Flask en el servidor, incluyendo el estado de lectura (y escritura) junto con el ID del usuario.
  - En el servidor, antes de aplicar los cambios, se verifica que la solicitud provenga de un usuario administrador.
  - Si la verificación es exitosa, la aplicación Flask realiza dos acciones clave:
    - 1. Almacena los nuevos permisos de lectura (y escritura) en la base de datos.
    - 2. Llama al servidor PubNub para otorgar o revocar el acceso de lectura (y escritura) a ese usuario específico en tiempo real.
  - Cabe destacar que la concesión de estos permisos es un segundo paso, ya que el primer paso es generar y almacenar una clave de autorización para el usuario en la base de datos.

En síntesis, el Botón de Lectura es un elemento interactivo crucial en el panel de control de administradores, que forma parte de una fila de usuario dinámicamente generada. Permite a los administradores visualizar y modificar los permisos de lectura de los usuarios en línea de manera controlada y en tiempo real, utilizando una combinación de plantillas Jinja, JavaScript y lógica de servidor para gestionar y aplicar estos accesos.

BOTóN DE ESCRITURA

En el contexto más amplio de los Elementos de la Fila de Usuario dentro de la Interfaz de Usuario (UI) en el index.html, el Botón de Escritura (`ID: write user ID`) es un componente interactivo clave que permite a los usuarios administradores controlar los permisos de escritura de los usuarios en línea. Aquí se detalla lo que se dice sobre este botón:

- 1. Ubicación y Apariencia en la Interfaz de Usuario:
  - El botón de escritura es uno de los dos botones de tipo interruptor ('switch buttons') que se encuentran en el extremo derecho de cada fila de usuario en la lista de usuarios en línea.
  - El otro interruptor es para los permisos de lectura.
  - Inicialmente, ambos interruptores (lectura y escritura) se configuran como 'checked' (activos).
  - Las filas de usuario, que incluyen este botón, forman parte de un panel de control más amplio que es visible exclusivamente para usuarios administradores. Una sentencia if en el código HTML asegura esta visibilidad restringida.
- 2. Identificación Dinámica (ID: write user ID):
  - El ID de este botón es write user ID.
  - La parte user ID en este ID se reemplaza dinámicamente con el ID real del usuario en línea al que corresponde la fila, ya que este ID es proporcionado por el servidor.
- 3. Estado y Población Dinámica:
  - Aunque inicialmente se configuran como 'checked', el estado real del interruptor (si está 'checked' o 'unchecked') se determina en el lado del servidor y se envía a la plantilla Jinja en index.html.
  - En el servidor, si el write access es 1, la variable de escritura se establece como 'checked'; de lo contrario, se establece como 'unchecked'.
  - En el lado del cliente, las plantillas Jinja leen este estado para configurar el atributo 'checked' del botón interruptor.
  - Los datos de los usuarios en línea se proporcionan a la página index.html en una lista de online user records, y el estado de escritura de cada usuario se encuentra en el índice 3 de su respectivo registro (n).

- 4. Funcionalidad para Control de Permisos:
  - La función principal del botón de escritura es permitir a los administradores otorgar o revocar permisos de escritura a usuarios o dispositivos específicos en tiempo real.
  - Cuando se presiona el botón 'Aplicar Cambios' (adyacente a los interruptores) para un usuario específico, el código JavaScript lee el estado actual de los botones de lectura y escritura.
  - Esta información se envía como parte de una solicitud POST a la aplicación Flask en el servidor, incluyendo el ID del usuario, el estado de lectura y el estado de escritura (referido como right stick).
  - En el servidor, antes de aplicar los cambios, la aplicación Flask verifica que la solicitud provenga de un usuario administrador.
  - Si la verificación es exitosa, la aplicación Flask realiza dos acciones clave:
    - 1. Almacena los nuevos permisos de escritura (y lectura) en la base de datos.
    - 2. Llama al servidor PubNub para otorgar o revocar el acceso de escritura (y lectura) a ese usuario específico en tiempo real.
  - Es importante mencionar que la concesión de estos permisos de lectura y escritura es un segundo paso, ya que el primer paso es generar la clave de autorización para el usuario específico y almacenarla en la base de datos.

En síntesis, el Botón de Escritura es un elemento interactivo fundamental en el panel de control de administradores, formando parte de una fila de usuario generada dinámicamente. Permite a los administradores visualizar y modificar los permisos de escritura de los usuarios en línea de manera controlada y en tiempo real, integrando plantillas Jinja, JavaScript y lógica de servidor para gestionar y aplicar estos accesos de forma segura.

BOTóN APLICAR

En el contexto más amplio de los Elementos de la Fila de Usuario dentro de la Interfaz de Usuario (UI) en el index.html, el botón 'Aplicar' (`ID: access user ID`) es un componente crucial que permite a los usuarios administradores guardar y aplicar los cambios de permisos para usuarios específicos en tiempo real. Aquí se detalla lo que se dice sobre este botón:

- 1. Ubicación y Apariencia en la Interfaz de Usuario:
  - Este botón es descrito como un 'botón táctil' ('touch button').
  - Se encuentra en el extremo derecho de cada fila de usuario, adyacente a los dos botones de tipo interruptor para permisos de lectura y escritura.
  - Las filas de usuario, que incluyen este botón, son parte de un panel de control más amplio que es visible exclusivamente para usuarios administradores. La visibilidad se controla mediante una sentencia if en el código HTML que compara el ID del usuario actual con el ID de un administrador.
- 2. Identificación Dinámica (ID: access user ID):
  - El ID del botón es access user ID.
  - La parte user ID en este ID se reemplaza dinámicamente con el ID real del usuario en línea al que corresponde la fila. Este ID es proporcionado por el servidor a la plantilla index.html.
- 3. Funcionalidad principal: Aplicar Cambios de Permisos:
  - La función principal de este botón es aplicar los cambios realizados en los permisos de lectura y escritura para ese usuario específico.
  - Cuando se presiona el botón 'Aplicar' (o 'Apply'), los permisos de acceso de lectura y escritura de ese usuario en particular se modifican en consecuencia en tiempo real.
- 4. Flujo de Interacción (Cliente-Servidor):
  - En el lado del cliente (JavaScript):

- Un método en main.js 'escucha' cualquier botón que comience con access en su ID (lo que implica que se refiere a este botón 'Aplicar').
- Cuando se presiona, el JavaScript divide el ID del botón para extraer el ID del usuario.
- Luego, lee el estado actual de los interruptores de lectura y escritura asociados a ese usuario.
- Finalmente, envía una solicitud POST a la aplicación Flask en el servidor. Esta solicitud incluye el formato `grant - user ID - read state - right stick` (donde right stick se refiere al estado de escritura).

- En el lado del servidor (Aplicación Flask):
  - La aplicación Flask tiene un endpoint configurado para recibir esta solicitud de grant user ID read and write.
  - Es crucial que el servidor verifique que la solicitud proviene de un usuario administrador antes de procesarla. Si no es así, se envía una respuesta de 'acceso denegado'.
  - Si la verificación es exitosa, se realizan dos acciones clave:
    - 1. Almacenar los nuevos permisos de lectura y escritura del usuario en la base de datos.
    - 2. Llamar al servidor PubNub para otorgar o revocar el acceso de lectura y escritura a ese usuario específico en tiempo real.
  - Un paso previo a la concesión de estos permisos es la generación de una clave de autorización para el usuario específico y su almacenamiento en la base de datos.

En resumen, el botón 'Aplicar' en cada fila de usuario es el punto de interacción clave para los administradores en el index.html del panel de control. Su función es traducir la selección de permisos en la interfaz de usuario en acciones concretas en el servidor y en la plataforma de comunicación en tiempo real (PubNub), asegurando una gestión dinámica y segura de los accesos para los usuarios en línea.

6.3. Funcionalidad del Lado del Servidor (Python Flask)

La funcionalidad del lado del servidor, implementada con Python Flask, es el núcleo de la gestión de las reglas para usuarios administradores y no administradores en la plataforma IoT. El servidor Flask no solo procesa las solicitudes y proporciona la interfaz, sino que también es responsable de la seguridad, la persistencia de los datos y la comunicación en tiempo real para aplicar estas reglas.

A continuación, se dan los detalles de la funcionalidad del lado del servidor con Python Flask en este contexto:

- 1. Base de Seguridad del Servidor
  - Antes de implementar las reglas de usuario, el servidor IoT se asegura utilizando **certificados SSL/TLS de Let's Encrypt** para su dominio personalizado. Esto garantiza que toda la comunicación con el servidor sea **cifrada de extremo a extremo**, lo que es fundamental para una gestión segura de permisos de usuario. El servidor redirige el tráfico HTTP a HTTPS.
  - La instalación del software 'certbot' y la configuración de las reglas de seguridad de entrada (como el puerto HTTPS 443 en AWS EC2) son pasos previos cruciales en el lado del servidor para establecer un entorno seguro.
- 2. Suministro Dinámico de Datos a la Interfaz de Usuario ('index.html')
  - El servidor Flask es responsable de **enviar detalles adicionales a la página web principal ('index.html')**, lo que permite poblar dinámicamente el dashboard con la información de los usuarios en línea.
  - Esto incluye el 'user ID' del usuario actual (obtenido de la sesión) y una lista de 'registros de usuarios en línea' ('online users record').
  - Para cada usuario en línea, el servidor prepara un registro que contiene su nombre, 'user ID', y el **estado de sus permisos de lectura y escritura**.
  - Es crucial que el servidor convierta los estados numéricos de los permisos (por ejemplo, 1 para permitido) a las cadenas 'checked' o 'unchecked ', que el código HTML ('index.html') interpreta para configurar correctamente el estado de los botones de conmutación (switch buttons) en la interfaz. Esta lógica de conversión se realiza en el servidor antes de renderizar la plantilla Jinja.

- 3. Identificación del Usuario Administrador
  - El servidor Flask gestiona la sesión del usuario y proporciona el 'user ID' del usuario actualmente conectado al cliente.
  - Aunque la verificación de la visibilidad del panel de control se realiza en el HTML con una sentencia 'if' que compara el 'user ID' actual con un 'user ID' de administrador codificado, la información del 'user ID' proviene del servidor.
- 4. Manejo de Solicitudes de Concesión de Permisos
  - El servidor Flask implementa un punto final (endpoint) específico para recibir las solicitudes del cliente (enviadas por JavaScript) cuando un administrador intenta cambiar los permisos de otro usuario. Estas solicitudes tienen un formato como 'grant-[user ID]-[read state]-[write state]' y se envían como peticiones 'POST'.
  - Una de las funcionalidades más críticas del lado del servidor es la validación de que la solicitud para conceder o denegar permisos provenga de un usuario administrador . Si la solicitud no es de un administrador, el servidor envía una respuesta de 'acceso denegado '.
  - Si la solicitud es válida y proviene de un administrador, el servidor realiza dos acciones principales:
    - 1. Almacena los permisos de lectura y escritura del usuario afectado en la base de datos .
    - 2. Llama al servidor PubNub para otorgar acceso de lectura y escritura en tiempo real a ese usuario específico. Esto se logra suscribiendo al usuario al canal correspondiente con los permisos adecuados.
  - La concesión de permisos es el segundo paso en un proceso que requiere, como primer paso, la generación de una clave de autorización para el usuario y su almacenamiento en la base de datos.
- 5. Comunicación y Actualización en Tiempo Real
  - El servidor utiliza PubNub como protocolo de comunicación principal para la funcionalidad de acceso de los usuarios administradores, lo que permite el otorgamiento de permisos de lectura y escritura en tiempo real a usuarios no administradores y dispositivos.
  - Después de que el servidor procesa una solicitud de concesión de acceso y notifica a PubNub, el cliente (navegador del usuario) se vuelve a suscribir al canal para reflejar los nuevos permisos.

En resumen, la funcionalidad del lado del servidor con Python Flask es el cerebro detrás de la gestión de usuarios. Se encarga de la seguridad fundamental, la entrega de datos dinámicos a la interfaz de usuario, la validación de roles de administrador para acciones críticas y la ejecución de cambios de permisos que se persisten en la base de datos y se aplican en tiempo real a través de servicios como PubNub.

6.3.1. Población de la Lista de Usuarios Online

En el contexto más amplio de la funcionalidad del lado del servidor utilizando Python Flask, la **población de la lista de usuarios en línea** es una característica clave para permitir a los administradores gestionar los permisos de los usuarios en tiempo real en una plataforma IoT.

Aquí se detalla cómo se gestiona esta funcionalidad desde el lado del servidor:

- Propósito de la Lista de Usuarios en Línea
  - El objetivo es desarrollar un servidor IoT que permita a los usuarios administradores ver una lista de todos los usuarios en línea y concederles permisos de lectura y escritura a usuarios no administradores y dispositivos en tiempo real.
  - Esta lista forma parte de un panel de control visible solo para usuarios administradores.
- Implementación en el Lado del Servidor (Python Flask)
  - Recopilación de Datos: Cuando la aplicación Flask devuelve la página web principal (`'index.html'`) al usuario, debe incluir detalles adicionales como el ID de sesión del usuario (`'session-user-ID'`) y la lista de usuarios en línea.

- Función `'get-all-logged-in-users'`: Esta función (presumiblemente definida en un módulo `'my-DB'`) es la encargada de preparar los datos de los usuarios.
  - Crea una variable `'online-user-records'`, que es un mapa inicialmente con la clave `'user-record'` y un valor de lista vacía.
  - Dentro de un bucle `'for'`, esta variable se **rellena** añadiendo los detalles de cada usuario. Por cada usuario, se adjunta una lista que contiene:
    - El nombre del usuario (en el índice 0).
    - El ID del usuario (en el índice 1).
    - El estado de acceso de lectura (en el índice 2).
    - El estado de acceso de escritura (en el índice 3).
  - Los estados de acceso de lectura y escritura (que son 1 o 0) se convierten a las cadenas `'checked'` o `'unchecked'` respectivamente. Esto es crucial porque el código HTML interpretará estas cadenas para establecer el estado de los botones de alternancia (`'switch buttons'`).
  - Finalmente, la función `'get-all-logged-in-users'` devuelve este mapa, utilizando la clave `'online-users'`.

- Visualización en el Lado del Cliente (HTML con Jinja Templates)
  - La página `'index.html'` utiliza **Jinja templates** para procesar los datos recibidos del servidor.
  - Se implementa un bucle `'for' ('for n in online-users-record')` para iterar a través de la lista de `'online-users'`, creando una fila (`'<li>'`) para cada usuario en línea.
  - Dentro de cada fila, se muestra el nombre del usuario (`'n'`), y se utilizan `'n'` para el ID de usuario en los botones, y `'n'` y `'n'` para establecer el estado de los botones de alternancia de permisos de lectura y escritura.
  - Para asegurar que solo los administradores vean el panel de control de usuarios, se añade una declaración `'if'` en el código HTML. Esta declaración compara el `'user-ID'` del usuario actual (obtenido del `'session-user-ID'` enviado por el servidor) con un `'user-ID'` de administrador codificado. Un usuario no administrador no tendrá acceso al panel de control de acceso, mientras que un administrador verá a otros usuarios en la lista en línea.

Este proceso permite que la aplicación Flask gestione y presente dinámicamente una lista interactiva de usuarios en línea, que es esencial para la administración de permisos en tiempo real dentro de la plataforma IoT.

6.3.2. Retorna `user-id` y `online-users-record` a la Página web

En el contexto de la funcionalidad del lado del servidor con Python Flask para una plataforma IoT, la acción de **retornar el `'user-ID'` y la lista `'online-user-records'` a la página web** es fundamental para habilitar el panel de control de administración y la gestión de permisos en tiempo real.

Aquí se detalla cómo se gestiona esta transferencia de datos desde el lado del servidor a la página web:

- Retorno del `'user-ID'` (identificador del usuario):
  - Cuando la aplicación Flask devuelve la página web principal (`'index.html'`) al usuario, incluye detalles adicionales como el `'user-ID'` del usuario actual.
  - Este `'user-ID'` se envía como `'session-user-ID'`.
  - En el lado del cliente (HTML), este `'session-user-ID'` se utiliza en una declaración `'if'` para determinar si el usuario actual es un administrador. Si el `'user-ID'` coincide con un `'user-ID'` de administrador codificado, se le concede acceso al panel de control de acceso. Los usuarios no administradores no verán este panel.

- Retorno de `'online-user-records'` (registros de usuarios en línea):
  - Para proporcionar la lista de usuarios en línea, se invoca una función (presumiblemente 'get-all-logged-in-users' en un módulo 'my-DB') en el lado del servidor.
  - Esta función crea una variable 'online-user-records', que es un mapa inicial con la clave 'user-record' y una lista vacía como valor.

- Luego, dentro de un bucle 'for', esta variable se rellena con los detalles de cada usuario en línea. Por cada usuario, se adjunta una lista que contiene:
  - El nombre del usuario (en el índice 0).
  - El ID del usuario (en el índice 1).
  - El estado de acceso de lectura (en el índice 2).
  - El estado de acceso de escritura (en el índice 3).
- Es crucial que los estados de acceso de lectura y escritura (que son 1 o 0) se conviertan a las cadenas `''checked''` o `''unchecked''` respectivamente. Esto se debe a que el código HTML interpretará estas cadenas para establecer el estado de los botones de alternancia (`'switch buttons'`) en la interfaz de usuario.
- Finalmente, la función `'get-all-logged-in-users'` devuelve este mapa, utilizando la clave `'online-users'`.

- Uso en la Página Web (Jinja Templates):
  - La página `'index.html'` utiliza **Jinja templates** para procesar la variable `'online-users'` recibida del servidor.
  - Un bucle `'for'` (`'for n in online-users-record'`) itera a través de la lista de usuarios en línea, creando una fila (`'<li>'`) para cada usuario.
  - Dentro de cada fila, se muestra el nombre del usuario (`'n'`) y se utilizan `'n'` para el ID del usuario en los botones, y `'n'` y `'n'` para establecer dinámicamente el estado de los botones de alternancia de permisos de lectura y escritura.

Este proceso permite que la aplicación Flask no solo identifique al usuario actual y su rol (admin/no-admin), sino también que **genere y envíe una lista detallada de todos los usuarios en línea con sus permisos actuales**, permitiendo a los administradores visualizar y modificar estos permisos en tiempo real desde el panel de control.

6.3.3. Función `get-all-logged-in-users`

La función `'get-all-logged-in-users'` es un componente central de la funcionalidad del lado del servidor en una aplicación Python Flask, diseñada para **gestionar y presentar la lista de usuarios en línea** a los administradores de una plataforma IoT.

En el contexto más amplio de la funcionalidad del lado del servidor (Python Flask), esta función cumple los siguientes propósitos y se implementa de la siguiente manera:

- Propósito y Contexto
  - La función es invocada por la aplicación Flask en el momento en que se devuelve la página web principal (`'index.html'`) al usuario.
  - Su objetivo principal es preparar y proporcionar la lista de todos los usuarios actualmente conectados para que un usuario administrador pueda verla en un panel de control. Este panel permite a los administradores conceder permisos de lectura y escritura a usuarios no administradores y dispositivos en tiempo real.

- Ubicación y Estructura
  - Aunque no se especifica explícitamente, se sugiere que esta función reside en un módulo llamado `'my-DB'`, indicando su rol en la interacción con la base de datos o el sistema que mantiene el estado de los usuarios.
  - La función crea una variable llamada `'online-user-records'`. Inicialmente, esta variable es un **mapa** (diccionario) con una clave `'user-record'` cuyo valor es una lista vacía.

- Población de Datos
  - Dentro de un bucle `'for'`, la variable `'online-user-records'` se **rellena** con los detalles de cada usuario en línea. Por cada usuario, se adjunta una lista que contiene la siguiente información:
    - El nombre del usuario (en el índice 0).

        - El ID del usuario (en el índice 1).
        - El estado de acceso de lectura (en el índice 2).
        - El estado de acceso de escritura (en el índice 3).

- Conversión de Estados de Permiso para HTML
    - Un paso crucial realizado por esta función es la conversión de los estados de acceso de lectura y escritura. Estos estados, que originalmente son valores binarios (1 o 0), se transforman en las cadenas `''checked''` o `''unchecked''`.
    - Esta conversión es fundamental porque el código HTML en la página web principal (`'index.html'`) interpretará directamente estas cadenas para establecer el estado de los **botones de alternancia** (`'switch buttons'`) que controlan los permisos en la interfaz de usuario.

- Retorno de Datos al Servidor Flask
    - Finalmente, la función `'get-all-logged-in-user'get-all-logged-in-users's'` devuelve el mapa `'online-user-records'`, pero lo hace utilizando la clave `'online-users'`.
    - Esta variable `'online-users'` es entonces accesible en la página `'index.html'` a través de los **Jinja templates** , permitiendo que la interfaz de usuario genere dinámicamente las filas de la tabla con los usuarios y sus respectivos botones de permiso.

En resumen, la función `'get-all-logged-in-users'` es esencial en la arquitectura del lado del servidor de Flask para la **visualización y administración en tiempo real de los permisos de los usuarios en línea**, transformando los datos brutos de la base de datos en un formato fácilmente consumible por el cliente web (HTML/Jinja) para una interacción dinámica.

- Variable `online-users-records` (Mapa: `user-record` ->Lista Vacia).

- En el contexto más amplio de la función `'get-all-logged-in-users'` y la funcionalidad del lado del servidor con Python Flask, la variable `'online-user-records'` (Mapa: `'user-record'` ->Lista Vacía) es fundamental para la construcción y gestión de la lista de usuarios en línea que se presenta a los administradores.

- Aquí se dan los detalles de esta variable:
    - 1. Inicialización de la Variable online-user-records.
        - Dentro de la función get-all-logged-in-users (que se encuentra en el módulo my-DB), el primer paso es crear una variable llamada online-user-records.
        - Esta variable se inicializa como un mapa (diccionario) con la clave user-record y un valor que es una lista vacía. Esta estructura permite que la función comience a recopilar datos de usuarios de manera organizada.
    - 2. Propósito y Población dentro de la Función get-all-logged-in-users.
        - La función get-all-logged-in-users es invocada desde el lado del servidor Flask para preparar la lista de usuarios en línea y sus permisos.
        - El mapa online-user-records se rellena dentro de un bucle for. Por cada usuario en línea, se le adjunta una lista con sus detalles:
            - Nombre del usuario (en el índice 0).
            - ID del usuario (en el índice 1).
            - Estado de acceso de lectura (en el índice 2).
            - Estado de acceso de escritura (en el índice 3
        - Es crucial destacar que los estados de acceso de lectura y escritura (que son 1 o 0) se convierten a las cadenas 'checked' o 'unchecked' respectivamente. Esta conversión es necesaria porque el código HTML en la página index.html (utilizando Jinja templates) leerá estas cadenas para establecer el estado de los botones de alternancia (switch buttons) que controlan los permisos
    - 3. Retorno de online-user-records para la Página Web.

    - Finalmente, la función get-all-logged-in-users devuelve este mapa online-user-records, pero lo hace con la clave online-users. Esta variable online-users es la que luego se utiliza en la página index.html mediante Jinja templates para construir dinámicamente el panel de control. Un bucle for itera sobre online-users-record para crear una fila (<li>) por cada usuario, mostrando su nombre y los botones de permisos con sus estados correctos (checked/unchecked).
  - En síntesis, la inicialización de online-user-records como un mapa con una lista vacía es el punto de partida para que la función get-all-logged-in-users pueda estructurar y procesar los datos de los usuarios en línea, incluyendo sus permisos de lectura y escritura, y convertirlos a un formato legible por HTML, facilitando así la gestión de permisos en tiempo real desde el panel de control de administración.

- Poblar con (Nombre, User ID, Lectura, Escritura)

- En el contexto más amplio de la función get-all-logged-in-users y la funcionalidad del lado del servidor con Python Flask, el proceso de poblar la variable online-user-records con el nombre, el ID de usuario, y los estados de acceso de lectura y escritura es fundamental para construir el panel de control de administración en tiempo real.

- Se detalla este proceso de la siguiente manera:

  - 1. Contexto de la Función get-all-logged-in-users.
    - Esta función, ubicada presumiblemente en el módulo my-DB, es la responsable de recopilar y estructurar los datos de los usuarios actualmente en línea.
    - Se invoca cuando la aplicación Flask devuelve la página web principal (index.html) al usuario, con el fin de proporcionar detalles adicionales para la interfaz.
  - 2. Inicialización de online-user-records.
    - Dentro de esta función, se crea una variable llamada online-user-records. Se inicializa como un mapa (diccionario) con la clave user-record y una lista vacía como valor. Esta estructura sirve como contenedor para los datos de los usuarios.
  - 3. Proceso de Población (Nombre, User ID, Lectura, Escritura)
    - La variable online-user-records se rellena dentro de un bucle for. Por cada usuario en línea, la función adjunta una lista que contiene sus detalles específicos en índices predefinidos:
      - El nombre del usuario se añade en el índice 0.
      - El ID del usuario se añade en el índice 1.
      - El estado de acceso de lectura se añade en el índice 2.
      - El estado de acceso de escritura se añade en el índice 3.
  - 4. Transformación de los Estados de Permiso (Lectura y Escritura).
    - Un paso crítico durante la población es la conversión de los estados de acceso de lectura y escritura. Inicialmente, estos estados son valores numéricos (1 para permitido, 0 para denegado).
    - Sin embargo, la función los transforma a las cadenas 'checked' o 'unchecked'.
    - Esta conversión es esencial porque el código HTML en la página index.html (que utiliza Jinja templates) leerá directamente estas cadenas para establecer el estado visual de los botones de alternancia (switch buttons) que controlan los permisos en el panel de administración.
  - 5. Retorno de los Datos Poblados.
    - Una vez que online-user-records ha sido poblada con los datos transformados de todos los usuarios en línea, la función get-all-logged-in-users devuelve este mapa. Lo hace utilizando la clave online-users para que sea accesible en la plantilla Jinja de index.html

  Este meticuloso proceso de población garantiza que la función get-all-logged-in-users no solo recopile la información necesaria de los usuarios en línea, sino que también la formatee adecuadamente para su presentación en la interfaz web, permitiendo a los administradores una gestión intuitiva y en tiempo real de los permisos en la plataforma IoT.

- Convertir Permisos de Lectura/Escritura a `checked` o `unchecked` para HTML.

- En el contexto más amplio de la función get-all-logged-in-users y la funcionalidad del lado del servidor con Python Flask, la conversión de los permisos de lectura y escritura a las cadenas 'checked' o 'unchecked' para HTML es un paso crucial para la visualización y gestión efectiva de permisos en el panel de control del administrador.
- Aquí se dan los detalles de este proceso:
  - 1. Propósito de la Función get-all-logged-in-users:
    - Esta función, presumiblemente ubicada en el módulo my-DB, es responsable de recopilar los detalles de todos los usuarios en línea.
    - Su objetivo es preparar estos datos para ser presentados en la página web principal (index.html) de la aplicación Flask, específicamente en el panel de control visible para los usuarios administradores. Este panel permite a los administradores conceder permisos de lectura y escritura en tiempo real a otros usuarios y dispositivos.
  - 2. La Necesidad de la Conversión:
    - Originalmente, los estados de acceso de lectura y escritura se almacenan como valores numéricos, típicamente 1 para un permiso concedido y 0 para un permiso denegado.
    - Sin embargo, el código HTML en la interfaz de usuario utiliza botones de alternancia (switch buttons) para representar estos permisos. Para que estos botones se muestren correctamente como 'activados ' o 'desactivados ', el código HTML no interpreta directamente 1 o 0, sino que necesita cadenas específicas como 'checked' (para activado) o 'unchecked' (para desactivado).
  - 3. Proceso de Conversión:
    - La función get-all-logged-in-users realiza esta conversión explícitamente.
    - Durante la población de la variable online-user-records (un mapa que contiene listas con los detalles de cada usuario), los valores numéricos de lectura y escritura se transforman.
    - Se utiliza una lógica condicional (sentencias if/else) para realizar esta transformación:
      - Si el acceso de lectura (o escritura) es 1, la variable correspondiente se establece en la cadena 'checked'.
      - De lo contrario (si es 0), la variable se establece en la cadena 'unchecked'.
  - 4. Integración con HTML y Jinja Templates:
    - Una vez que los estados de lectura y escritura se han convertido a 'checked' o 'unchecked', estos valores se incluyen en la lista de detalles de cada usuario que se adjunta a online-user-records.
    - La función devuelve este mapa bajo la clave online-users, que luego es utilizada por los Jinja templates en la página index.html.
    - En el HTML, cuando se itera a través de online-users-record para crear cada fila de usuario, estas cadenas 'checked' o 'unchecked' se insertan directamente en los atributos de los botones de alternancia (switch buttons), lo que permite que el navegador muestre su estado correcto de forma dinámica.
  - En resumen, la conversión de los permisos de lectura y escritura a 'checked' o 'unchecked' es un paso de mapeo de datos crítico en el lado del servidor que facilita la comunicación entre la lógica de Python Flask (donde se gestionan los permisos) y la presentación en el lado del cliente (HTML con Jinja), asegurando que los administradores puedan visualizar y manipular los permisos de los usuarios en línea de manera intuitiva y en tiempo real.
- Retorna Mapa con Clave 'on-line-users' y valor una lista.
- En el contexto más amplio de la función get-all-logged-in-users y la funcionalidad del lado del servidor con Python Flask, el hecho de que retorne un mapa con la clave 'online-users' y un valor que es una lista es un aspecto fundamental para la comunicación efectiva entre el servidor y la interfaz de usuario.
- Se detalla este proceso de la siguiente manera:
  - 1. Propósito General de la Función get-all-logged-in-users:

- Esta función, presumiblemente definida en un módulo my-DB, tiene la responsabilidad de recopilar y preparar los datos de todos los usuarios que están actualmente en línea.
- Su objetivo es hacer que esta información esté disponible para la página web principal (index.html), especialmente para el panel de control del administrador, donde se visualizan y gestionan los permisos de lectura y escritura en tiempo real.

- 2. Preparación de los Datos (online-user-records):
  - Dentro de la función, se inicializa una variable llamada online-user-records. Esta variable comienza como un mapa (diccionario) con la clave user-record y un valor que es una lista vacía.
  - A continuación, en un bucle for, esta variable online-user-records se rellena añadiendo los detalles de cada usuario en línea. Cada entrada es una lista que contiene el nombre del usuario (índice 0), el ID del usuario (índice 1), el estado de acceso de lectura (índice 2) y el estado de acceso de escritura (índice 3).
  - Un paso crucial antes del retorno es la conversión de los estados de acceso de lectura y escritura (que son 1 o 0) a las cadenas 'checked' o 'unchecked'. Esto es esencial porque el código HTML interpretará estas cadenas para establecer el estado de los botones de alternancia (switch buttons) en la interfaz de usuario.
- 3. El Retorno del Mapa con la Clave 'online-users':
  - Esta variable online-users (el mapa retornado) es entonces accesible en la página index.html del lado del cliente a través de los Jinja templates.
  - La plantilla HTML utiliza un bucle for (for n in online-users-record) para iterar a través de la lista de usuarios. Este bucle permite crear dinámicamente una fila (<li>) para cada usuario en línea.
  - Dentro de cada fila, se extraen y muestran el nombre del usuario (usando n), el ID del usuario (usando n), y los estados de lectura y escritura (n y n), que al ser 'checked' o 'unchecked', configuran directamente los botones de alternancia en la interfaz.
- 4. Uso en el Lado del Cliente (Jinja Templates):
  - Esta variable online-users (el mapa retornado) es entonces accesible en la página index.html del lado del cliente a través de los Jinja templates.
  - La plantilla HTML utiliza un bucle for (for n in online-users-record) para iterar a través de la lista de usuarios. Este bucle permite crear dinámicamente una fila (<li>) para cada usuario en línea.
  - Dentro de cada fila, se extraen y muestran el nombre del usuario (usando n), el ID del usuario (usando n), y los estados de lectura y escritura (n y n), que al ser 'checked' o 'unchecked', configuran directamente los botones de alternancia en la interfaz.
- En conclusión, el patrón de retornar un mapa con la clave 'online-users' y una lista de registros desde la función get-all-logged-in-users es una estrategia deliberada en el lado del servidor para empaquetar y entregar de manera estructurada los datos de los usuarios en línea a la plantilla HTML. Esto facilita la generación dinámica del panel de control de administración, permitiendo a los administradores visualizar y gestionar los permisos de los usuarios en tiempo real en la plataforma IoT.

6.4. Renderizado de la Interfaz (Jinja Templates)

Las Plantillas Jinja (Jinja Templates) son un componente esencial del renderizado de la interfaz de usuario ('index.html') que permite implementar de manera dinámica y segura las reglas para usuarios administradores y no administradores en la plataforma IoT. Permiten que el servidor Flask inyecte datos y lógica condicional directamente en el HTML antes de que se envíe al navegador del cliente.

Aquí se detalla cómo las Plantillas Jinja se utilizan en este contexto:

- 1. Población Dinámica de la Lista de Usuarios en Línea
  - El servidor Flask prepara una lista de registros de usuarios en línea ('online users record') que incluye el nombre del usuario, su ID, y el estado de sus permisos de lectura y escritura.

- Las plantillas Jinja en 'index.html' utilizan un bucle 'for' ('for n in online users record') para iterar sobre esta lista.
- Cada iteración del bucle genera una fila ('<li>') en la interfaz, mostrando la información de un usuario en línea.
- Dentro de cada fila, se accede a los elementos del registro del usuario utilizando índices:
  - 'n' para el nombre del usuario.
  - 'n' para el ID del usuario.

- 2. Configuración Dinámica de Permisos de Lectura y Escritura
  - El servidor es responsable de convertir los estados numéricos de los permisos de lectura y escritura (por ejemplo, 1 para permitido) a las cadenas checked (activado) o unchecked (desactivado).
  - Estas cadenas son luego inyectadas por Jinja directamente en el atributo 'checked' de los botones tipo 'switch ' en el HTML.
  - De esta forma, los botones de conmutación de permisos de lectura y escritura ('read user ID' y 'write user ID') para cada usuario aparecen con el estado correcto reflejando sus permisos actuales.
  - Los estados de lectura y escritura se obtienen de 'n' y 'n' del registro del usuario, respectivamente.
- 3. Restricción del Panel de Control para Usuarios Administradores
  - Una de las funcionalidades más críticas implementadas con Jinja es asegurar que el panel de control completo, que contiene la lista de usuarios y los botones para gestionar sus permisos, sea visible 'únicamente para los usuarios administradores ' .
  - Esto se logra añadiendo una sentencia 'if' directamente en el código HTML del 'index.html'.
  - Esta condición verifica si el 'user ID' del usuario actualmente conectado (proporcionado por el servidor a través de la sesión) coincide con un 'user ID' de administrador predefinido.
  - Si e l 'user ID' actual no es el del administrador (por ejemplo, para un usuario como 'Anam Chaudhary ' que no es administrador), el panel de control no se renderiza en absoluto en la página web que recibe el cliente.
  - Esto garantiza que los usuarios no administradores no tengan acceso visual ni funcional a las herramientas de gestión de permisos, reforzando la seguridad de la plataforma.

En resumen, las Plantillas Jinja son fundamentales para crear una interfaz de usuario dinámica que se adapta al rol del usuario. Permiten al servidor renderizar contenido HTML personalizado, mostrando listas de usuarios con sus permisos actuales y, lo que es más importante, controlando la visibilidad de funciones críticas de administración, asegurando que solo los usuarios con los privilegios adecuados puedan ver y modificar las reglas de acceso en la plataforma IoT.

6.4.1. Bucle `'for n in online-users-record'`

El bucle `'for n in online-users-record'` es una parte fundamental del renderizado de la interfaz en el lado del cliente utilizando **plantillas Jinja** dentro del contexto más amplio de la Visibilidad del Panel de Control (Admin Solamente). Permite mostrar dinámicamente la lista de usuarios en línea en el panel de administración.

A continuación, se detalla este aspecto:

- Origen de `'online-users-record'`:
  - Para poblar la lista de usuarios en línea, el servidor (específicamente la función `'get all logged in users'` en `'my DB'`) es responsable de **devolver un mapa** que contiene los detalles de todos los usuarios que han iniciado sesión.
  - Este mapa se nombra `'online user records'` (o `'online-users-record'`) y se construye con el `'name'` del usuario en el índice 0, el `'user-id'` en el índice 1, el estado de `'read'` (lectura) en el índice 2 y el estado de `'write'` (escritura) en el índice 3.

- Los estados de `'read'` y `'write'` se convierten a `checked` o `unchecked` para que el código HTML pueda establecer correctamente el estado de los botones de interruptor.
- Este `'online-users-record'` (que es una lista, a pesar de ser referido como un mapa con un valor de lista) se envía al cliente junto con el `'user-id'` de la sesión cuando se devuelve la página principal al usuario.

- Uso del Bucle `'for'` en `'index.html'` (Jinja):
  - Una vez que la variable `'online-users-record'` llega a la página `'index.html'`, las plantillas Jinja se utilizan para iterar sobre ella y renderizar la información.
  - El bucle se escribe como `'for n in online users record'`.
  - Este bucle `'for'` se ejecuta para cada elemento en la lista `'online-users-record'`, creando **múltiples filas** en la tabla del panel de control, una por cada usuario en línea.
  - Cada `'n'` dentro del bucle **representa una lista individual** que contiene los detalles de un usuario en particular dentro de `'user records'`.

- Acceso a los Datos del Usuario dentro del Bucle:
  - Dentro de cada iteración del bucle (`'n'`), se accede a los datos específicos del usuario utilizando los índices definidos:
    - El **nombre del usuario** se muestra tomando el `'0th index'` de `'n'`.
    - El `'user-id'` del usuario se obtiene del `'1st index'` de `'n'`. Este `'user-id'` se utiliza para reemplazar placeholders en los ID de los botones de interruptor y el botón `APLICAR` (acceso).
    - El estado de lectura `('read')` se encuentra en el `'2nd index'` de `'n'`.
    - El estado de escritura `('write')` se encuentra en el `'3rd index'` de `'n'`.
  - Estos estados de lectura y escritura `('n' y 'n')` se utilizan para establecer las condiciones de `checked` o `'unchecked '` para los botones de interruptor de permisos en la interfaz.

- Propósito en el Renderizado de la Interfaz:
  - El bucle `'for n in online-users-record'` es esencial para construir la etapa del panel de control de administradores donde se lista a **todos los usuarios en línea**.
  - Esta lista permite al administrador ver a otros usuarios (por ejemplo, `Anam Chaudhary` en el escenario de prueba) y, lo más importante, **interactuar con botones de interruptor para conceder permisos de lectura y escritura en tiempo real**.

En resumen, el bucle `'for n in online-users-record'` combinado con las plantillas Jinja es el mecanismo del lado del cliente que permite al panel de control del administrador renderizar de forma dinámica y detallada la lista de usuarios en línea y sus respectivos controles de permisos, basándose en los datos proporcionados por el servidor.

6.4.2. Acceso a Datos por índice

En esta etapa se explica en detalle cómo se accede a los datos de los usuarios en línea mediante índices (0: Nombre, 1: User ID, 2: Lectura, 3: Escritura) en el contexto del renderizado de la interfaz con **Plantillas Jinja**. Este método es clave para mostrar dinámicamente la información en el panel de control del administrador.

A continuación, se describe cómo se maneja este acceso a los datos:

- Estructura de `'online-users-record'`:
  - La información de los usuarios en línea se organiza en una variable llamada `'online-users-record'`. Aunque se la describe como un **mapa con la clave como registro de usuario y el valor es una list**, se especifican que es una **lista** en sí misma, donde cada elemento dentro de ella es otra lista que representa los detalles de un usuario individual.
  - Esta estructura es devuelta por el servidor (desde la función `'get all logged in users'` en `'my DB'`) al cliente, junto con el `'user-id'` de la sesión.

- Organización de Datos dentro de Cada Registro de Usuario:
  - Dentro de cada lista o **registro** que representa a un usuario, la información se almacena en posiciones específicas o **índices**:
    - Índice 0: Contiene el **nombre del usuario**.
    - Índice 1: Almacena el **ID del usuario ('user-id')**.
    - Indice 2: Indica el estado de los permisos de **lectura ('read')** del usuario. Este valor se convierte a **'checked'** o **'unchecked '** para que el código HTML pueda interpretarlo directamente.
    - Índice 3: Indica el estado de los permisos de **escritura ('write')** del usuario. Similar al de lectura, también se convierte a **'checked'** o **'unchecked'**.
- Renderizado en `'index.html'` con Jinja Templates:
  - Para mostrar esta lista de usuarios en la interfaz, la página `'index.html'` utiliza un **bucle 'for' de Jinja**: `'for n in online users record'`.
  - Este bucle itera sobre la lista `'online-users-record'`, y en cada iteración, la variable `'n'` representa la lista de datos de un usuario individual.
  - Dentro del bucle, se accede a los datos específicos de cada usuario utilizando sus índices:
    - El **nombre del usuario** se muestra al acceder a `'n'` en el índice 0 (`'n'`).
    - El `'user-id'` se obtiene de `'n'` en el índice 1 (`'n'`) y se utiliza para construir dinámicamente los ID de los botones de interruptor (`'read user ID'`, `'write user ID'`) y el botón `APLICAR` (`'access user ID'`).
    - Los estados de **lectura y escritura** se acceden a través de `'n'` en los índices 2 y 3 (`'n'` y `'n'`), respectivamente, para establecer el estado inicial (`'checked'` o `'unchecked'`) de los botones de interruptor correspondientes en la interfaz.
- Contexto Amplio de Renderizado de la Interfaz:
  - Este método de acceso a datos por índice es crucial para **renderizar dinámicamente la etapa del panel de control de administradores** donde se lista a todos los usuarios en línea.
  - La posibilidad de ver a otros usuarios (como `Anam Chaudhary` en el ejemplo) y sus permisos se logra gracias a esta estructura de datos y su renderizado por medio de Jinja.
  - Permite a los administradores **gestionar en tiempo real los permisos de lectura y escritura** de usuarios y dispositivos no administradores mediante los botones de interruptor y el botón **APLICAR** que se generan en cada fila de la tabla.

En síntesis, el acceso a datos por índice dentro del bucle `'for n in online-users-record'` y las plantillas Jinja es el mecanismo fundamental para que el panel de control del administrador pueda construir una **vista interactiva y actualizada de todos los usuarios en línea**, facilitando la gestión de permisos en el ecosistema IoT.

6.5. Visibilidad del Panel de Control (Admin Solamente)

La visibilidad del panel de control es un aspecto fundamental de las reglas para usuarios administradores y no administradores, y se aqui detalla cómo se asegura que solo los usuarios administradores puedan acceder a las funcionalidades de gestión de permisos. Esta restricción se implementa principalmente a través del renderizado de la interfaz del lado del servidor utilizando plantillas Jinja .

Aquí se detalla este aspecto:

- 1. Definición y Contenido del Panel de Control
  - El panel de control al que se hace referencia es una etapa específica del dashboard de la plataforma IoT.
  - Esta etapa está diseñada para mostrar una lista de todos los usuarios en línea .

- Junto al nombre de cada usuario en línea, el panel incluye botones de conmutación (switch buttons) para otorgar permisos de lectura y escritura . También hay un botón para aplicar los cambios.
- La interfaz permite a los administradores gestionar los permisos de otros usuarios en tiempo real.

- 2. Restricción de Visibilidad: Solo para Administradores
  - Se enfatiza que la última funcionalidad a implementar es asegurarse de que 'este panel de control es únicamente visible para los usuarios administradores ' .
  - Esto significa que los usuarios que no tienen el rol de administrador no deben ver esta etapa en absoluto.
  - El dashboard para usuarios administradores mostrará la lista de usuarios en línea con los botones para conceder permisos de lectura y escritura.
- 3. Implementación a Través de Plantillas Jinja y Lógica Condicional
  - La restricción de visibilidad se logra añadiendo una sentencia 'if' directamente en el código HTML de 'index.html' .
  - Esta sentencia 'if' compara el 'user ID' del usuario actualmente conectado (que es proporcionado por el servidor a la plantilla Jinja) con un 'user ID' de administrador codificado (hard-coded) .
  - El 'user ID' del usuario actual se obtiene de la sesión y se envía al cliente. Para identificar el 'user ID' del administrador, se sugiere imprimir el 'user ID' de la sesión en la consola y luego codificarlo en el HTML.
  - La lógica es clara: 'si el user ID es igual a mi user ID ' (refiriéndose al ID del administrador), entonces el panel de control se renderizará. Si no coincide, el panel no se mostrará.
  - Es importante recordar 'cerrar esta sentencia 'if' ' al final de la etapa del panel de control.
- 4. Experiencia del Usuario Administrador vs. No Administrador
  - Cuando un usuario administrador inicia sesión, verá el panel de control con la lista de usuarios y las opciones para modificar permisos.
  - Sin embargo, un usuario no administrador , como el ejemplo de 'Anam Chaudhary ', al iniciar sesión en el servidor IoT de Packt, 'no tendrá acceso al panel de control de acceso ' . Esto demuestra que la implementación condicional de Jinja funciona eficazmente para ocultar la etapa de administración a usuarios no autorizados.

En resumen, la visibilidad del panel de control para gestionar permisos de usuarios es una característica de seguridad crítica. Se implementa en el lado del servidor mediante la inyección condicional de HTML a través de plantillas Jinja, asegurando que solo los usuarios administradores, identificados por su 'user ID', puedan ver y utilizar esta funcionalidad. Esto garantiza que las reglas para usuarios administradores y no administradores se apliquen estrictamente, protegiendo la integridad de la gestión de acceso en la plataforma IoT.

6.5.1. Sentencia `'if user-id == my-admin-user-id'` en HTML (codificado)

La sentencia `'if user-id == my-admin-user-id'` en el código HTML (codificado) se utiliza para controlar la visibilidad del panel de control, asegurando que solo los usuarios administradores puedan verlo.

A continuación, se detalla su contexto y funcionamiento:

- Propósito de la Sentencia `'if'`.
  - La funcionalidad principal de esta sentencia es **hacer que el panel de control sea visible únicamente para los usuarios administradores**.
  - Se añade una sentencia `'if'` antes de que comience la etapa del panel de control en el código HTML.
  - El `'user-id'` que se envía al cliente se utiliza en esta sentencia para determinar si el usuario actual es el administrador.

- Implementación del `'my-admin-user-id'` (Hardcodeado).
  - Para implementar esta verificación, el `'my-admin-user-id'` se obtiene imprimiendo el `'session user-id'` en la consola y luego hardcodeándolo directamente en el código HTML.
  - Después de la sentencia `'if'`, se debe incluir una sentencia `'end if'` para cerrar el bloque condicional.
- Comportamiento para Usuarios No Administradores.
  - Cuando un usuario que no es administrador (por ejemplo, 'Anam Chaudhary ' en el ejemplo proporcionado) inicia sesión en el servidor IOT, **no tendrá acceso al panel de control de acceso**.
  - Sin embargo, desde el panel de control del administrador, el usuario no administrador sí aparecerá en la lista de usuarios en línea.
- Contexto Amplio de Visibilidad del Panel de Control (Admin Solamente).
  - Esta implementación forma parte de un enfoque más amplio para **permitir a los usuarios administradores gestionar permisos**.
  - El panel de control del administrador está diseñado para mostrar una lista de todos los usuarios en línea con botones para conceder permisos de lectura y escritura en tiempo real a los usuarios y dispositivos no administradores.
  - Al seleccionar el botón `APLICAR`, los permisos de lectura y escritura de un usuario específico se cambiarán en tiempo real.
  - Antes de conceder permisos de lectura y escritura, es necesario generar una clave de autorización para el usuario específico y almacenarla en la base de datos.

En resumen, la sentencia `'if user-id == my-admin-user-id'` es un mecanismo del lado del cliente, codificado en el HTML, para **restringir la interfaz de usuario del panel de control a los administradores**, lo que les permite gestionar las autorizaciones de otros usuarios y dispositivos dentro del ecosistema IOT.

6.5.2. Verificar `'session.user-id'` en consola

Para controlar la visibilidad del panel de control de manera que solo los administradores puedan acceder a él, se indican que es necesario **verificar el `'session.user-id'` en la consola**.

Aquí se detalla el proceso y su relevancia en el contexto de la visibilidad del panel de control (solo para administradores):

- Objetivo: Asegurar que el panel de control sea visible **únicamente para los usuarios administradores**.
- Mecanismo de Verificación: Se añade una **sentencia `'if'` en el código HTML** antes de que comience la etapa del panel de control. Esta sentencia comparará el `'user-id'` actual del usuario con el `'ID'` de un usuario administrador predefinido.
- Determinación del 'ID' del Administrador `('my-admin-user-id')`: Para obtener el `'ID'` del usuario administrador que se utilizará en la sentencia `'if'`, se debe **imprimir el `'session user-id'` en la consola**. Una vez obtenido, este `'ID'` se **hardcodea (se codifica directamente) en el código HTML**. Es importante recordar cerrar la sentencia `'if'` con un `'end if'`.
- Comportamiento para Usuarios No Administradores: Como resultado de esta implementación, si un usuario que no es el administrador (por ejemplo, `'Anam Chaudhary'`) inicia sesión, **no tendrá acceso al panel de control de acceso**. Sin embargo, desde el panel de control del administrador, el usuario no administrador sí aparecerá en la lista de usuarios en línea.
- Contexto más Amplio (Visibilidad del Panel de Control Admin Solamente): Esta funcionalidad es crucial para el diseño del panel de control de administradores, que permite a los administradores **gestionar permisos en tiempo real**. El panel muestra una lista de todos los usuarios en línea con botones para conceder permisos de lectura y escritura a usuarios y dispositivos no administradores. Al seleccionar el botón `APLICAR`, los permisos se cambian en tiempo real. Antes de conceder permisos, se requiere generar una clave de autorización para el usuario específico y almacenarla en la base de datos.

En síntesis, la verificación de `'session.user-id'` en la consola es un paso intermedio para **identificar y hardcodear el `'ID'` del administrador directamente en el código HTML**, lo que permite una comprobación del lado del cliente para **restringir la visibilidad de la interfaz del panel de control exclusivamente a los usuarios administradores**.

6.5.3. Prueba con Usuario No Administrador (no tiene acceso al panel)

Se describe claramente la prueba con un usuario no administrador en el contexto de la visibilidad del panel de control (Admin Solamente), confirmando que **dichos usuarios no tienen acceso al panel**.

A continuación, se detalla este aspecto:

- Objetivo de la Prueba:
  - El propósito de esta prueba es **asegurarse de que el panel de control sea visible únicamente para los usuarios administradores**. Esto se logra mediante la implementación de una sentencia `'if'` en el código HTML que compara el `'user-id'` del usuario actual con el `'ID'` del administrador, el cual ha sido previamente obtenido de la `'session user-id'` en la consola y codificado.
- Escenario de Prueba con Usuario No Administrador:
  - Para verificar la funcionalidad, se describe un escenario donde se reinicia el servidor y se conecta **otro usuario que no es el administrador**.
- Comportamiento Esperado y Observado:
  - Como se esperaba, si el usuario no es el administrador, **no obtendrá acceso al panel de control**.
  - Se utiliza el ejemplo de una cuenta de Facebook con el nombre `Anam Chaudhary`, configurada como usuario no administrador. Al iniciar sesión desde esa cuenta en el servidor IOT, **Anam Chaudhary no obtiene acceso al panel de control de acceso**.
- Impacto en el Panel del Administrador:
  - A pesar de que el usuario no administrador no ve el panel de control, **desde el panel de control del administrador, si se actualiza la página, se verá al usuario `Anam Chaudhary` en la lista de usuarios en línea**.
- Contexto Amplio de Visibilidad (Admin Solamente):
  - Esta prueba confirma la efectividad de la medida de seguridad implementada para el panel de control. El panel de control del administrador está diseñado para mostrar una **lista de todos los usuarios en línea**, permitiendo a los administradores **conceder permisos de lectura y escritura en tiempo real** a usuarios y dispositivos no administradores mediante botones específicos. Al seleccionar `APLICAR`, los permisos de un usuario específico se cambian en tiempo real.
  - Antes de otorgar estos permisos, es necesario **generar una clave de autorización** para el usuario y almacenarla en la base de datos.
  - Por lo tanto, la restricción de visibilidad a los administradores es fundamental para garantizar que solo ellos puedan realizar estas acciones críticas de gestión de permisos.

En síntesis, se demuestran que, tras implementar la lógica de verificación de `'user-id'`, **los usuarios no administradores son efectivamente impedidos de acceder al panel de control**, lo que valida la funcionalidad de seguridad y la exclusividad del panel para la gestión de permisos por parte de los administradores.

6.5.4. Prueba con Administrador (puede ver otros usuarios en lista online)

Se indica claramente que la **prueba con un administrador demuestra que este puede ver a otros usuarios en la lista de usuarios en línea**, lo cual es fundamental en el contexto más amplio de la Visibilidad del Panel de Control (Admin Solamente).

A continuación, se detalla este aspecto:

- Diseño del Panel de Control para Administradores:
  - El panel de control está diseñado para incluir una etapa específica que lista a **todos los usuarios en línea**.
  - Junto al nombre de cada usuario en línea, se prevén botones para **otorgar permisos de lectura y escritura**, así como un botón para `APLICAR` los cambios.
  - La meta de esta característica es que los administradores puedan **gestionar permisos de lectura y escritura en tiempo real** para los usuarios y dispositivos no administradores.
- Lógica de Servidor para la Lista de Usuarios en Línea:
  - Para poblar esta lista, se implementa una funcionalidad en el servidor para **devolver el `'user-id'` y una lista de todos los usuarios que han iniciado sesión**.
  - La función `'get all logged in users'` (obtener todos los usuarios que han iniciado sesión) crea un mapa (`'online user records'`) que se llena con el nombre del usuario (índice 0), el `'user-id'` (índice 1) y los estados de lectura y escritura (índices 2 y 3 respectivamente).
  - Estos datos se devuelven al cliente para ser utilizados en la página `'index.html'` mediante plantillas Jinja, utilizando un bucle `'for'` para crear una fila por cada usuario en línea.
- Confirmación de la Visibilidad del Administrador en una Prueba:
  - Se describe un escenario de prueba en el que un usuario no administrador (por ejemplo, `Anam Chaudhary`) inicia sesión y, como se espera, no tiene acceso al panel de control.
  - Sin embargo, desde el **panel de control del administrador, al refrescar la página, se puede ver al usuario `Anam Chaudhary` en la lista de usuarios en línea**. Esta observación directa confirma que el administrador tiene la capacidad de ver a otros usuarios.
- Contexto Amplio de Visibilidad del Panel de Control (Admin Solamente):
  - La capacidad del administrador de ver una lista de todos los usuarios en línea es una **característica clave del diseño del sistema para la gestión de la seguridad y los permisos**.
  - Esta visibilidad permite al administrador interactuar con cada usuario en la lista para **concederles o denegarles permisos específicos**.
  - Antes de conceder permisos de lectura y escritura, es necesario **generar una clave de autorización para el usuario y almacenarla en la base de datos**.

En resumen, se detalla cómo el panel de control del administrador está diseñado para mostrar activamente una lista de todos los usuarios en línea, una capacidad que se verifica mediante pruebas directas. Esta funcionalidad es esencial para que los administradores puedan llevar a cabo su rol de **gestión y control de permisos en tiempo real** dentro del ecosistema de IoT.

6.6. ENDPOINT Para el BOTóN `APLICAR`

El Endpoint para el Botón `APLICAR` es una funcionalidad central en la gestión de permisos de usuarios dentro de la plataforma IoT, y su implementación está directamente ligada a las **Reglas para Usuarios Administradores y No Administradores**, asegurando que solo los usuarios autorizados puedan realizar cambios.

Aquí se detalla lo que se propone al respecto en este proyecto:

- 1. Propósito y Ubicación del Botón 'Aplicar'

- El botón `APLICAR` forma parte del panel de control de acceso , que es visible 'únicamente para los usuarios administradores '.
- Este botón se encuentra junto a las opciones de permisos de lectura y escritura para cada usuario en línea listado en el dashboard.
- Su función es `'aplicar los cambios'` realizados en los permisos de lectura y escritura de un usuario específico. Al seleccionarlo, los `'permisos de acceso de lectura y escritura de ese usuario específico cambiarán en tiempo real '`.

- 2. Interacción del Lado del Cliente (JavaScript)
  - Se añade un endpoint para este botón 'Aplicar' que enviará una solicitud desde el código JavaScript a la aplicación Flask.
  - En 'main.js', se implementa un método que 'escucha cualquier botón de conmutación (switch button) que se extrae de nuestro dashboard '.
  - Este método verifica si la ID del botón de conmutación 'comienza con 'XS' ' (posiblemente refiriéndose a 'access') y luego divide esta ID para extraer la ID del usuario .
  - Además, lee el estado de los interruptores de lectura y escritura asociados a ese usuario.
  - Finalmente, envía la solicitud al servidor Flask en un formato específico: 'grant - user ID - read state - write state' .
- 3. Recepción y Procesamiento en el Servidor (Flask)
  - En la aplicación Flask, se añade un endpoint para recibir esta solicitud 'grant user ID read and write' .
  - Verificación de permisos de administrador: Es crucial que el servidor 'verifique una vez más si esta solicitud de concesión proviene de un usuario administrador '. Esto refuerza las reglas de acceso, asegurando que solo un administrador pueda iniciar cambios en los permisos.
  - Respuesta condicional:
    - Si la solicitud no proviene de un administrador, el servidor enviará una respuesta de 'acceso denegado ' ('access denied') .
    - Si todo está correcto y la solicitud es de un administrador, se enviará una respuesta de 'acceso concedido ' ('access granted') .
  - Acciones posteriores: Si el acceso es concedido, el servidor realiza dos acciones principales:
    - 1. Almacenar los permisos de lectura y escritura del usuario en la base de datos .
    - 2. Llamar al servidor PubNub para otorgar acceso de lectura y escritura a este usuario específico .
- 4. Manejo de la Respuesta en el Cliente
  - En 'main.js', se maneja la respuesta del servidor, extrayendo el JSON.
  - Se verifica si el JSON tiene la clave 'access ' y si el 'access ' es 'granted '.
  - Si el acceso es concedido, se 'restablece la suscripción en el canal nuevamente '.
- 5. Requisito Previos: Clave de Autorización
  - Se aclara que la 'concesión de permisos de lectura y escritura ' es un segundo paso. El primer paso que lo precede es 'generar la clave de autorización para ese usuario específico y almacenarla en la base de datos ' . Esto asegura que haya una base de seguridad y autenticación antes de que se puedan modificar los permisos.

En resumen, el endpoint para el botón 'Aplicar' no solo es una interfaz para modificar permisos, sino que está profundamente integrado con la arquitectura de seguridad de la plataforma. Garantiza que cualquier cambio en los derechos de acceso sea iniciado por un usuario administrador validado y que estos cambios se reflejen de manera persistente en la base de datos y en tiempo real a través de PubNub, consolidando así las reglas para usuarios administradores y no administradores.

6.6.1. Método JavaScript en main.js

En el contexto más amplio del Endpoint para el Botón 'Aplicar', el método JavaScript en main.js juega un papel fundamental para habilitar la funcionalidad de gestión de permisos de usuario en el panel de control del administrador en el index.html. Aquí se detalla lo que se dice sobre este método:

- 1. Propósito y Visibilidad del Botón 'Aplicar' (`ID: access_user ID`):
  - El botón 'Aplicar' es un 'botón táctil' que se encuentra en cada fila de usuario del panel de control, adyacente a los interruptores de permisos de lectura y escritura.
  - Su función es cambiar los permisos de acceso de lectura y escritura de un usuario específico en tiempo real cuando se selecciona.
  - Todo el panel, incluyendo este botón, es visible únicamente para usuarios administradores.
- 2. El Método JavaScript en main.js (Lado del Cliente):
  - El archivo main.js contiene un método diseñado para escuchar cualquier botón en el dashboard.
  - Este método verifica si el ID de un botón de interruptor comienza con 'access', lo que indica que es el botón 'Aplicar' o 'Access' para un usuario.
  - Cuando se detecta una pulsación en uno de estos botones:
    - Extrae el ID del usuario: El método divide el ID del botón (por ejemplo, access-user ID) y toma la segunda parte (el índice 1) como el ID del usuario.
    - Lee el estado de los interruptores: Luego, el método lee el estado actual de los interruptores de permisos de lectura y escritura asociados a ese usuario específico, identificándolos por sus IDs (por ejemplo, read user ID y write user ID).
    - Envía la Solicitud POST: Finalmente, el método JavaScript envía una solicitud POST a la aplicación Flask en el servidor. La solicitud se formatea como 'grant - user ID - read state - right stick', donde 'right stick' se refiere al estado del interruptor de escritura. Esta acción se realiza utilizando una función similar a send event, que se usa para enviar solicitudes POST al servidor.
- 3. El Endpoint en la Aplicación Flask (Lado del Servidor):
  - La aplicación Flask debe tener un endpoint configurado para recibir esta solicitud POST de 'grant user ID read and write'.
  - Verificación de Administrador: Es crucial que la aplicación Flask verifique que la solicitud proviene de un usuario administrador antes de procesarla. Si no es un administrador, el servidor enviará una respuesta de 'acceso denegado'.
  - Acciones del Servidor (si se concede el acceso): Si la verificación de administrador es exitosa, el servidor realiza dos acciones principales:
    - 1. Almacena los nuevos permisos: Guarda los permisos de lectura y escritura del usuario en la base de datos.
    - 2. Llama al servidor PubNub: Contacta al servidor PubNub para otorgar o revocar el acceso de lectura y escritura en tiempo real a ese usuario específico.
  - Paso Previo: Es importante señalar que la concesión de estos permisos de lectura y escritura es un segundo paso. El primer paso es generar la clave de autorización para ese usuario específico y almacenarla en la base de datos.

En resumen, el método JavaScript en main.js es el motor que traduce la interacción del administrador con el botón 'Aplicar' en una solicitud estructurada enviada al servidor Flask. Esta interacción bidireccional cliente-servidor, con su correspondiente validación y procesamiento en el backend, asegura que los cambios de permisos de los usuarios se apliquen de manera segura y en tiempo real.

ESCUCHA CLICKS EN BOTONES CON ID QUE EMPIEZAN POR 'XS'

En el contexto más amplio del Método JavaScript en main.js, el aspecto de escuchar clics en botones con ID que empieza por 'XS' es fundamental para la interacción de los administradores con el panel de control, específicamente para la gestión de permisos de usuario. Aquí se detalla lo que se dice al respecto:

- 1. Propósito del Método JavaScript en main.js:
  - El archivo main.js contiene un método diseñado para escuchar (listen on) cualquier botón de interruptor ('switch button') que se encuentre en el dashboard.
  - Este método es esencial para capturar las interacciones del usuario con los controles del panel.
- 2. Identificación de Botones Relevantes (ID que empieza por 'XS'):
  - Para identificar el botón 'Aplicar Cambios' de entre todos los botones del dashboard, el método JavaScript verifica si el ID de un botón de interruptor 'starts with XS'.
  - A pesar de la fonética 'XS', este identificador se refiere al botón 'Aplicar Cambios', cuyo ID es access user ID. La intención es detectar el botón cuyo ID comienza con 'access'.
- 3. Funcionalidad al Detectar un Clic:
  - Una vez que el método JavaScript detecta que se ha presionado un botón cuyo ID comienza con 'XS' (es decir, el botón 'Aplicar Cambios' para un usuario específico):
    - Extrae el ID del usuario: El método divide el ID del botón (por ejemplo, access-user ID) y toma la segunda parte (el índice 1) como el ID del usuario.
    - Lee el estado de los interruptores: Procede a leer el estado actual de los botones de tipo interruptor de lectura (read user ID) y escritura (write user ID) asociados a ese mismo usuario.
    - Envía la solicitud POST: Finalmente, utiliza una función similar a send event para enviar una solicitud POST a la aplicación Flask en el servidor. Esta solicitud incluye el ID del usuario, el estado de lectura (read state) y el estado de escritura (right stick), siguiendo el formato `'grant - user ID - read state - right stick'`.

En resumen, la escucha de clics en botones con IDs que empiezan por 'XS' es el mecanismo que el método JavaScript en main.js utiliza para identificar cuándo un administrador desea aplicar cambios de permisos a un usuario. Esta detección es el primer paso en un flujo cliente-servidor que culminará en la actualización de los permisos en la base de datos y la plataforma PubNub en tiempo real, tras la validación de que la solicitud proviene de un usuario administrador.

EXTRAE USER ID DEL ID DEL BOTóN

En el contexto más amplio del Método JavaScript en main.js, la extracción del ID del usuario del ID del botón es un paso fundamental para identificar al usuario específico cuyos permisos se van a modificar en el panel de control del administrador. Aquí se detalla lo que se dice sobre este proceso:

- 1. Propósito del Método JavaScript en main.js:
  - El archivo main.js contiene un método diseñado para escuchar los clics en cualquier botón de tipo interruptor ('switch button') que se encuentre en el dashboard.
  - Este método es esencial para capturar las interacciones del administrador cuando desea modificar los permisos de los usuarios en línea.
- 2. Identificación del Botón 'Aplicar Cambios':
  - Para saber qué botón ha sido pulsado, el método JavaScript verifica si el ID de un botón de interruptor 'starts with XS'. Aunque se menciona 'XS', en el contexto de la interfaz, se refiere al botón 'Aplicar Cambios' cuyo ID es access user ID.
- 3. Proceso de Extracción del ID del Usuario:
  - Una vez que el método JavaScript detecta un clic en un botón cuyo ID comienza con access (o XS), procede a dividir este ID en dos partes usando el guion - como separador.
  - Luego, toma la segunda parte (el índice 1) de este ID dividido, y esta parte es el ID del usuario.
  - Por ejemplo, si el ID del botón es access-user ID, el método lo dividiría en ['access', 'user ID'] y tomaría 'user ID' como el identificador del usuario.
- 4. Uso del ID del Usuario Extraído:

- Una vez que el ID del usuario ha sido extraído, el método JavaScript lo utiliza junto con el estado actual de los interruptores de permisos de lectura y escritura (read state y right stick) de ese usuario.
- Toda esta información se envía en una solicitud POST a la aplicación Flask en el servidor, con un formato como 'grant - user ID - read state - right stick'. Esta solicitud activa el endpoint en el servidor para procesar los cambios de permisos.

En resumen, la extracción del ID del usuario del ID del botón por parte del método JavaScript en main.js es un mecanismo inteligente y dinámico. Permite al sistema identificar con precisión a qué usuario corresponden los cambios de permisos deseados por el administrador, lo cual es vital para una gestión efectiva y segura de los accesos en tiempo real.

LEE ESTADO DE BOTONES DE LECTURA ESCRITURA

En el contexto más amplio del Método JavaScript en main.js, la acción de leer el estado de los botones de lectura y escritura es una función crítica que permite al sistema capturar las preferencias de permisos establecidas por el administrador para un usuario en línea específico. Aquí se detalla lo que se dice sobre este proceso:

- 1. Propósito del Método JavaScript en main.js:
  - El archivo main.js contiene un método diseñado para escuchar los clics en cualquier botón de tipo interruptor ('switch button') que se encuentre en el dashboard.
  - Este método es esencial para manejar las interacciones de los administradores con el panel de control, especialmente cuando desean modificar los permisos de los usuarios en línea.
- 2. Activación de la Lectura de Estados:
  - La lectura de los estados de los botones de lectura y escritura se activa cuando el método JavaScript detecta un clic en el botón 'Aplicar Cambios'.
  - Este botón 'Aplicar Cambios' se identifica porque su ID comienza con 'XS' (que en la práctica se refiere a IDs como access user ID).
- 3. Proceso de Lectura de los Estados:
  - Después de identificar el botón 'Aplicar Cambios' y extraer el ID del usuario asociado, el método JavaScript procede a leer el estado del interruptor de lectura y del interruptor de escritura.
  - Para hacer esto, se especifica el ID de cada uno de estos interruptores (por ejemplo, read user ID y write user ID), lo que permite al JavaScript obtener su estado actual (si están 'checked' o 'unchecked').
  - Los botones de lectura y escritura son interruptores que se configuran inicialmente como 'checked' (activos), pero su estado puede ser modificado por el administrador. El estado real ('checked' o 'unchecked') se determina a partir de los datos del servidor (índice 2 para lectura y 3 para escritura del registro del usuario) cuando se carga la página.
- 4. Uso de los Estados Leídos:
  - Una vez que se han leído los estados de los botones de lectura (read state) y escritura (right stick), esta información, junto con el ID del usuario extraído, se empaca en una solicitud POST.
  - Esta solicitud se envía a la aplicación Flask en el servidor con un formato específico: `grant - user ID - read st`
  - El servidor utiliza esta información para almacenar los nuevos permisos de lectura y escritura del usuario en la base de datos y para llamar al servidor PubNub con el fin de otorgar o revocar el acceso en tiempo real a ese usuario específico.

En resumen, la capacidad del método JavaScript en main.js para leer el estado de los botones de lectura y escritura es fundamental. Es el mecanismo mediante el cual la interfaz de usuario traduce la intención del administrador en datos procesables que luego se envían al servidor para la gestión y aplicación en tiempo real de los permisos de acceso de los usuarios.

ENVIA SOLISITUD POST AL SERVIDOR

En el contexto más amplio del Método JavaScript en main.js, la acción de enviar una solicitud POST al servidor con el formato `grant-user ID-read state-write state` es el paso culminante que traduce la interacción del administrador en la interfaz de usuario a una acción concreta en el backend para gestionar los permisos. Aquí se detalla lo que se dice sobre este envío de solicitud POST:

- 1. Activación por el Botón 'Aplicar Cambios':
  - El método JavaScript en main.js está diseñado para escuchar cualquier botón en el dashboard.
  - Específicamente, verifica si el ID de un botón de tipo interruptor 'starts with XS', lo que, como hemos visto, se refiere al botón 'Aplicar Cambios' (ID: access user ID) presente en cada fila de usuario para aplicar las modificaciones de permisos.
- 2. Preparación de la Solicitud por JavaScript:
  - Una vez que el método detecta un clic en el botón 'Aplicar Cambios':
    - Extracción del ID del usuario: Divide el ID del botón pulsado (por ejemplo, access-user ID) utilizando el guion - como separador y toma la segunda parte (el índice 1) como el ID del usuario.
    - Lectura de los estados de permisos: Luego, lee el estado actual del interruptor de lectura (read state) y el estado del interruptor de escritura (right stick) para ese usuario específico, identificándolos por sus respectivos IDs (por ejemplo, read user ID y write user ID).
  - Estos datos (ID de usuario, estado de lectura y estado de escritura) son cruciales para formar la solicitud.
- 3. Formato y Envío de la Solicitud POST:
  - El método JavaScript envía la solicitud al servidor con el formato específico: `grant - user ID - read state -`
  - Esta acción se realiza mediante una función similar a send event, que se utilizó en partes anteriores del proyecto para enviar solicitudes POST al servidor.
- 4. Recepción y Procesamiento en el Servidor (Aplicación Flask):
  - La aplicación Flask en el servidor está equipada con un endpoint específico para recibir esta solicitud POST, que se describe como grant user ID read and write.
  - Verificación de Administrador: Un paso fundamental en el servidor es verificar que la solicitud grant provenga de un usuario administrador. Si la solicitud no es de un administrador, el servidor enviará una respuesta de 'acceso denegado'.
  - Acciones del Servidor (si la verificación es exitosa): Si la solicitud es válida y proviene de un administrador, el servidor realiza dos acciones clave:
    - 1. Almacena los nuevos permisos de lectura y escritura del usuario en la base de datos.
    - 2. Llama al servidor PubNub para otorgar o revocar el acceso de lectura y escritura en tiempo real a ese usuario específico.
  - Paso Previo: Es importante señalar que la concesión de estos permisos es un segundo paso. El primer paso es generar y almacenar una clave de autorización para el usuario específico en la base de datos.

6.6.2. Endpoint en FLASK para Recibir Solisitud grant user-id read and write

En el contexto más amplio del Endpoint para el Botón 'Aplicar', el Endpoint Flask para Recibir Solicitud es el componente en el lado del servidor responsable de procesar los cambios de permisos de usuario iniciados por un administrador a través de la interfaz de usuario en index.html. Esto es lo que se dice sobre este endpoint:

- 1. Propósito y Origen de la Solicitud:

- Cuando un administrador interactúa con el botón 'Aplicar Cambios' (cuyo ID comienza con access user ID o XS) en el panel de control, el método JavaScript en main.js envía una solicitud al servidor.
- Este botón permite cambiar los permisos de lectura y escritura de un usuario específico en tiempo real.
- La solicitud es de tipo POST, y su formato es grant - user ID - read state - right stick (donde right stick representa el estado de escritura). Se describe también como una solicitud grant user ID read and write.

- 2. Configuración del Endpoint en la Aplicación Flask:
  - La aplicación Flask debe tener un endpoint configurado para recibir esta solicitud POST específica.
  - Este endpoint está diseñado para manejar la lógica de otorgar permisos.
- 3. Verificación de Seguridad (Administrador):
  - Un paso crucial e indispensable en el lado del servidor es verificar que la solicitud grant provenga de un usuario administrador. Esta es una medida de seguridad fundamental para evitar que usuarios no autorizados modifiquen permisos.
  - Si la solicitud no es de un administrador, el servidor enviará una respuesta de 'acceso denegado'.
  - Si la verificación es exitosa y la solicitud proviene de un administrador, el servidor procede con las acciones, enviando implícitamente una respuesta de 'acceso concedido' (access granted).
- 4. Acciones del Servidor tras la Verificación Exitosa:
  - Una vez que se confirma que la solicitud es válida y proviene de un administrador, el servidor Flask realiza dos acciones principales:
    - 1. Almacenar los nuevos permisos: Guarda los permisos de lectura y escritura (read y write access) del usuario específico en la base de datos.
    - 2. Llamar al servidor PubNub: Contacta al servidor PubNub para otorgar o revocar el acceso de lectura y escritura en tiempo real a ese usuario específico. PubNub es una de las principales tecnologías utilizadas en el proyecto para protocolos de comunicación en tiempo real.
- 5. Paso Previo a la Concesión de Permisos:
  - Es importante destacar que la concesión de estos permisos de lectura y escritura es un segundo paso. El primer paso que lo precede es generar la clave de autorización para ese usuario específico y almacenarla en la base de datos.

En resumen, el Endpoint Flask para Recibir Solicitud actúa como el punto de entrada seguro en el servidor para la gestión de permisos. Recibe solicitudes POST del cliente, realiza una estricta verificación de administrador y, si es válida, actualiza los permisos del usuario en la base de datos y en la plataforma de comunicación en tiempo real (PubNub), asegurando una gestión robusta y segura de los accesos en el entorno IoT.

Verificar si la Solisitud Procede de un Usuario Administrador

En el contexto más amplio del Endpoint Flask para Recibir Solicitud, la acción de verificar si la solicitud procede de un usuario administrador es una medida de seguridad crítica e indispensable para asegurar la integridad y el control de los permisos de los usuarios dentro del panel de control del servidor IoT. Aquí se detalla lo que se dice sobre este proceso:

- 1. Rol del Endpoint Flask:
  - La aplicación Flask en el servidor tiene un endpoint específicamente configurado para recibir solicitudes POST que provienen del lado del cliente (main.js).
  - Estas solicitudes se activan cuando un administrador hace clic en el botón 'Aplicar Cambios' (ID: access user ID) en la interfaz de usuario de index.html para modificar los permisos de un usuario en línea.

- El formato de la solicitud es `grant - user ID - read state - right stick`, indicando el deseo de otorgar (o revocar) permisos de lectura y escritura a un usuario específico.

- 2. La Verificación de Seguridad del Administrador:
  - Una vez que la solicitud grant user ID read and write es recibida por la aplicación Flask, un paso crucial es 'verificar una vez más si esta solicitud grant proviene de un usuario administrador'.
  - Esta verificación es fundamental porque el panel de control y sus funcionalidades de gestión de permisos están diseñados para ser visibles y operables únicamente por usuarios administradores. Existe una sentencia if en el HTML que controla la visibilidad de este panel.
- 3. Resultados de la Verificación:
  - Acceso Denegado: Si el servidor determina que la solicitud no proviene de un usuario administrador, enviará una respuesta al solicitante con el mensaje 'acceso denegado' (access denied). Esto previene que usuarios no autorizados realicen cambios en los permisos.
  - Acceso Concedido: Si la verificación es exitosa y se confirma que la solicitud es válida y ha sido iniciada por un usuario administrador, el servidor procede a ejecutar las acciones solicitadas y envía implícitamente una respuesta de 'acceso concedido' (access granted).
- 4. Acciones Posteriores a la Verificación Exitosa:
  - Una vez que se ha verificado que la solicitud procede de un administrador, el servidor Flask realiza dos acciones principales:
    - 1. Almacenar los nuevos permisos: Guarda los permisos de lectura y escritura del usuario específico en la base de datos.
    - 2. Llamar al servidor PubNub: Contacta al servidor PubNub para otorgar o revocar el acceso de lectura y escritura en tiempo real a ese usuario o dispositivo específico.
  - Es importante recordar que la concesión de estos permisos es un segundo paso; el primer paso es generar la clave de autorización para el usuario y almacenarla en la base de datos.

En síntesis, la verificación de que la solicitud procede de un usuario administrador dentro del Endpoint Flask es una piedra angular de la seguridad del sistema. Actúa como un guardián, permitiendo que solo los usuarios autorizados realicen cambios sensibles en los permisos de otros usuarios, asegurando así un control robusto y seguro en el entorno del Internet de las Cosas.

Responder 'Acceso Denegado' o 'Acceso Concedido'

En el contexto más amplio del Endpoint Flask para Recibir Solicitud, la acción de responder 'Acceso Denegado' o 'Acceso Concedido' es el resultado directo y crucial de la verificación de seguridad que el servidor realiza al procesar las solicitudes de cambio de permisos. Aquí se detalla lo que se dice sobre estas respuestas:

- 1. Contexto del Endpoint Flask:
  - La aplicación Flask en el servidor está configurada con un endpoint para recibir solicitudes POST provenientes del código JavaScript en main.js.
  - Estas solicitudes se activan cuando un administrador interactúa con el botón 'Aplicar Cambios' (cuyo ID comienza con access user ID) en el panel de control del index.html.
  - La solicitud tiene el formato grant - user ID - read state - right stick o grant user ID read and write, y su propósito es modificar los permisos de lectura y escritura de un usuario en línea.
- 2. La Verificación de Seguridad:
  - Un paso crucial e indispensable en el lado del servidor es 'verificar una vez más si esta solicitud grant proviene de un usuario administrador'. Esta verificación es fundamental para la seguridad del sistema, ya que el panel de control para la gestión de permisos está diseñado para ser accesible y operable solo por administradores.
- 3. Respuestas Basadas en la Verificación:

- 'Acceso Denegado' (access denied): Si el servidor determina que la solicitud no procede de un usuario administrador, enviará una respuesta al solicitante (es decir, al cliente que envió la solicitud) con el mensaje 'acceso denegado'. Esto sirve para prevenir que usuarios no autorizados puedan realizar cambios sensibles en los permisos.
- 'Acceso Concedido' (access granted): Si la verificación es exitosa, lo que significa que la solicitud es válida y proviene de un usuario administrador, el servidor procede a ejecutar las acciones solicitadas. Se menciona que si 'todo sale bien' (if everything goes well), se enviará la respuesta de 'acceso concedido' (access granted). Esta respuesta implícita o explícita señala que la operación ha sido autorizada y se procederá con la gestión de permisos.

- 4. Manejo de la Respuesta en el Lado del Cliente (JavaScript):
  - El main.js en el lado del cliente está preparado para recibir esta respuesta del servidor.
  - Una vez recibida, el JavaScript extrae el JSON de la respuesta y verifica si contiene la clave 'access'. Si el valor asociado a 'access' es granted (es decir, 'acceso concedido'), se restablece la suscripción al canal nuevamente. Esto sugiere que, tras la confirmación de los permisos, podría ser necesario actualizar las suscripciones en tiempo real para reflejar los cambios. También se recomienda modificar el método de suscripción para que indique si la suscripción fue exitosa.
- 5. Acciones Posteriores a un 'Acceso Concedido':
  - Si la verificación es exitosa y se concede el acceso, el servidor Flask realiza dos acciones principales:
    - 1. Almacenar los nuevos permisos de lectura y escritura del usuario en la base de datos.
    - 2. Llamar al servidor PubNub para otorgar o revocar el acceso de lectura y escritura a ese usuario específico en tiempo real.
  - Cabe recordar que la concesión de estos permisos es un segundo paso, ya que el primer paso es generar la clave de autorización para el usuario y almacenarla en la base de datos.

En conclusión, el Endpoint Flask utiliza la respuesta de 'Acceso Denegado' o 'Acceso Concedido' como un mecanismo de retroalimentación esencial para comunicar el resultado de su verificación de seguridad al cliente. Esta funcionalidad no solo protege el sistema de accesos no autorizados, sino que también permite al lado del cliente reaccionar adecuadamente a la confirmación de los cambios de permisos, manteniendo la coherencia y la seguridad en la gestión en tiempo real de los dispositivos IoT.

Almacenar Permisos de Lectura/Escritura en la Base de Datos

En el contexto más amplio del Endpoint Flask para Recibir Solicitud, la acción de almacenar los permisos de lectura y escritura en la base de datos es una función principal y esencial que el servidor realiza después de verificar la validez y la autorización de una solicitud de cambio de permisos. Aquí se detalla lo que se dice sobre este proceso:

- 1. Activación de la Solicitud y Rol del Endpoint Flask:
  - El método JavaScript en main.js envía una solicitud POST a la aplicación Flask en el servidor cuando un administrador hace clic en el botón 'Aplicar Cambios' (cuyo ID comienza con access user ID) en el panel de control del index.html.
  - Esta solicitud, con un formato como grant - user ID - read state - right stick o grant user ID read and write, tiene como objetivo modificar los permisos de lectura y escritura de un usuario en línea.
  - El endpoint Flask está configurado específicamente para recibir y procesar estas solicitudes de otorgamiento de permisos.
- 2. Verificación de Seguridad Previa al Almacenamiento:
  - Antes de almacenar cualquier cambio de permisos, el servidor realiza una verificación crucial: 'comprobar una vez más si esta solicitud grant proviene de un usuario administrador'.
  - Esta medida de seguridad es indispensable, ya que el panel de control de permisos es visible y operable solo para usuarios administradores.

- Si la solicitud no es de un administrador, el servidor responderá con 'acceso denegado'. Solo si la verificación es exitosa ('todo sale bien' y se obtiene 'acceso concedido') se procede con el almacenamiento.

- 3. Almacenamiento de Permisos en la Base de Datos:
  - Una vez que la solicitud ha sido validada y confirmada como proveniente de un usuario administrador, una de las dos acciones principales que el servidor Flask debe realizar es 'almacenar los permisos de lectura y escritura del usuario en la base de datos'.
  - Esto implica registrar el estado actual (otorgado o revocado) de los permisos de lectura y escritura para el user ID especificado en la solicitud.

- 4. Contexto de los Pasos de Almacenamiento:
  - Las fuentes aclaran que el 'otorgar permisos de lectura y escritura es un segundo paso'.
  - El primer paso que precede a esto es 'generar la clave de autorización para ese usuario específico y almacenarla en la base de datos'. Esto sugiere que la base de datos no solo guarda los estados de lectura y escritura, sino también una clave fundamental para la autorización del usuario.

- 5. Integración con PubNub:
  - Además de almacenar los permisos en la base de datos, el servidor también realiza la segunda acción principal: 'llamar al servidor PubNub para otorgar acceso de lectura y escritura a este usuario específico' en tiempo real. Esto asegura que los cambios no solo se persistan, sino que también se apliquen inmediatamente en la plataforma de comunicación.

En resumen, la capacidad de almacenar los permisos de lectura y escritura en la base de datos es una funcionalidad central del Endpoint Flask para Recibir Solicitud. Este proceso se lleva a cabo de forma segura, precedido por una estricta verificación de administrador y como parte de una secuencia de dos pasos que incluye la generación de claves de autorización, asegurando así una gestión robusta y persistente de los accesos de usuario en el sistema IoT.

Llamar a Pub/Nub para Conceder Acceso al Usuario

En el contexto más amplio del Endpoint Flask para Recibir Solicitud, la acción de llamar a PubNub para conceder acceso al usuario es una de las dos funciones principales y críticas que el servidor realiza después de validar y autorizar una solicitud de cambio de permisos. Esta acción asegura que los cambios se apliquen en tiempo real a la plataforma de comunicación. Aquí se detalla lo que se dice sobre este proceso:

- 1. Activación de la Solicitud y Rol del Endpoint Flask:
  - El método JavaScript en main.js envía una solicitud POST a la aplicación Flask en el servidor cuando un administrador hace clic en el botón 'Aplicar Cambios' (cuyo ID comienza con access user ID) en el panel de control del index.html.
  - Esta solicitud, con un formato como grant - user ID - read state - right stick o grant user ID read and write, tiene como objetivo modificar los permisos de lectura y escritura de un usuario en línea.
  - El endpoint Flask está configurado específicamente para recibir y procesar estas solicitudes.

- 2. Verificación de Seguridad Previa a la Llamada a PubNub:
  - Antes de realizar cualquier acción, el servidor realiza una verificación crucial: 'comprobar una vez más si esta solicitud grant proviene de un usuario administrador'.
  - Esta medida de seguridad es indispensable, ya que el panel de control de permisos es visible y operable solo para usuarios administradores.
  - Si la solicitud no es de un administrador, el servidor responderá con 'acceso denegado'. Solo si la verificación es exitosa ('todo sale bien' y se obtiene 'acceso concedido') se procede con la llamada a PubNub.

- 3. Llamada a PubNub para Conceder Acceso:

- Una vez que la solicitud ha sido validada y confirmada como proveniente de un usuario administrador, una de las dos acciones principales que el servidor Flask debe realizar es 'llamar al servidor PubNub para conceder acceso de lectura y escritura a este usuario específico'.
- Esto garantiza que los cambios de permisos no solo se persistan en la base de datos, sino que también se apliquen inmediatamente en la plataforma de comunicación en tiempo real.

- 4. Propósito y Beneficios de Usar PubNub:
  - PubNub es una de las principales tecnologías utilizadas en el proyecto para los protocolos de comunicación en tiempo real.
  - Se utiliza la funcionalidad de PubNub Access Manager para que los usuarios administradores puedan 'otorgar acceso de lectura y escritura en tiempo real a todos los usuarios que no son administradores y a los dispositivos'.
  - Esto permite una 'comunicación en tiempo real, fuerte, segura y escalable' para el ecosistema IoT. PubNub es una solución para protocolos de comunicación en tiempo real y ligeros.
- 5. Contexto de los Pasos:
  - Las fuentes aclaran que el 'otorgar permisos de lectura y escritura (a través de PubNub) es un segundo paso'.
  - El primer paso que precede a esto es 'generar la clave de autorización para ese usuario específico y almacenarla en la base de datos'. Esto indica que la autorización en PubNub se basa en una clave que ya debe existir.

En resumen, la llamada a PubNub para conceder acceso al usuario es una función central del Endpoint Flask, que se ejecuta de forma segura después de una verificación de administrador. Esta acción es vital para aplicar los cambios de permisos de lectura y escritura en tiempo real a través de la plataforma PubNub, facilitando así una gestión dinámica y segura de los accesos de usuarios y dispositivos en el sistema IoT.

6.6.3. Manejar Respuesta en main.js

En el contexto más amplio del Endpoint Flask para el Botón 'Aplicar', el manejo de la respuesta en main.js es el paso final en el lado del cliente, donde se procesa la confirmación del servidor sobre los cambios de permisos solicitados, permitiendo que la interfaz de usuario reaccione adecuadamente. Aquí se detalla lo que se dice sobre cómo main.js maneja esta respuesta:

- 1. Contexto de la Solicitud y el Endpoint:
  - Recordemos que el método JavaScript en main.js es el encargado de enviar una solicitud POST a la aplicación Flask en el servidor. Esto ocurre cuando un administrador presiona el botón 'Aplicar Cambios' (cuyo ID comienza con access user ID) para modificar los permisos de lectura y escritura de un usuario en línea.
  - El Endpoint Flask recibe esta solicitud (grant user ID read and write), realiza una verificación crucial para asegurar que proviene de un usuario administrador y, si todo es válido, procede a almacenar los permisos en la base de datos y a llamar a PubNub para conceder el acceso en tiempo real.
- 2. Recepción de la Respuesta en main.js:
  - El archivo main.js está preparado para recibir la respuesta que el servidor Flask envía después de procesar la solicitud.
  - El servidor envía una respuesta que puede ser 'acceso denegado' si la solicitud no proviene de un administrador, o 'acceso concedido' si la verificación es exitosa y se procede con los cambios.
- 3. Procesamiento de la Respuesta JSON:
  - Una vez recibida la respuesta, el código JavaScript en main.js debe extraer el JSON de esta respuesta.

- Después de extraer el JSON, se verifica si contiene la clave 'access'. Esta clave es la que indica el estado de la autorización.

- 4. Acción al Recibir 'Acceso Concedido':
  - Si el valor asociado a la clave 'access' es 'granted' (es decir, 'acceso concedido'), entonces main.js procede a 'restablecer la suscripción al canal nuevamente' (reset scribe on the channel again). Esto sugiere que, una vez confirmados los cambios de permisos, es posible que sea necesario reconfigurar las suscripciones en tiempo real para reflejar las nuevas autorizaciones del usuario en la plataforma de comunicación (probablemente PubNub).
- 5. Recomendación para el Método SUBSCRIBE:
  - Las fuentes también mencionan que 'es bueno modificar el método SUBSCRIBE para que nos diga si se ha suscrito con éxito o no' (it is also good to modify the SUBSCRIBE method to let us know whether it has subscribed successfully or not). Esta es una mejora sugerida para proporcionar una mejor retroalimentación sobre el éxito de la operación de suscripción después de que se conceden los permisos.

En resumen, el manejo de la respuesta en main.js es un componente reactivo y vital en el flujo de gestión de permisos. Se encarga de recibir, interpretar y actuar sobre la confirmación del servidor (especialmente el 'acceso concedido'), asegurando que la interfaz de usuario y las suscripciones en tiempo real se sincronicen con los cambios de permisos aplicados por el administrador.

Extraer JSON de la Respuesta

En el contexto más amplio del Manejo de Respuesta en main.js, la acción de extraer el JSON de la respuesta es un paso fundamental que permite al lado del cliente interpretar la decisión del servidor sobre la solicitud de cambio de permisos y actuar en consecuencia. Aquí se detalla lo que se dice sobre este proceso:

- 1. Ciclo de Comunicación Cliente-Servidor:
  - Previamente, el método JavaScript en main.js envía una solicitud POST al endpoint de la aplicación Flask en el servidor. Esta solicitud se activa cuando un administrador presiona el botón 'Aplicar Cambios' (ID: access user ID) en el index.html para modificar los permisos de lectura y escritura de un usuario.
  - El servidor Flask procesa esta solicitud, que incluye una verificación crucial para determinar si proviene de un usuario administrador. Tras esta verificación y la aplicación de los cambios (almacenamiento en la base de datos y llamada a PubNub), el servidor envía una respuesta de vuelta al cliente.
- 2. Recepción y Extracción del JSON en main.js:
  - El código JavaScript en main.js está diseñado para recibir esta respuesta del servidor.
  - El siguiente paso es 'extraer el JSON' (extract the JSON) de la respuesta recibida. Esto es esencial porque la respuesta del servidor probablemente está formateada en JSON para facilitar el intercambio estructurado de datos.
- 3. Interpretación de la Respuesta para 'Acceso Concedido' o 'Denegado':
  - Una vez extraído el JSON, main.js procede a 'verificar si tiene la clave 'access" (check whether it has the key as access).
  - Esta clave 'access' es la que indica el estado de la autorización otorgada por el servidor. Si el valor asociado a 'access' es 'granted' (es decir, 'acceso concedido'), significa que la solicitud de cambio de permisos fue validada y ejecutada con éxito por el servidor.
  - Si la verificación de administrador en el servidor fallara, la respuesta contendría un 'acceso denegado' (access denied). Aunque la fuente no detalla cómo main.js reacciona específicamente a 'acceso denegado', la estructura implica que solo se procede con acciones si el acceso es 'concedido'.
- 4. Acciones Posteriores a la Extracción Exitosa del JSON:

- Si el JSON indica que el 'access' fue 'granted', main.js realiza la acción de 'restablecer la suscripción al canal nuevamente' (reset scribe on the channel again). Esto sugiere que, una vez que los permisos se han modificado y confirmado, las suscripciones en tiempo real (posiblemente a través de PubNub) deben actualizarse para reflejar las nuevas capacidades del usuario.
- Además, se recomienda modificar el método SUBSCRIBE para que indique si la suscripción fue exitosa o no, lo que mejoraría la retroalimentación sobre el proceso.

En resumen, la extracción del JSON de la respuesta en main.js es un paso crucial en el ciclo de gestión de permisos. Permite al cliente interpretar de manera estructurada el resultado de la operación del servidor (si el acceso fue concedido o denegado) y, en caso de éxito, realizar las acciones necesarias en la interfaz de usuario y en la comunicación en tiempo real para mantener la coherencia y la funcionalidad del sistema.

Si 'access' es 'granted', re-suscribirse al canal

En el contexto más amplio del Manejo de Respuesta en main.js, la acción de re-suscribirse al canal si 'access' es 'granted' es un paso crucial y consecuente en el lado del cliente, que asegura que el sistema se adapte dinámicamente a los cambios de permisos del usuario confirmados por el servidor. Aquí se detalla lo que se dice sobre este proceso:

- 1. Contexto de la Interacción y el Rol de main.js:
  - El método JavaScript en main.js es el encargado de iniciar una solicitud POST al servidor Flask. Esto sucede cuando un administrador interactúa con el botón 'Aplicar Cambios' (cuyo ID empieza con access user ID) en el panel de control del index.html para modificar los permisos de lectura y escritura de un usuario en línea.
  - El servidor Flask procesa esta solicitud, realizando una verificación de seguridad crucial para asegurar que proviene de un usuario administrador. Tras la verificación y la aplicación de los cambios (almacenamiento en la base de datos y llamada a PubNub), el servidor envía una respuesta al cliente.
- 2. Recepción y Procesamiento de la Respuesta en main.js:
  - main.js está diseñado para recibir esta respuesta del servidor.
  - El primer paso es extraer el JSON de la respuesta. Este formato estructurado es esencial para que el cliente pueda interpretar los datos enviados por el servidor.
  - Luego, main.js verifica si el JSON contiene la clave 'access'. Esta clave es la que indica el estado de la autorización o el resultado de la operación en el servidor.
- 3. Condición para Re-suscripción ('access' es 'granted'):
  - Si el valor asociado a la clave 'access' es 'granted' (es decir, 'acceso concedido'), esto significa que la solicitud de cambio de permisos fue validada, ejecutada con éxito por el servidor, y que la verificación de administrador fue positiva.
  - En este caso de éxito, el código JavaScript en main.js procederá a 'restablecer la suscripción al canal nuevamente' (reset scribe on the channel again).
- 4. Implicaciones y Propósito de la Re-suscripción:
  - Esta acción sugiere que, una vez que los permisos de un usuario han sido modificados y confirmados por el servidor, las suscripciones en tiempo real (probablemente a través de PubNub, que se usa para comunicación en tiempo real y protocolos ligeros) deben actualizarse.
  - Al re-suscribirse, el sistema asegura que el usuario (o el dispositivo) reciba los datos de acuerdo con sus nuevos permisos de lectura y escritura, manteniendo la coherencia entre el backend y el frontend en un ecosistema IoT en tiempo real. PubNub es clave para otorgar acceso de lectura y escritura en tiempo real a usuarios y dispositivos.
- 5. Mejora Sugerida para el Método SUBSCRIBE:
  - Las fuentes también mencionan que 'es bueno modificar el método SUBSCRIBE para que nos diga si se ha suscrito con éxito o no'. Esto indica la importancia de tener una retroalimentación clara sobre el resultado de la operación de suscripción, lo cual es vital para el diagnóstico y la estabilidad del sistema.

En resumen, la lógica en main.js para re-suscribirse al canal si 'access' es 'granted' es un componente fundamental para la reactividad y la coherencia del sistema de gestión de permisos en tiempo real. Permite que la interfaz de usuario y los canales de comunicación se ajusten inmediatamente después de que el servidor haya confirmado y aplicado los cambios de acceso, garantizando que el ecosistema IoT funcione de manera segura y actualizada.

Modificar Método 'SUBSCRIBE para Confirmación de éxito

En el contexto más amplio del Manejo de Respuesta en main.js, la recomendación de modificar el método SUBSCRIBE para confirmación de éxito es una mejora sugerida para aumentar la robustez y la retroalimentación del sistema después de que se han aplicado cambios en los permisos de los usuarios. Aquí se detalla lo que se dice sobre esta modificación:

- 1. Contexto de la Interacción Cliente-Servidor en main.js:
  - El método JavaScript en main.js es el punto de inicio para enviar una solicitud POST al servidor Flask. Esta solicitud se dispara cuando un administrador presiona el botón 'Aplicar Cambios' (cuyo ID comienza con access user ID) en el index.html para modificar los permisos de lectura y escritura de un usuario en línea.
  - El servidor Flask procesa esta solicitud, realiza una verificación de seguridad crucial para asegurarse de que proviene de un usuario administrador y, si todo es válido, procede a almacenar los permisos en la base de datos y a llamar a PubNub para conceder el acceso en tiempo real.
  - Posteriormente, el servidor envía una respuesta al cliente (main.js).
- 2. Manejo de la Respuesta en main.js y la Re-suscripción:
  - En main.js, una vez que se recibe la respuesta del servidor, el código JavaScript extrae el JSON de esta respuesta y verifica si contiene la clave 'access'.
  - Si el valor asociado a 'access' es 'granted' (es decir, 'acceso concedido'), esto indica que la solicitud de cambio de permisos fue validada y ejecutada con éxito por el servidor.
  - En este escenario de éxito, main.js realiza la acción de 'restablecer la suscripción al canal nuevamente' (reset scribe on the channel again). Esto es crucial para que el usuario (o dispositivo) reciba actualizaciones de datos de acuerdo con sus nuevos permisos, especialmente en un entorno de comunicación en tiempo real como el proporcionado por PubNub.
- 3. La Recomendación de Modificar el Método SUBSCRIBE:
  - Directamente relacionado con la acción de re-suscripción, se sugiere que 'es bueno modificar el método SUBSCRIBE para que nos diga si se ha suscrito con éxito o no' (it is also good to modify the SUBSCRIBE method to let us know whether it has subscribed successfully or not).
- 4. Propósito y Beneficios de la Modificación:
  - Esta modificación tiene como objetivo proporcionar una retroalimentación explícita sobre el resultado de la operación de suscripción.
  - En un sistema IoT que depende de la comunicación en tiempo real para el control y monitoreo, es vital saber si una suscripción (o re-suscripción) fue exitosa. Si la re-suscripción falla después de un cambio de permisos, el cliente podría no recibir las actualizaciones esperadas, lo que afectaría la funcionalidad del panel de control o del dispositivo.
  - Una confirmación de éxito en el método SUBSCRIBE permitiría al main.js manejar posibles fallos de re-suscripción, mejorando la fiabilidad y el diagnóstico del sistema.

En resumen, la sugerencia de modificar el método SUBSCRIBE para confirmar el éxito en el contexto del manejo de respuesta en main.js es una mejora de diseño propuesta para el lado del cliente. Su objetivo es asegurar que la re-suscripción a los canales de comunicación, que ocurre después de que el servidor ha concedido (o modificado) los permisos, se verifique adecuadamente, proporcionando una mayor certeza sobre la funcionalidad en tiempo real y la integridad del sistema IoT.

6.6.4. Requisito Previo: Generar Clave de Autorización y Almacenar en Base de Datos

En el contexto más amplio del Endpoint Flask para el Botón 'Aplicar', la generación de la clave de autorización para un usuario y su almacenamiento en la base de datos se presenta como un prerrequisito fundamental y el 'primer paso' en el proceso de gestión de permisos. Esto es lo que se dice al respecto:

- 1. Rol del Botón 'Aplicar' y la Solicitud al Endpoint Flask:
  - Cuando un usuario administrador interactúa con el botón 'Aplicar Cambios' (ID: access user ID) en el panel de control (index.html), el código JavaScript en main.js envía una solicitud POST al endpoint de la aplicación Flask en el servidor.
  - Esta solicitud (grant - user ID - read state - right stick o grant user ID read and write) tiene como objetivo modificar los permisos de lectura y escritura de un usuario específico en tiempo real.
- 2. La Generación y Almacenamiento de la Clave de Autorización como Primer Paso:
  - Las fuentes aclaran que 'otorgar permisos de lectura y escritura es un segundo paso'.
  - El 'primer paso que viene antes de esto es generar la clave de autorización para ese usuario específico y almacenarla en la base de datos'.
  - Esto indica que, antes de que el endpoint de Flask procese una solicitud para cambiar los permisos de lectura y escritura de un usuario a través del botón 'Aplicar', ya debe existir una clave de autorización generada para ese usuario y debe estar almacenada en la base de datos. Esta clave es esencial para el esquema de seguridad y acceso del sistema IoT.
- 3. Acciones del Endpoint Flask (Segundo Paso):
  - Una vez que el endpoint Flask recibe la solicitud POST del botón 'Aplicar' y después de haber verificado que la solicitud proviene de un usuario administrador, el servidor realiza dos acciones principales:
  - 1. Almacenar los nuevos permisos de lectura y escritura del usuario en la base de datos.
  - 2. Llamar al servidor PubNub para otorgar o revocar el acceso de lectura y escritura a ese usuario específico en tiempo real.
  - Estas dos acciones constituyen el 'segundo paso' del proceso, que se basa en la existencia previa de la clave de autorización.

En resumen, la generación de la clave de autorización para un usuario y su almacenamiento en la base de datos es un prerrequisito fundamental que debe cumplirse antes de que el endpoint Flask, activado por el botón 'Aplicar', pueda procesar y aplicar los cambios en los permisos de lectura y escritura de ese usuario. Este proceso de dos pasos garantiza un control de acceso robusto y seguro en el ecosistema IoT, donde la clave de autorización inicial sienta las bases para la gestión dinámica de permisos.

Bibliografía

[1] Python Software Foundation, "Python Language Reference." https://www.python.org, 2023. Accessed: 2023-10-27.

[2] Raspberry Pi Foundation, "Raspberry Pi Documentation." https://www.raspberrypi.org/documentation/, 2023. Accessed: 2023-10-27.

[3] Pocoo, "Flask - Web Development, One Drop at a Time." https://flask.palletsprojects.com/, 2023. Accessed: 2023-10-27.

[4] Amazon Web Services, Inc., "What is AWS?." https://aws.amazon.com/what-is-aws/, 2023. Accessed: 2023-10-27.

[5] PubNub Inc., "Real-time Communication APIs and Platform." https://www.pubnub.com/, 2023. Accessed: 2023-10-27.