



EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA



TECNOLÓGICO
NACIONAL DE MÉXICO®



Instituto Tecnológico de Chihuahua

Control de acceso para barreras vehiculares en el ITCh

INFORME TÉCNICO DE RESIDENCIA PROFESIONAL

Que presenta:

Oscar Alberto Valles Limas

Estudiante de la carrera de:

Ingeniería electrónica

Asesor interno:

<<Nombre Completo del/la asesora Interno(a)>>

Chihuahua, Chih., a junio de 2025.

Dedicatoria o Agradecimiento (opcional)

Índice

ÍNDICE DE CUADROS, GRÁFICAS Y FIGURAS.

INTRODUCCIÓN

CAPÍTULO I.

CARACTERIZACIÓN DEL ÁREA EN QUE PARTICIPÓ

(Este capítulo deberá tener un máximo de 10 páginas del documento.)

1.1 Datos Generales

1.2 Breve reseña histórica de la empresa

1.3 Organigrama de la empresa

1.4 Misión, Visión y Políticas

1.5 Productos y clientes

1.6 “Layout” (si aplica)

1.7 Premios y certificaciones

1.8 Relación de la empresa con la sociedad

CAPÍTULO II.

PLANTEAMIENTO DEL PROBLEMA O ÁREA DE OPORTUNIDAD

2.1. Caracterización del área en que realizó el proyecto

2.2. Antecedentes y definición del problema para la realización del proyecto de Residencia o bien, precisar el área de oportunidad de interés para la Organización.

2.3. Objetivos.

2.4. Justificación

CAPÍTULO III.

FUNDAMENTO TEÓRICO

CAPÍTULO IV.

DESARROLLO DEL PROYECTO

CAPÍTULO V.

ANÁLISIS DE RESULTADOS

RECOMENDACIONES

FUENTES DE INFORMACIÓN

ANEXOS

GLOSARIO

El índice general o tabla de contenido tiene como finalidad poder identificar en forma gráfica y lógica las partes que conforman el reporte técnico; por tal razón, es necesario estructurarlo cuidadosamente y con toda claridad para facilitar la ubicación de cada uno de los temas desarrollados.

Para redactar el índice general se procede de la siguiente manera:

Del lado izquierdo de la hoja se escriben los títulos de las diferentes partes que conforman el reporte y del lado derecho la página en la cual se inicia el desarrollo de cada uno de los temas y subtemas, en orden progresivo.

A partir de la introducción, la numeración se inicia con números arábigos, las hojas anteriores a ésta con números romanos en minúsculas.

Existen diferentes tipos de índice, se sugiere lo siguiente:

INDICE

Índice general

Índice de tablas

Índice de figuras

Índice de anexos

ÍNDICE DE CUADROS, GRÁFICAS Y FIGURAS.

Es una lista de las tablas, figuras y anexos. (Trabajar como en índice)

INTRODUCCIÓN

El presente informe técnico tiene como objetivo documentar el desarrollo, implementación y resultados obtenidos durante el periodo de residencia profesional llevado a cabo en el Instituto Tecnológico de Chihuahua. Este proyecto se centró en la integración de conocimientos adquiridos a lo largo de la carrera de Ingeniería Electrónica, aplicados en un entorno real mediante el diseño y desarrollo de un sistema de control de accesos vehiculares.

Durante el desarrollo del proyecto se abordaron diversas áreas técnicas, tales como la programación de microcontroladores, la implementación de protocolos de comunicación como MQTT, el uso de herramientas de software libre como Linux, y la gestión de contenedores mediante Docker. Asimismo, se hizo uso de plataformas de control de versiones como GitHub para el seguimiento y documentación del avance del proyecto.

Este informe detalla cada una de las etapas del proyecto, desde la planeación inicial hasta la implementación final, incluyendo los retos enfrentados, las soluciones adoptadas y las competencias profesionales fortalecidas. Además, se presentan recomendaciones para futuras mejoras y posibles aplicaciones del sistema en otros contextos industriales o institucionales.

CAPÍTULO I.

CARACTERIZACIÓN DEL ÁREA EN QUE PARTICIPÓ

1.1 Datos Generales

1.1.1. Instituto Tecnológico de Chihuahua.

1.1.2. Av. Tecnológico 2909, Tecnológico, 31200 Chihuahua, Chih.

Teléfono: 614-201-2000



1.1.3. Educacional (manufactura, comercial o de servicio)

1.1.4. Pequeña empresa.

1.1.5. Educación.

1.2 Breve reseña histórica de la empresa

El Instituto Tecnológico de Chihuahua nace el 26 de septiembre de 1948, cuando el Lic. Manuel Gual Vidal, secretario de Educación Pública y el Ing. Fernando Foglio Miramontes, Gobernador Constitucional del Estado, pusieron la primera piedra de lo que serían las instalaciones de este Instituto.

La construcción de los edificios se inició el 13 de noviembre del mismo año, siendo el ingeniero Alfredo Guevara Cepeda, el contratista y el ingeniero Jesús Roberto Duran el ingeniero residente.

El 9 de octubre de 1948, la secretaria de educación pública oficializa la designación del Ing. Gustavo Alvarado Pier como Directore del Tecnológico de Chihuahua.

El 20 de septiembre de 1949 inicia el Tecnológico sus actividades docentes en el 3er. Piso del palacio de gobierno (ya que sus edificios se encontraban en construcción) con 1° y 2° años de vocacional y un curso propedéutico para los jóvenes que habían llevado la preparatoria en el Instituto Científico y Literario y querían ingresar al Tecnológico.

Se inscriben en 1° de vocacional 20 alumnos, así como en 2° y curso propedéutico 43. El primer alumno inscrito fue Carlos Ballesteros Flores.

En septiembre de 1950 se inician, en los edificios del Tecnológico, los cursos de ingeniería industrial en productos orgánicos y en productos inorgánicos, se inscriben en ingeniería industrial en productos orgánicos 24 alumnos.

La inauguración de las instalaciones del Tecnológico se lleva a cabo el 22 de octubre de 1952. Se inician los cursos de capacitación para obreros, vocacional, contra semestre (nivel sub-profesional), ingeniería en productos orgánicos e inorgánicos.

Siendo el Tecnológico el primero en ofrecer las carreras técnicas a Nivel Superior, ya que el Tecnológico de Durango tenía integrada su estructura educativa en 1957 con ciclos de Secundaria Técnica, vocacional y carreras cortas de Mecánico Tornero, Electricista Embobinador e Instalador y Mecánico Automotriz, y hasta la 2da quincena de septiembre de 1960 inicia con 24 alumnos la carrera de Ingeniería Industrial. La de Técnico la inició el tecnológico de Durango el 2 de agosto de 1958, con planes de estudio elaborados en el Tecnológico de Chihuahua en 1948. En 1953 egresa la primera generación de Técnicos en el país (es la única del país que ofrece esta carrera en ese año). En 1954 egresa la primera generación de ingenieros industriales que se convierte también en la primera del país.

1.3 Organigrama de la empresa



1.4 Misión, Visión y Políticas

- Misión.

Formar integralmente profesionales de educación superior, competitivos de la ciencia, la tecnología y otras áreas de conocimiento, comprometidos con el desarrollo económico, social, cultural, incluyente y sustentable.

- Visión.

El Tecnológico Nacional de México Campus Chihuahua es una institución de educación superior tecnológica de vanguardia, líder en innovación, desarrollo y transferencia de tecnología, con reconocimiento internacional por el destacado desempeño de sus egresados y por su capacidad innovadora y sustentable en la generación y aplicación de conocimientos que impactan significativamente los sectores productivos, público y privado.

- Valores.
- Lealtad.
- Responsabilidad.
- Honestidad.
- Respeto.
- Espíritu de servicio.

1.5 Productos y clientes

Servicios Tecnológicos:

El ITCh también ofrece servicios tecnológicos a empresas y organizaciones, como capacitación en temas de calidad, contabilidad, finanzas, procesamiento de imágenes, radiofrecuencia, normas ISO y control estadístico de procesos.

Clientes:

Los estudiantes son el principal grupo objetivo de los servicios educativos del ITCh. Además, empresas y organizaciones que buscan personal especializado y servicios de consultoría o capacitación, son clientes potenciales de los servicios tecnológicos del instituto.

1.7 Premios y certificaciones

Certificaciones:

- **Modelo de Equidad de Género:**

El ITCH ha implementado un modelo que promueve la igualdad de género en sus procesos, garantizando que todos los estudiantes y personal tengan las mismas oportunidades, [según Mextudia](#).

- **Sistema de Gestión Ambiental:**

El instituto ha implementado un sistema que busca minimizar el impacto ambiental de sus operaciones, promoviendo prácticas sostenibles, [según Mextudia](#).

- **Sistema de Gestión de la Calidad:**

El ITCH cuenta con un sistema que asegura la calidad de sus procesos educativos y administrativos, garantizando la excelencia en la formación de sus estudiantes, [según Mextudia](#).

Reconocimientos y Premios:

- El ITCH ha sido reconocido por su trayectoria y calidad en la formación de ingenieros electrónicos, obteniendo premios y distinciones que demuestran su compromiso con la excelencia, [según Mextudia](#).
- Estudiantes de la carrera de Ingeniería Electrónica han recibido certificados de SolidWorks, reconocidos por su dominio en herramientas de diseño y manufactura, [según el Instituto Tecnológico de Chihuahua](#).
- El ITCH ha sido reconocido por su compromiso con la equidad de género, implementando un modelo que garantiza que todos los estudiantes y personal tengan las mismas oportunidades, [según Mextudia](#).

1.8 Relación de la empresa con la sociedad

Formar integralmente profesionales de educación superior, competitivos de la ciencia, la tecnología y otras áreas de conocimiento.

CAPÍTULO II.

PLANTEAMIENTO DEL PROBLEMA O ÁREA DE OPORTUNIDAD

Desarrollar un sistema automatizado para la gestión de barreras vehiculares en estacionamientos con accesos controlados, utilizando tecnologías distribuidas en red. El sistema deberá integrar dispositivos embebidos como ESP32 y Raspberry Pi, junto con herramientas de visualización y control como Node-RED y bases de datos para garantizar un funcionamiento confiable, escalable y en tiempo real.

Brindar solución a la gestión de barreras vehiculares en estacionamientos con accesos controlados de manera automatizada, utilizando tecnologías distribuidas en red.

2.1. Caracterización del área en que realizó el proyecto.

2.1.1. Descripción del área

Formar profesionistas con competencias profesionales para diseñar, modelar, implementar, operar, integrar, mantener, instalar, administrar, innovar y transferir tecnología electrónica existente y emergente en proyectos interdisciplinarios a nivel nacional e internacional para resolver problemas y atender las necesidades de su entorno con ética, actitud emprendedora, creativa y comprometidas con el desarrollo sustentable.

2.1.2. Organigrama del área



2.1.3. Actividades del área

Diseñar , analizar y cosntruir equipos y/o sistemas electrónicos para la solucionar de problemas la solución de problemas en el entorno profesional, aplicando normas técnicas y estándares nacionales e internacionales.

2.1.4. Interrelación con otras áreas de la empresa

- **Ingeniería Electrónica:**

El laboratorio es fundamental para la enseñanza práctica de la electrónica, permitiendo a los estudiantes desarrollar habilidades en el diseño, construcción, prueba y mantenimiento de circuitos y sistemas electrónicos.

- **Ingeniería Eléctrica:**

El laboratorio facilita la enseñanza de principios eléctricos y la aplicación de la electrónica en sistemas eléctricos.

- **Ingeniería Mecatrónica:**

El laboratorio es esencial para la enseñanza de la integración de la electrónica en sistemas mecánicos.

- **Ingeniería Industrial:**

El laboratorio apoya la formación de ingenieros industriales en el manejo de equipos electrónicos y la automatización de procesos.

- **Ingeniería en Computación:**

El laboratorio facilita la enseñanza de principios de la electrónica y la interconexión de dispositivos electrónicos en sistemas computacionales.

2.1.5. Funciones y ubicación del residente

- Diseño del sistema de barreras vehiculares.
- Programación en los siguientes lenguajes Arduino, C#, JavaScript, python.
- Desarrollo de interfaces gráficas.
- Control de versiones.

2.2. Antecedentes y definición del problema para la realización del proyecto de Residencia o bien, precisar el área de oportunidad de interés para la Organización.

El uso y la implementación de un control de barreras vehiculares es un control necesario que toda empresa o escuela debería de tener debido a la seguridad y el control que brinda el uso de las mismas debido a la automatización de la información y el control de accesos y salidas de la misma, en este caso el instituto tecnológico de Chihuahua, en el cual se pueden controlar de manera remota y manual gracias a la supervisión de un área especializada encargada de esta tarea, se pueden obtener datos precisos de la asistencia de personal y estudiantes, así como desarrollar estrategias futuras para otras áreas, teniendo conocimiento de cuantos vehículos se encuentran en la institución en cada momento, tener el conocimiento de cuantos docentes que cuenten con un vehículo registrado se

encuentran en cada momento y en qué área estos se encuentran, así como la asistencia de los alumnos y su hora de ingreso a la institución periódicamente, así como el control de retrasos en el acceso a las aulas y una clave única con distintos niveles de acceso conocidas como Ids. Esto se puede extrapolar a ofertas de horarios flexibles para estudiantes, poniendo como ejemplo al centro de lenguas extranjeras y maternas, el cual tiene problemas con asignar horarios adecuados para los alumnos de la institución, con estos datos parte del problema se solventaría ya que se podrían ofertar mejores horarios para los y las alumnas de la institución, sabiendo su disponibilidad, cuanto tiempo en promedio un alumno pasa dentro de la institución, y cuando este se retira de la misma, adaptando estos datos al modelo de clases de inglés.

El control de esta información puede beneficiar a la institución en gran medida brindando una mejor experiencia a los empleados y a los alumnos del instituto tecnológico de Chihuahua

2.3. Objetivos.

2.3.1. Objetivo general.

1. Desarrollar un sistema automatizado para la gestión de barreras vehiculares en estacionamientos con accesos controlados.

2.3.2. Objetivos específicos.

1. Utilizar tecnologías distribuidas en red.
2. Integrar dispositivos embebidos como ESP32 y Raspberry Pi.
3. Utilizar herramientas de visualización y control como Node-RED y bases de datos

2.4. Justificación

Se tiene como finalidad el control de barreras vehiculares para el uso de agentes internos y externos en el instituto tecnológico de Chihuahua debido al uso constante de entradas y salidas en las diferentes áreas brindando seguridad y control de cada uno de los accesos.

2.5. Alcances y Limitaciones.

2.5.1. Alcances.

Control de acceso de las áreas vehiculares al rededor del instituto tecnologico de Chihuahua, control general y especifico.

2.5.2. Limitaciones.

Limitado a areas vehiculares, no a otros accesos que esten fuera de estos.

CAPÍTULO III.

FUNDAMENTO TEÓRICO

3.1 Protocolo MQTT.

El protocolo MQTT (Message Queuing Telemetry Transport) es uno de los más utilizados en el ámbito del Internet de las Cosas (IoT). Se trata de un protocolo de mensajería ligero, basado en el modelo de publicación/suscripción (Pub/Sub), que facilita el intercambio de datos entre dispositivos conectados, como sensores, actuadores, microcontroladores y controladores lógicos programables (PLC).

En MQTT, los dispositivos emisores (publicadores) y receptores (suscriptores) se comunican mediante temas o topics, pero están desacoplados entre sí. La gestión de esta comunicación es responsabilidad del bróker MQTT, el cual recibe todos los mensajes publicados y los distribuye a los suscriptores correspondientes en función del tema al que estén suscritos.

Una de las ventajas clave de MQTT es su bajo consumo de ancho de banda y recursos, lo que lo hace ideal para dispositivos embebidos y redes con limitaciones. Existen múltiples implementaciones del protocolo, tanto comerciales como de código abierto. Por ejemplo, HiveMQ ofrece un cliente MQTT basado en Java y un bróker MQTT disponible en versiones comerciales y de código abierto.



Figura 3.1.1

3.2 Plataforma de Control y Visualización (Node-RED).

Node-RED es una herramienta de desarrollo basada en flujo que permite la creación de aplicaciones mediante una interfaz gráfica. Está diseñada para facilitar la recolección, transformación y visualización de datos, sin requerir conocimientos avanzados de programación.

Gracias a su enfoque de bajo código (low-code), Node-RED es accesible para usuarios de distintos perfiles y es ampliamente utilizado en ámbitos como la domótica, el control industrial y sistemas IoT. Sus flujos están basados en nodos que representan operaciones, entradas, salidas o conexiones entre dispositivos y servicios.

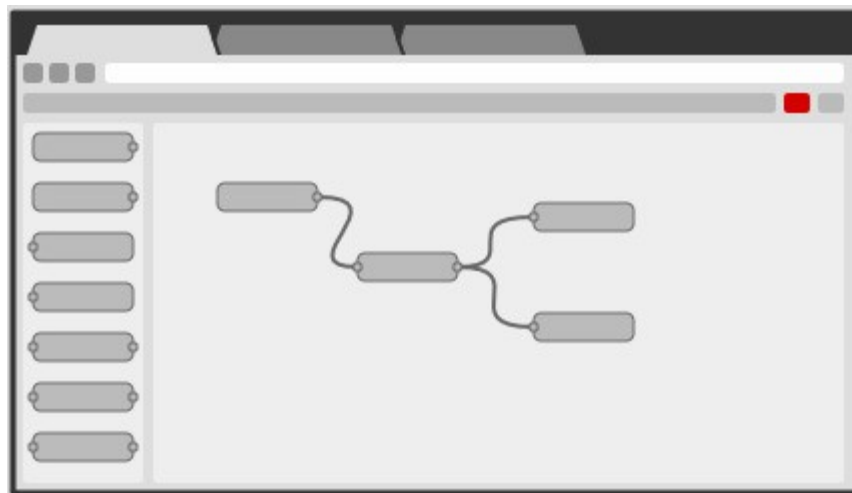


Figura 3.2.1

3.3 RabbitMQ.

RabbitMQ es un sistema de mensajería de código abierto que actúa como message broker, permitiendo la comunicación entre aplicaciones o componentes distribuidos. Es confiable, maduro y ampliamente utilizado tanto en entornos locales como en la nube.

RabbitMQ implementa el protocolo AMQP (Advanced Message Queuing Protocol) y es conocido por su robustez, escalabilidad y facilidad de implementación en

arquitecturas modernas, incluyendo microservicios y sistemas basados en eventos.

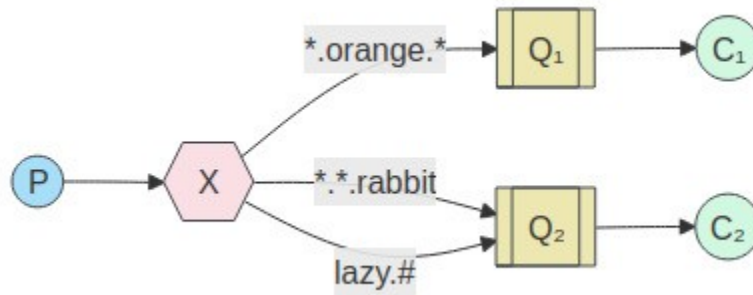


Figura 3.3.1

3.4 Docker compose.

Docker Compose es una herramienta que permite definir y ejecutar aplicaciones multicontenedor. Mediante un archivo de configuración en formato YAML (`docker-compose.yml`), es posible describir los servicios, redes y volúmenes que conforman una aplicación.

Con un solo comando, Docker Compose puede construir, iniciar y gestionar todos los servicios especificados en el archivo, facilitando el desarrollo, despliegue y mantenimiento de entornos complejos. Funciona en entornos de desarrollo, pruebas, producción y en flujos de trabajo de integración continua (CI/CD).

Las funcionalidades principales incluyen:

- Iniciar, detener y reconstruir servicios.
- Ver el estado de los contenedores en ejecución.
- Consultar registros de los servicios.
- Ejecutar comandos dentro de contenedores.

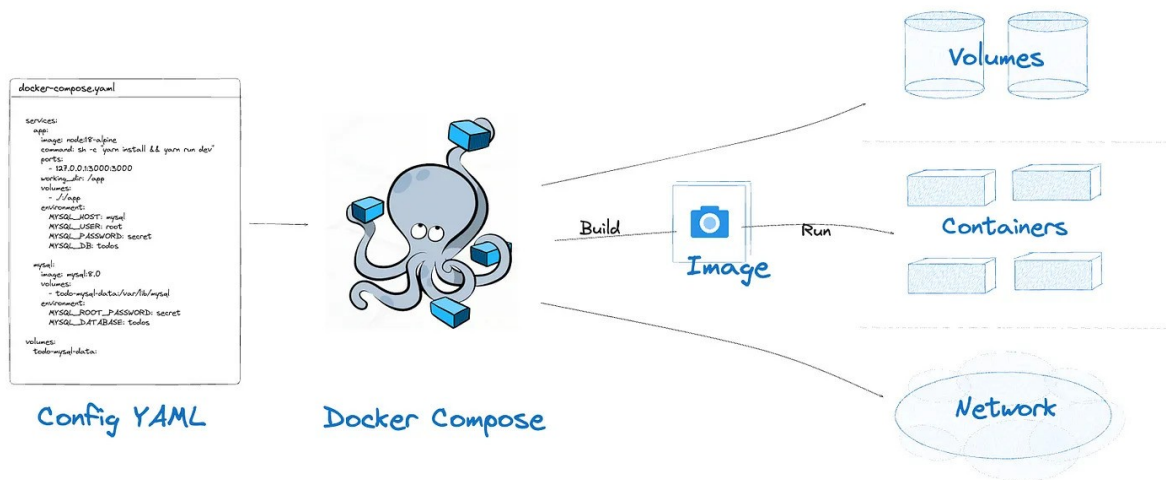


Figura 3.4.1

3.5 Portainer.io

Portainer es una plataforma gráfica de administración para entornos basados en contenedores. Ofrece una interfaz intuitiva para gestionar instancias de Docker y Kubernetes, ya sea en entornos locales, centros de datos, la nube o dispositivos edge.

Permite a los usuarios desplegar, supervisar y administrar contenedores, redes y volúmenes sin necesidad de recurrir a la línea de comandos, lo que facilita la adopción de tecnologías de contenedores en equipos con distintos niveles de experiencia.



Figura 3.5.1

3.6 Visual studio code.

Visual Studio Code (VS Code) es un editor de código fuente gratuito, ligero y multiplataforma desarrollado por Microsoft. Es compatible con numerosos lenguajes de programación y cuenta con funcionalidades como:

- Resaltado de sintaxis.
- Autocompletado de código (IntelliSense).
- Depuración integrada.
- Integración con sistemas de control de versiones como Git.

Su extensibilidad mediante plugins y su enfoque en el desarrollo moderno lo convierten en una de las herramientas más populares entre desarrolladores.

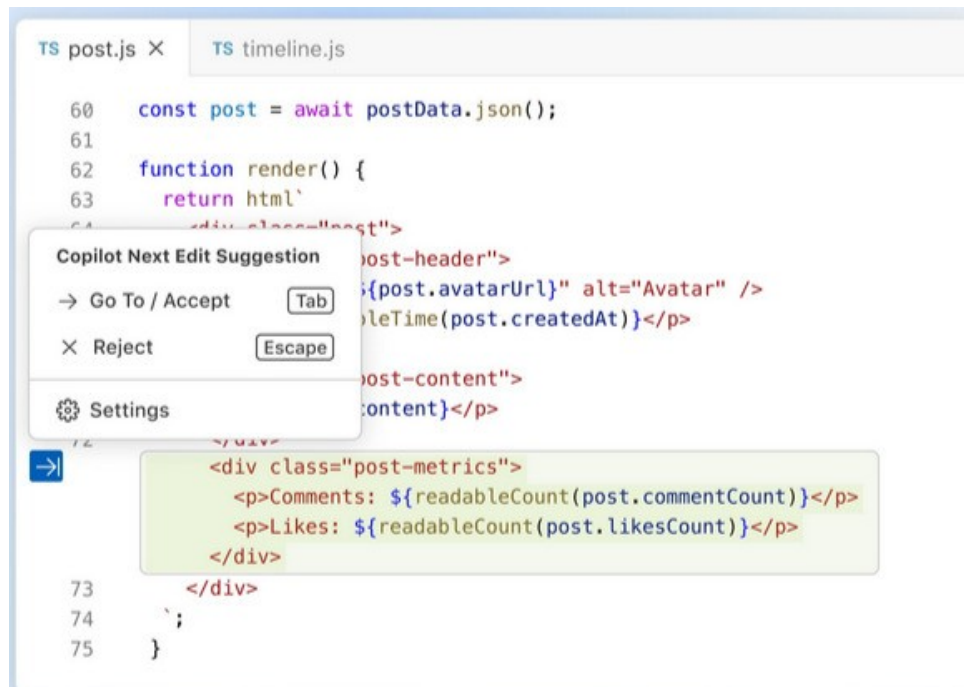
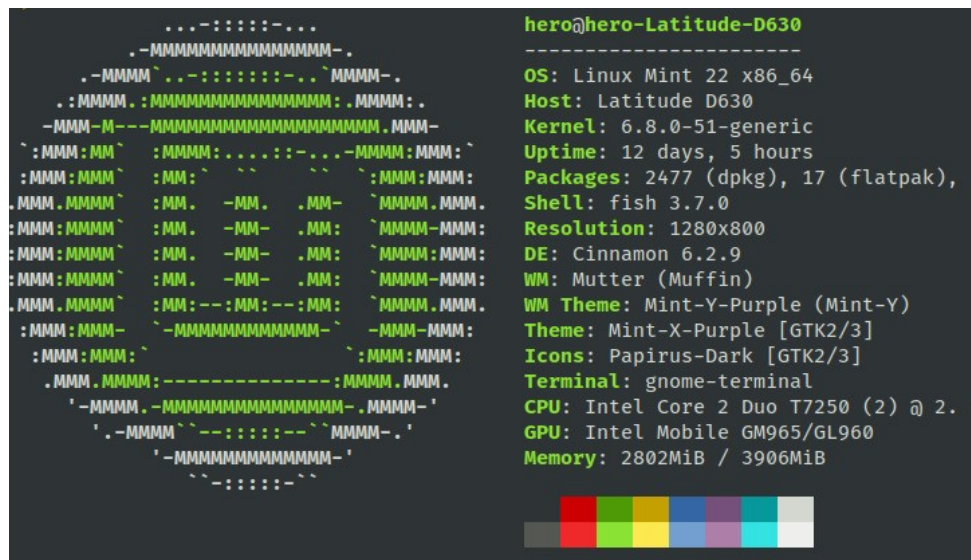


Figura 3.6.1

3.7 Linux.

Linux es un sistema operativo de código abierto que actúa como intermediario entre el hardware y las aplicaciones. Su arquitectura se divide en tres componentes principales:

- Kernel: es el núcleo del sistema operativo. Se encarga de gestionar recursos como la memoria, los procesos, los dispositivos y el sistema de archivos.
- Espacio de usuario: incluye herramientas administrativas, el shell (línea de comandos), daemons (procesos en segundo plano) y entornos de escritorio.
- Aplicaciones: son programas que permiten realizar tareas específicas. Linux ofrece repositorios desde donde se pueden instalar aplicaciones para tareas de desarrollo, administración, ofimática, entre otras.



```

      ...-:::-...
    .-MMMMMMMMMMMMMM-.
    .-MMM~..-:::-..`MMM-.
    .:MMM.:MMMMMMMMMMMMMM.:MMM:.
    -MM-M--MMMMMMMMMMMMMMMMMM.MMM-
    ^:MMM:MM~:MMM:~:::-..-MMM:MM:~
    :MMM:MM~:MM:~`~`~`~:MMM:MM:
    .MMM.MMM~:MM.-MM..MM-`MMM.MMM.
    :MMM:MM~:MM.-MM-..MM:~MMM-MMM:
    :MMM:MM~:MM.-MM-..MM:~MMM-MMM:
    :MMM:MM~:MM.-MM-..MM:~MMM-MMM:
    :MMM:MM~:MM:--MM:--MM:~MMM.MMM.
    :MMM:MM~`-MMMMMMMMMMMMMM-`-MMM-MMM:
    :MMM:MM~`~:MMM:MM:
    .MMM.MMM:-----:MMM.MMM.
    '-MMM.-MMMMMMMMMMMMMM-.MMM-'
    '-MMM~--:::-~`MMM-.'
    '-MMMMMMMMMMMMMM-'
    ~-:::-~

```

```

hero@hero-Latitude-D630
-----
OS: Linux Mint 22 x86_64
Host: Latitude D630
Kernel: 6.8.0-51-generic
Uptime: 12 days, 5 hours
Packages: 2477 (dpkg), 17 (flatpak),
Shell: fish 3.7.0
Resolution: 1280x800
DE: Cinnamon 6.2.9
WM: Mutter (Muffin)
WM Theme: Mint-Y-Purple (Mint-Y)
Theme: Mint-X-Purple [GTK2/3]
Icons: Papyrus-Dark [GTK2/3]
Terminal: gnome-terminal
CPU: Intel Core 2 Duo T7250 (2) @ 2.
GPU: Intel Mobile GM965/GL960
Memory: 2802MiB / 3906MiB

```




Figura 3.7.1

3.8 ESP32.

El ESP32 es un microcontrolador de bajo costo y consumo energético desarrollado por Espressif Systems. Integra conectividad Wi-Fi y Bluetooth, lo que lo hace ideal para aplicaciones IoT.

Está disponible en múltiples versiones y módulos de desarrollo, con diversas configuraciones de pines y funcionalidades. Su versatilidad y bajo precio lo han convertido en una opción popular para proyectos de automatización, monitoreo, control y prototipado.



Figura 3.8.1

3.9 WifiManager.

Al iniciar su ESP, se configura en modo Estación e intenta conectarse a un punto de acceso previamente guardado. Si no tiene éxito (o no hay ninguna red guardada), el ESP cambia al modo Punto de Acceso y activa un servidor DNS y un servidor web (IP predeterminada: 192.168.4.1).

Con cualquier dispositivo con wifi y navegador (ordenador, teléfono o tableta), conéctese al punto de acceso recién creado.

Debido al Portal Cautivo y al servidor DNS, aparecerá una ventana emergente de tipo "Unirse a la red" o cualquier dominio al que intente acceder será redirigido al portal de configuración.

Seleccione uno de los puntos de acceso escaneados, introduzca la contraseña y haga clic en Guardar.

El ESP intentará conectarse. Si tiene éxito, cede el control a su aplicación. De lo contrario, vuelva a conectarse al punto de acceso y reconfigúrelo.

Hay opciones para cambiar este comportamiento o para iniciar manualmente el portal de configuración y el portal web de forma independiente, así como para ejecutarlos en modo sin bloqueo.

AutoConnectAP

WiFiManager

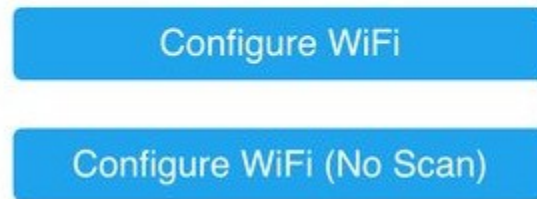


Figura 3.9.1

CAPÍTULO IV.

DESARROLLO DEL PROYECTO

4.1 Conexion MQTT inicial.

En el ámbito inicial se tiene que observar el problema desde una perspectiva de un ecosistema completo, debido al control de acceso para barreras vehiculares en el ITCh lleva un análisis inicial, esto debido a que se tienen que tomar ciertas consideraciones al momento de realizar el sistema, como el control humano sistema, siendo este dividido en control manual y control remoto, a su vez la configuración de cada uno de estos dispositivos, mediante una interfaz de usuario y botones físicos, así como la conexión con el sistema o servidor principal en el cual se obtendrá o se registrara la información de identificadores o lds.

Al comenzar se tuvo que tener en claro lo que es el protocolo MQTT el cual está indicado en el (capitulo III fundamentos teóricos) el cual utilizaría como interprete una ESP32 y un broker publico llamado HIVEMQ para pruebas iniciales. Teniendo los siguientes materiales:

- 1 Esp32.
- 1 Led.
- 2 Cables.
- 1 Resistencia de 220 Ω .
- Acceso a una cuenta de HIVEMQ.

Como prueba inicial se creo un topico llamado control-led para la comunicación entre la ESP32 y el broker de HIVEMQ. Obteniendo el siguiente resultado:

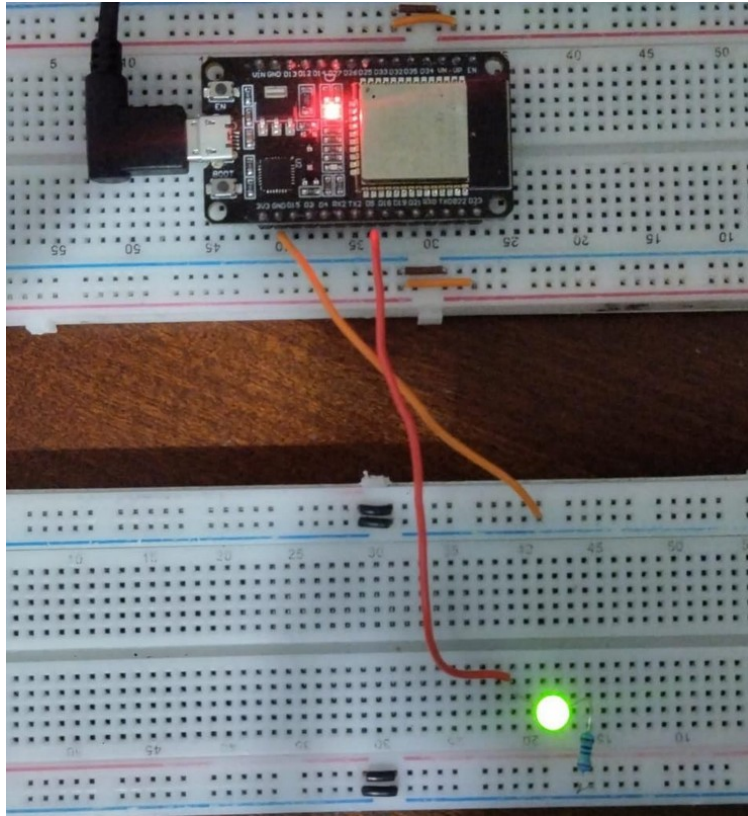


Figura 4.1.1

De manera simple tenemos una conexión entre la ESP32 y el broker en el cual mediante un tópico llamado control-led se enciende un led o se apaga, todo esto lo realizamos mediante la interfaz de usuario del broker antes mencionado, tal cual como se muestra en la siguiente figura:

Send Message 0

If you cannot see any messages, make sure you are subscribed to the correct topics. You can always subscribe to the (#) wildcard to receive all messages.

Message *

Topic * control-led **QoS *** QoS: 0

Send Message

Figura 4.1.2

Tal como se muestra en la figura 4.1.2 tenemos el apartado de (Message) en el cual podemos enviar distintos tipos de mensajes, estos pueden llegar a ser mensajes de tipo carácter, booleanos, enteros, dobles, o una combinación de más de un tipo antes mencionado como una simple cadena de caracteres, cabe mencionar que el topic o tópico es muy importante ya que este tiene que coincidir con el que configuramos con anterioridad en la ESP32 si no el mensaje se enviara pero no llegara al destino indicado.

El uso y manejo de los tópicos es muy importante en el uso del protocolo MQTT ya que este es la base de todo el ecosistema, un buen manejo de los tópicos asegura una buena implementación del sistema.

4.2 Conexion basica con botones fisicos e interfaz de usuario en node-red.

Una vez realizada una conexión exitosa con un broker mediante comunicación MQTT y el control de un elemento físico, en este caso un led, se pasó al siguiente paso el cual es un control más complejo, agregando un segundo led y botones, conformado por el siguiente material:

- 1 ESP32.
- 2 Leds.
- Cables.
- 2 Resistencias de 220Ω.
- 2 Resistencias de pull-up.
- 2 Botones.
- Acceso a una cuenta de HIVEHQ.
- Node-red.

En el consiguiente proceso se implementó un sistema más complejo, el cual dependía de dos leds y dos botones, una comunicación entre el broker y la ESP32 y a su vez una interfaz gráfica de usuario propia la cual se conectará automáticamente con el broker y poder mandar los mensajes en el protocolo MQTT sin que el usuario tenga que insertar sus propias credenciales al broker y

hacer una comunicación con la ESP32 y configurar de manera manual los tópicos. Todo esto fue realizado en node-red, obteniendo el siguiente resultado físico.

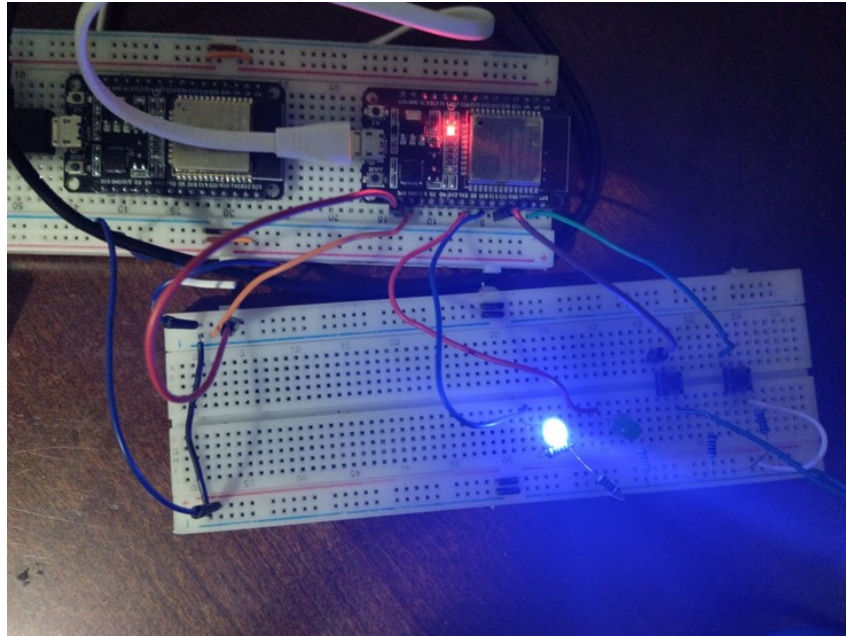


Figura 4.2.1

Como se puede observar en la figura 4.2.1 tenemos la implementación de los materiales antes mencionados, teniendo dos leds, uno el cual representaría el apagado o cerrado de la pluma de acceso vehicular siendo este de color azul y otro que representaría el encendido o levantamiento de la pluma vehicular representado por el color verde. A su vez tenemos dos botones, uno para el encendido y el apagado manual los cuales el usuario puede controlar sin afectar el sistema, para la interfaz de usuario se realizó una programación mediante nodos el cual haría la función de intermediario entre la ESP32 y el broker de HIVEMQ, tal cual como se muestra en las siguientes figuras:

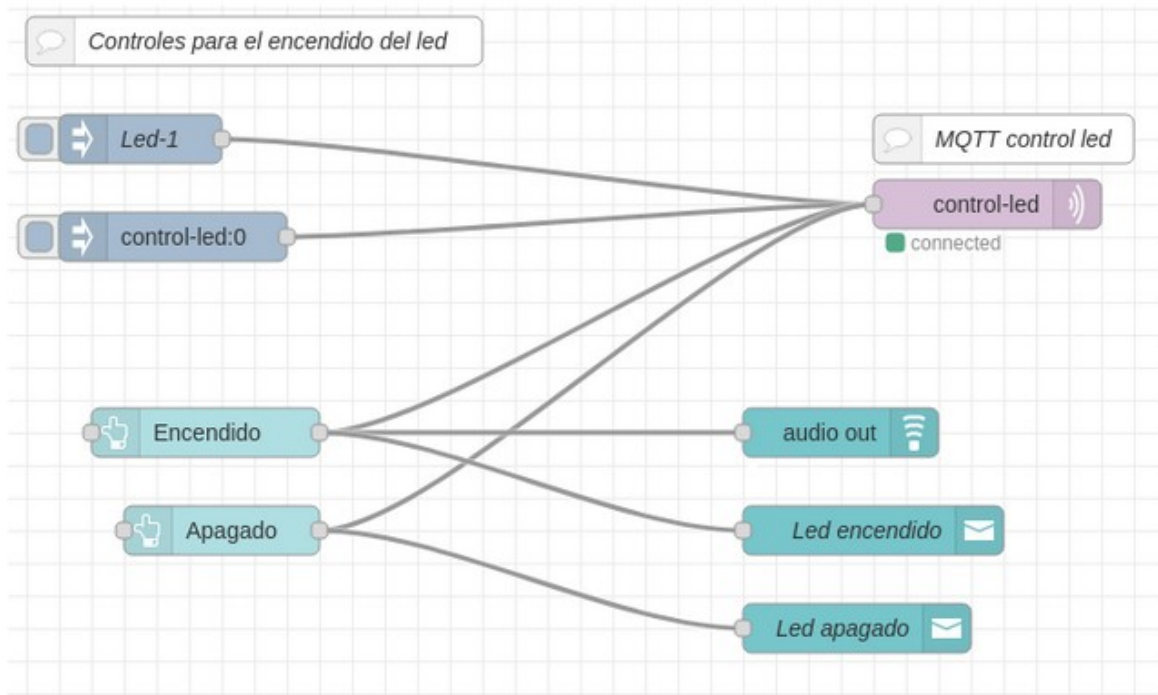


Figura 4.2.2



Figura 4.2.3

El uso de nodos muestra una programación simple y cansilla de entender a simple vista en el cual tenemos dos botones representados en la figura 4.2.2 los cuales están conectados a un nodo de salida MQTT, en el cual se configuro el broker, sin embargo, lo que va a ver el usuario final es la figura 4.2.3 en donde simplemente tenemos dos botones virtuales de encendido y apagado los cuales controlarían los leds de encendido y apagado mencionados anteriormente.

Como agregado se utilizó un simulador llamado wokwi de ESP32 para la corroboración del funcionamiento de más de una ESP32 al mismo tiempo, tal como se muestra en la siguiente figura:

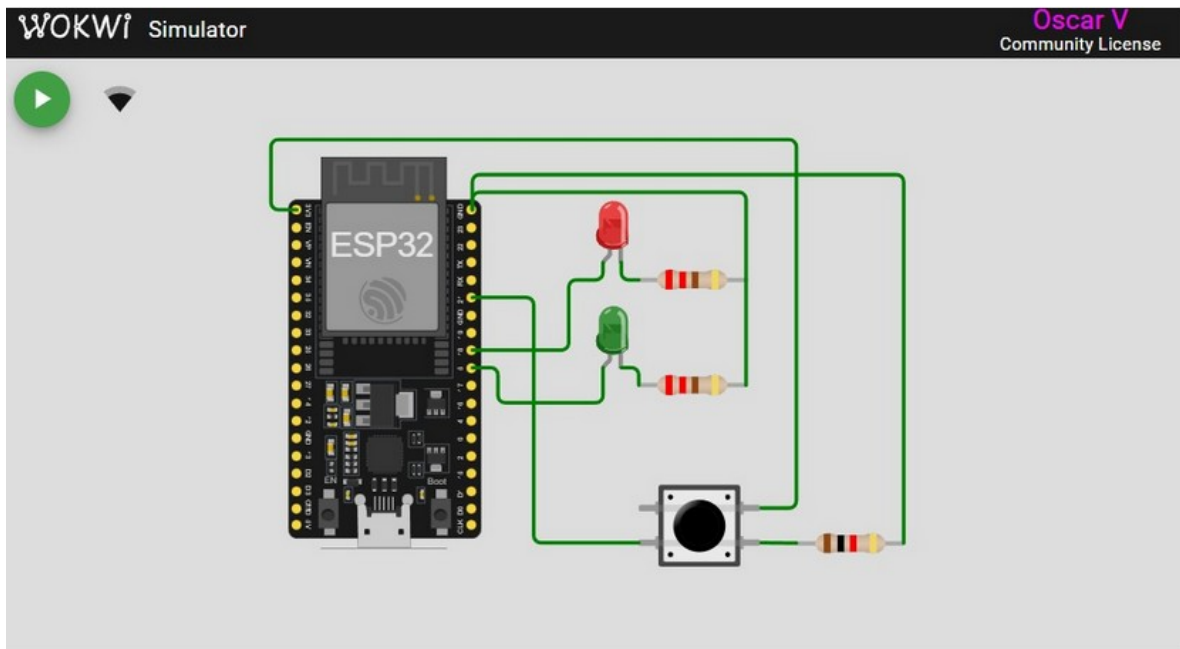


Figura 4.2.4

4.3 Implementacion de wifimanager al proyecto.

Una vez realizadas las funciones básicas y tal como se menciona en el capítulo 3, se presenta el uso de la librería wifimanager al proyecto el cual tenemos que implementar en la ESP32, se tuvo que estudiar detalladamente cada uno de los aspectos que conllevan esta librería, así como sus distintos modos de configuración los cuales son diversos, centrándonos en los principales como la conexión a una red wifi local con la cual la ESP32 podrá tener acceso a la red local del instituto tecnológico de chihuahua, siendo esta uno de los modos de conexión que se implementaron, al configurar estos tenemos una interfaz agradable para el usuario, en la cual la ESP32 realiza la emisión de una señal wifi en la cual se pueden conectar cualquier dispositivo que tenga acceso a wifi, tales como un celular, una Tablet, una computadora, entre otras, en la cual aparecerá el nombre de una red la cual se eligió con anterioridad, para el efecto de las distintas pruebas, tal como se muestra en la siguiente figura.

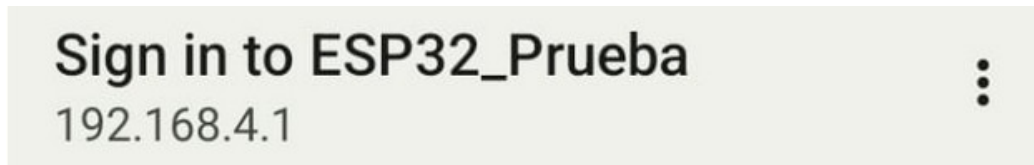


Figura 4.3.1

Esta señal habilita un menú en la cual podemos configurar nuestra conexión wifi a la ESP32 sin necesidad de tocar una línea de código, la cual lo hace reconfigurable y agradable al usuario ya que esta se puede habilitar manualmente tal y como lo vamos a ver posteriormente. Al tener esta interfaz de usuario tendremos las siguientes opciones.

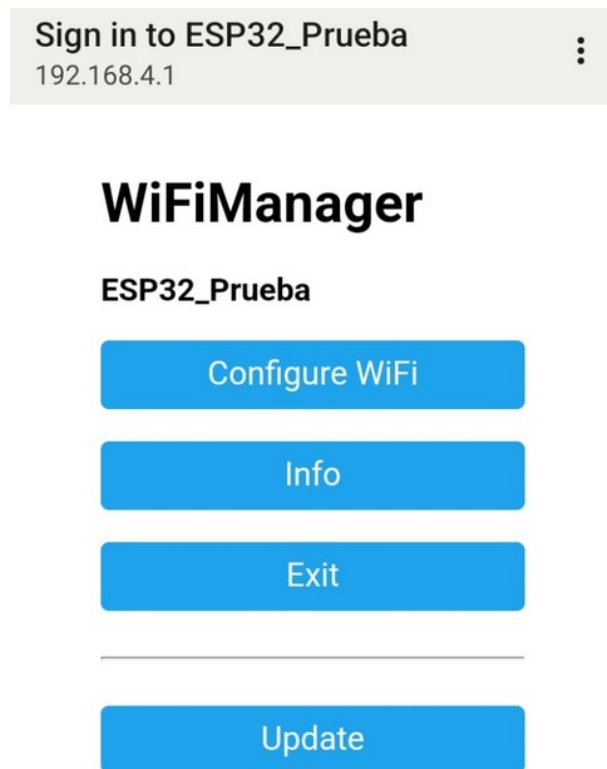


Figura 4.3.2

Tal como se muestra en la figura 4.3.2 tenemos distintas opciones, teniendo como título la herramienta o librería utilizada, siendo esta llamada ESP32_Prueba debido

al tiempo de desarrollo en el cual se estaba realizando la conexión exitosa, como segunda opción se tiene la opción principal llamada configure WiFi la cual nos permite realizar conexiones exitosas a la red wifi a la que tengamos permisos para acceder, la conexión permite redes 5G aunque estas no son recomendables debido a su alcance limitado, sin embargo depende del uso que se le esté dando, este menú despliega todas las redes disponibles y nos permite la conexión wifi, como segundo botón de configuración tenemos información, el cual nos despliega información acerca de este driver, el cual puede ser editado si así se desea, por ultimo tenemos el botón de update, el cual nos permite actualizar si es necesario, cabe mencionar que todo este menú es personalizable y depende de las necesidades del proyecto, también qué tipo de usuario tendrá acceso a este menú, ya que al ser un menú de configuración debe de ser manejado por personal autorizado con credenciales correspondientes de acceso.

4.4 Implementación de docker compose.

Creacion de imagenes.

Al estar trabajando con linux, específicamente con las distribuciones de linux mint y debian 12 el uso y la instalacion de docker compose es diferente a una instalacion comun de windows la cual principalmente se apoya del uso de la terminal, en mi caso en particular estoy utilizando fish, pero los comandos estan para una terminal bash, así que se recomienda trabajar sobre esta en particular, como sistema operativo principal estoy utilizando linux mint así como lo mencione anteriormente, este es un deribado de debian por lo tanto al estar docker compose soportado para debian funciona en esta distribucion sin problemas de compatibilidad tal como se muestra en la siguiente figura:

Supported platforms

Docker provides `.deb` and `.rpm` packages for the following Linux distributions and architectures:

Platform	x86_64 / amd64
Ubuntu	✓
Debian	✓
Red Hat Enterprise Linux (RHEL)	✓
Fedora	✓

Figura 4.4.1

Una vez instalado docker compose, se utilizaron tres comandos principales para su uso y manejo, siendo el primero:

- `sudo docker images` (el cual nos permite tener una visualización general de las imágenes instaladas en el sistema)
- `sudo docker compose up` (el cual nos permite levantar todo un ambiente a partir de la configuración de un archivo `.yml`)
- `sudo docker compse down` (el cual nos permite terminar un ambiente previamente activo)

```
⚡hero >> sudo docker images
[sudo] password for hero:
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
portainer/portainer-ce  latest             d3e9f39af9c5       3 weeks ago        268MB
nodered/node-red      latest             89ca28c71740       2 months ago        601MB
hello-world           latest             74cc54e27dc4       4 months ago        10.1kB
rabbitmq              3.8-management-alpine  69b0f3f87ca9       2 years ago        166MB
```

Figura 4.4.2

Portainer.

Al estar utilizando la herramienta de docker compose podemos crear containers o contenedores los cuales nos permitirán guardar y utilizar estas imágenes en una ruta específica del sistema, utilizando los comandos antes mencionados. Para esto me apoye de la terminal la cual me permitió navegar por el sistema sin ningún problema, también el uso del editor de texto ya implementado por defecto en las distribuciones de linux llamado nano el cual me permite editar, crear y guardar archivos en la extensión que yo dese.

Para la configuración del contenedor inicial se utilizaron los siguientes comandos:

- `cd <la ruta a la cual se quiere navegar>`
- `sudo mkdir <nombre de la carpeta que se desea crear>`
- `ls` (el cual nos permite ver una vista general de nuestros archivos)
- `nano` (el cual nos permitirá crear nuestro archivo .yaml)

La ruta principal será la raíz la cual en el sistema linux está representada por un slash (/) en el cual se utilizó el comando `sudo mkdir my_docker`, para crear una carpeta en la cual se van a alojar las imágenes que se utilizaron durante este proyecto tal y como se muestra en las siguientes figuras.



```
⚡hero >> sudo mkdir my_rabbitmq
```

Figura 4.4.4

Una vez realizado el comando de la figura 4.4.4 pasamos a utilizar `cd /my_docker` esta carpeta fue la principal en la que se trabajó principalmente para levantar el ambiente de portainer.io, tal como se muestra una descripción general en el capítulo III de este reporte, se volvió a utilizar el comando `sudo mkdir` pero con otro nombre, tal cual como se muestra en la siguiente figura:



```
⚡hero >> sudo mkdir my_portainer
```

Figura 4.4.5

El cual va a crear una subcarpeta para alojar la imagen de portainer.io, en el cual se utilizó `cd /my_portainer`, lo cual me ubico en la subcarpeta, para después utilizar el comando `nano` y crear el archivo `.ym`, tal como se muestra en la siguiente figura:

```
File: docker-compose.yml
1 | version: "3.8"
2 | services:
3 |   portainer:
4 |     image: portainer/portainer-ce:latest
5 |     container_name: my_portainer
6 |     restart: always
7 |     ports:
8 |       - "8000:8000"
9 |       - "9000:9000"
10 |    volumes:
11 |      - /var/run/docker.sock:/var/run/docker.sock
12 |      #- ./my_data/:data
13 |
14 | volumes:
15 |   my_data:
```

Figura 4.4.6

Tal como se muestra en la figura 4.4.6 la sintaxis es muy importante al crear un archivo `.yml` para que este sea interpretado correctamente por `docker compose`, como puntos principales se tiene la versión en la que se va a trabajar, en este caso la 3.8 puede ser inferior o superior, los servicios, en este caso `portainer`, seguido la imagen que se va a descargar, luego el nombre del contenedor en este caso `my_portainer`, tal como creamos utilizando la terminal, en este caso `restart` o reinicio lo configure en `always` ya que quiero que este se ejecute siempre que el equipo inicie, ya que este al ser un gestor de imágenes es conveniente que siempre este activo, el manejo de los puertos también es algo muy importante ya que se tienen que tener una gestión adecuada de estos ya que pueden existir conflictos por el uso de estos, se crea un volumen en la ruta indicada en el archivo `.yml` y por último se crea un subvolumen el cual se llama `my_data` el cual sirve para guardar ciertos datos de `portainer`.

Cuando se creó el archivo .yaml exitosamente, se utilizó el comando `sudo docker compose up` para levantar todo el ambiente de trabajo, una vez realizado este último comando se pudo utilizar el ambiente el cual pide el registro de credenciales para seguridad. Una vez configuradas las credenciales de acceso se tiene un ambiente tal como se muestra en la siguiente figura:

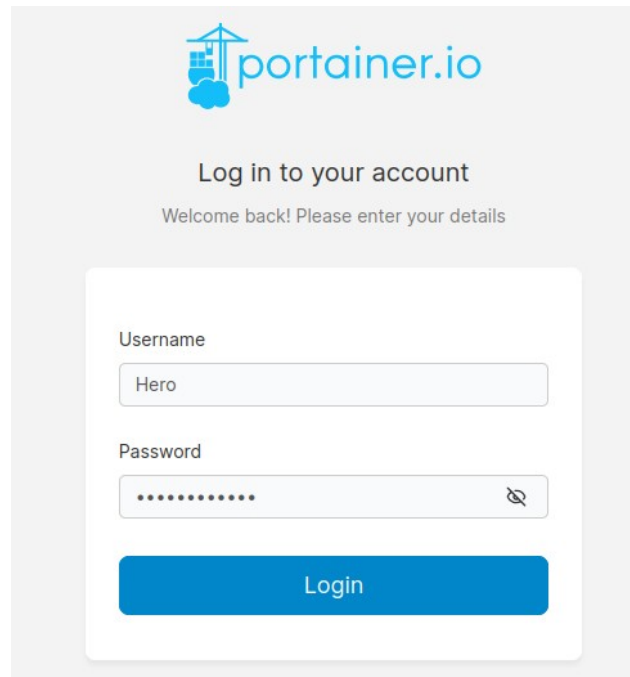


Figura 4.4.7

Una vez utilizadas las credenciales de acceso se puede ver el ambiente desplegado a su totalidad.

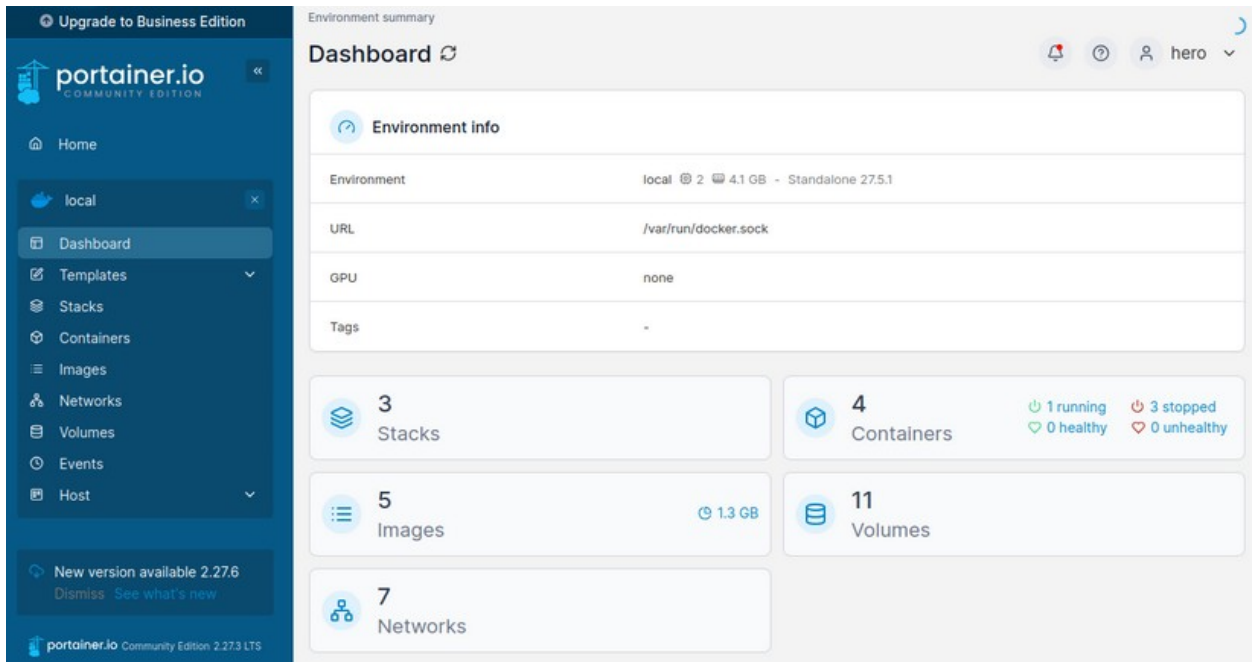


Figura 4.4.8

En el cual me centré en el apartado de dashboard el cual me da una vista general de mi sistema físico como el procesador y ram así como en mi caso específico en el cual estoy trabajando con intel, stacks, containers, volumes y networks, así como cuales de estos contenedores se están ejecutando en cada momento, su estado, representado por saludable o no saludable en caso de una corrupción del sistema.

Node-red.

Anteriormente se mencionó el uso de node-red de manera local en el equipo, ahora lo que se realizó fue el uso de esa misma plataforma con el añadido de docker compose en conjunto con portainer, ya que al ser una imagen totalmente independiente al sistema este puede ser portable con la ayuda de docker hub en un flujo de trabajo unido.

Su configuración es similar al apartado de portainer, con el uso de los comandos de terminal antes mencionados y la elaboración del archivo .yaml, al estar en el ambiente linux la ruta que utilice es la siguiente `/my_docker/my_nodered/` para esto se utilizaron los comandos `cd`, `sudo mkdir`, `ls` y `nano` para elaborar el

archivo .yml así como una subcarpeta llamada my_data/ ubicada en la ruta antes mencionada.

El archivo .yml se muestra en la siguiente figura:

```
7 | version: "3.7"
8 |
9 | services:
10 |   node-red:
11 |     image: nodered/node-red:latest
12 |     environment:
13 |       - TZ=Europe/Amsterdam
14 |     ports:
15 |       - "80:1880"
16 |     networks:
17 |       - node-red-net
18 |     volumes:
19 |       - my_data:/data
20 |
21 | volumes:
22 |   my_data:
23 |
24 | networks:
25 |   node-red-net:
```

Figura 4.4.9

En el apartado de services o servicios se configuro la imagen que se va a utilizar, en este caso node-red en la última versión, cabe mencionar que hay ocasiones en las cuales trabajar con la última versión no es lo más optimo, en este caso en particular si se puede trabajar con la última versión sin problemas, en este caso en ambiente es el recomendado por la página oficial de node red, así como lo mencione antes el uso de los puertos es importante porque pueden existir conflictos entre los mismos, en este caso yo tengo instalado node red de manera local y utiliza el puerto 1880 de manera predeterminada y al utilizar docker y en específico node red que utiliza este puerto no puedo usar el puerto 1880 ya que este está ocupado entonces simplemente se mapea en otro puerto libre, el 80 tal y como se muestra en la figura, cabe mencionar que este node red es totalmente independiente al node red instalado de manera local, esto quiere decir que se

puede tener dos versiones distintas del mismo ambiente trabajando de manera totalmente independiente sin conflicto alguno.

Al tener instalado portainer se puede habilitar este ambiente sin ningún problema desde su ambiente gráfico, sin depender de la terminal/consola tal como se muestra en la siguiente figura:

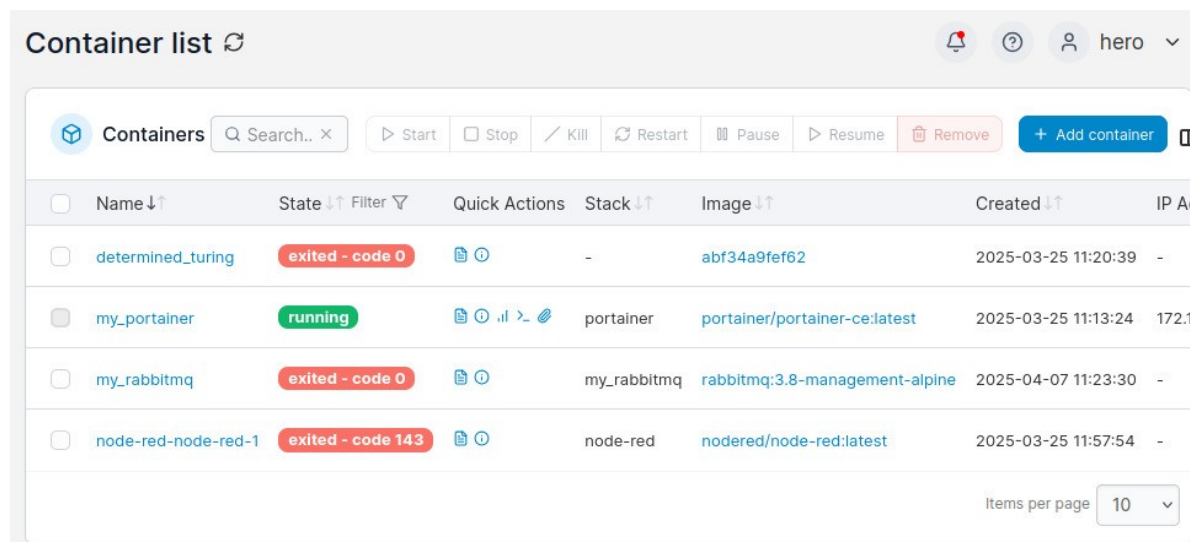


Figura 4.4.10

En este caso en el apartado de containers tenemos un submenú llamado container list, el cual muestra los contenedores instalados en el sistema, estos son los principales utilizados en el proyecto. Gracias a la interfaz gráfica de portainer tenemos opciones principales, siendo start, el cual sirve para iniciar el ambiente del contenedor, stop para detenerlo, kill el cual elimina el contenedor, cabe mencionar que para utilizar esta opción se pregunta con anterioridad si este realmente se desea eliminar, restart el cual sirve para reiniciar el contenedor en caso de instalar un plugin o en caso de que sea necesario por alguna falla, pause el cual detiene el contenedor momentáneamente pero sin llegar a cerrarlo, resume el cual hace que continúe el curso del contenedor en caso de que se seleccione la opción pause. La ventaja de portainer es que con un click se puede acceder a las opciones del contenedor, y poder modificarlo en caso de que sea necesario. La configuración de node-red básica no es suficiente ya que se tienen que agregar librerías para la elaboración del proyecto en concreto, siendo las siguientes:

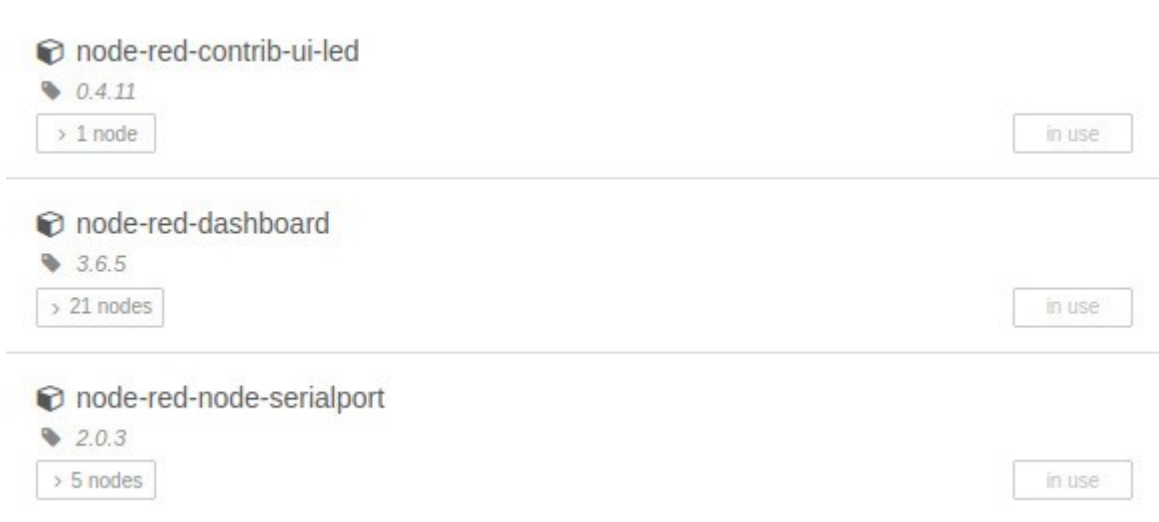


Figura 4.4.11

En node red se tiene una gran cantidad de librerías, sin embargo, estas tres librerías mostradas en la figura 4.4.11 son las principales, siendo dos de ellas para la interfaz gráfica y una para la comunicación serial. Comenzando con node-red-dashboard, estos nodos permiten la elaboración de una interfaz gráfica añadiendo botones, graficas, cuadros de texto, iconos manteniéndolo personalizable y ajustable al tema del usuario en cuestión, al ser un control de accesos se necesitan ayudas visuales, por eso la implementación de node-redpcontrib-ui-led el cual agrega un led con 3 estados, activo, desactivado e inactivo, el cual me sirvió para la implementación de la interfaz gráfica final. Por último, se tiene node-red-node-serialport el cual agrega la comunicación serial entre la ESP32 y node red, permitiendo tener todas las comunicaciones en un solo ecosistema.

Rabbitmq.

Una vez utilizadas las herramientas de portainer y node-red así como su interconexión, tuve que agregar un tercer elemento, un cluster personalizado a las necesidades del proyecto, en este caso rabbitmq el cual tiene como por objetivo llevar registro de las solicitudes de las plumas del instituto tecnológico de Chihuahua, procediendo con su instalación se tiene que realizar el archivo .yaml como en las instalaciones anteriores, tal y como se muestra en la siguiente figura.

```

File: docker-compose.yml
1 | version: "3.8"
2 | services:
3 |     rabbitmq:
4 |         image: rabbitmq:3.8-management-alpine
5 |         container_name: my_rabbitmq
6 |         environment:
7 |             - RABBITMQ_DEFAULT_USER=guest
8 |             - RABBITMQ_DEFAULT_PASS=guest
9 |         ports:
10 |             # AMQP protocol port
11 |             - '3672:5672'
12 |             # MQTT manprotocol port
13 |             - '3883:1883'
14 |             # MQTT plugin
15 |             - '35675:15675'
16 |             # MQTT Web Example
17 |             - '35670:15670'
18 |             # HTTP management UI
19 |             - '15672:15672'

```

Figura 4.4.12

Tal como se muestra en la figura x.x.x en este caso se esta utilizando una versión específica de rabbitmq para la realización del proyecto el cual es rabbitmq 3.8 management alpine, cabe mencionar que se puede utilizar cualquier versión, todo esto depende del uso en específico que se le quiera dar o del equipo de trabajo, el nombre del contenedor será my_rabbitmq, este puede variar. En el apartado de los puertos se tienen configuraciones interesantes, como primera configuración se tiene el protocolo AMQP el cual puede ser utilizado como una opción adyacente al protocolo MQTT que se presenta en el reporte, al no ser tan altamente utilizado solo hago mención del mismo como una curiosidad, luego se tiene el protocolo MQTT el cual si es altamente utilizado a lo largo del presente reporte, este no viene instalado de manera predeterminada, sin embargo este es un plugin que se instala mediante la terminal.

Rabbitmq puede ser ejecutado y controlado desde una terminal, sin embargo, un ambiente grafico es más cómodo en este caso, en este caso en la configuración de puertos se agregó este ambiente grafico el cual proporciona un manejo visual

de la interfaz para la creación de queues y exchanges. Como mencioné, cada uno de los ambientes mencionados en el presente reporte pueden ser controlados mediante terminal, en este caso para agregar el plugin de protocolo MQTT se hizo de esa manera ya que garantiza una instalación correcta y funcional.

Una vez realizado la instalación básica de rabbitmq, con sus configuraciones iniciales, en portainer.io aparecerá de manera automática la opción de inicializar rabbitmq, con todas sus opciones antes mencionadas, tal como se muestra en la siguiente figura:



Figura 4.4.13

Tal como se muestra en la figura 4.4.13 rabbitmq está pausado temporalmente, sin embargo, con las opciones que ya se mencionaron con anterioridad se puede utilizar sin ningún problema.

Una vez abriendo su apartado gráfico se tienen el siguiente menú:



Figura 4.4.14

En la figura 4.4.14 se tiene como principal apartado una parte de ingreso de credenciales, las cuales se configuraron previamente en el archivo .yaml de la figura 4.4.12 y su apartado de inicio de sesión o login. Una vez iniciada sesión se tendrá el siguiente ambiente grafico el cual se muestra a continuación.

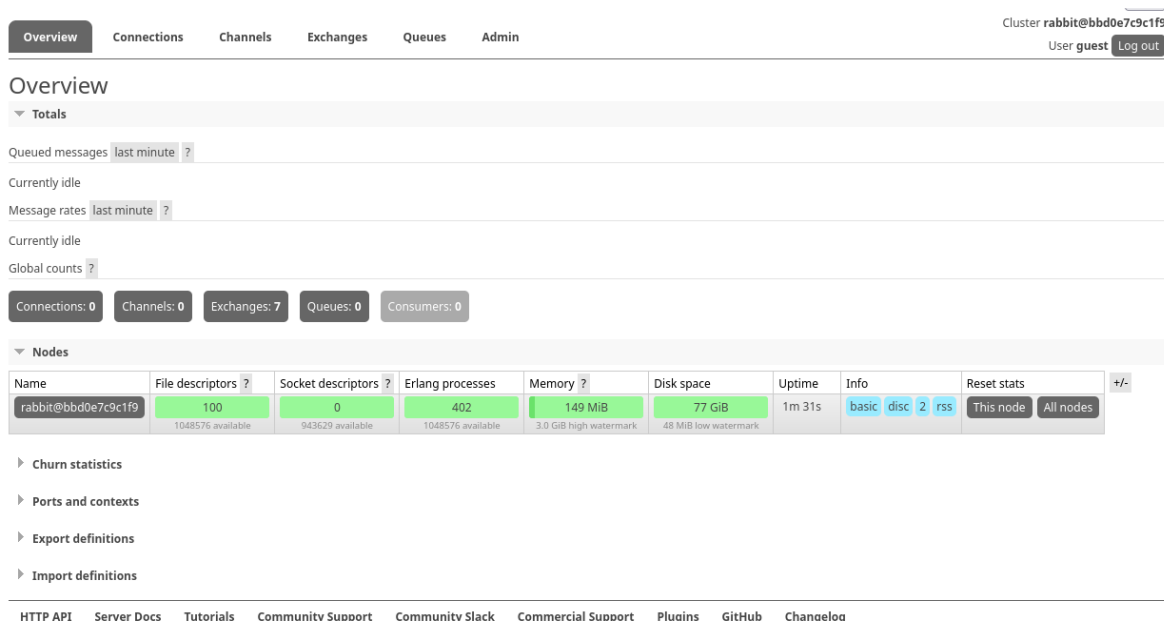
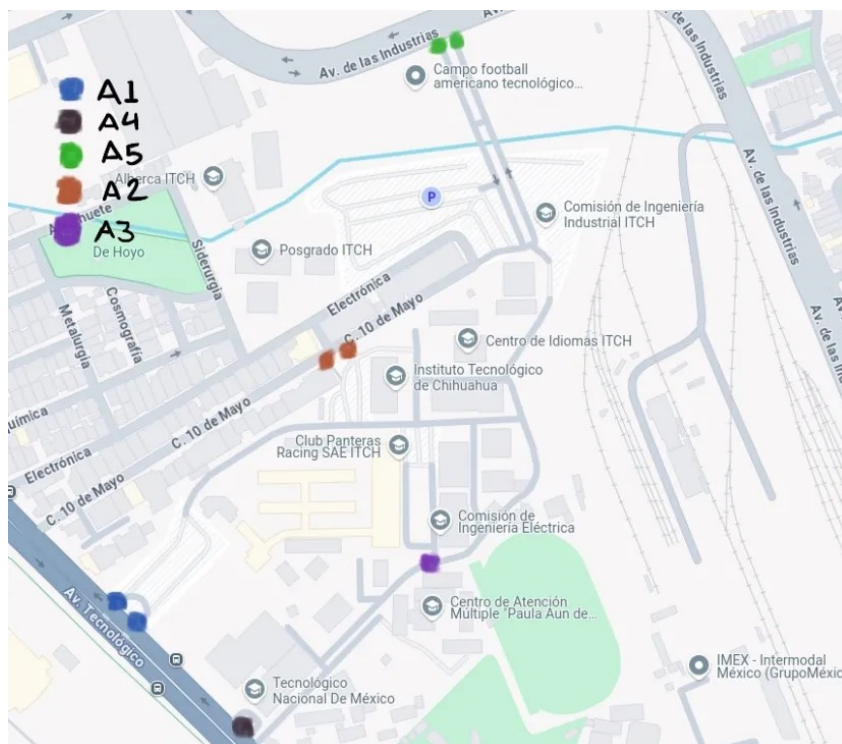


Figura 4.4.15

Esta figura representa un apartado grafico general de rabbitmq, en el cual tenemos el apartado general o overview en el cual nos da datos generales del estado del cluster así como la memoria en uso, el espacio libre del disco duro del cluster, cabe mencionar que un cluster puede ser desde una computadora de uso doméstico hasta una super computadora o servidores dedicados, en este caso las pruebas se llevaron a cabo por una computadora domestica o de uso común, luego el apartado de connections el cual es uno de los más importantes para mí, ya que muestra un estado o un listado de las conexiones al cluster, en este caso las conexiones de node-red, de esta manera podemos saber de manera visual si hay un problema de comunicación entre rabbitmq y node red o una falta de la misma, pasando a los exchanges, esto es el apartado más importante de toda la configuración ya que estos recibirán las solicitudes de node-red y las ESP32 su funcionamiento será explicado detalladamente más adelante, por ultimo las

que nos servirán para llevar registro de cada una de las solicitudes, también su configuración será detallada más adelante.

En términos generales se utilizó el protocolo MQTT como protocolo general de comunicación, el cual en parte es cierto, sin embargo la manera en que se envían y reciben los mensajes entre las distintas plataformas utilizadas es diferente, al inicio de este capítulo se mostraba la comunicación entre la ESP32 y node-red utilizando el protocolo MQTT, el protocolo MQTT en términos generales es el uso de tópicos y slashes así como numerales o almohadillas o signo de suma, estos se pueden observar en el capítulo 3 del presente reporte.



Tal como se muestra en la figura 4.5.1 se tomaron en cuenta los principales accesos en el instituto tecnologico de chihuahua, representados en 5 colores distintos, los cuales se dividen a su vez en 5 areas, estos accesos son de entradas y salidas vehicular, la representacion de estas areas es importante para poder observar el sistema de manera adecuada.

El protocolo MQTT que se va a seguir es el siguiente `tec1/A()/entrada o salida/on o off`, para clarificar tenemos como nivel mas alto `tec1` el cual representa la instituto tecnologico de Chihuahua, luego el area con un parentesis el cual puede ser cualquiera de las 5 antes mencionadas, puede ser una entrada o una salida, el `on` y el `off` representa el estado de la pluma, cuando esta esta abierta o cerrada, esta ruta es la mas basica, cabe mencionar que se pueden agregar mas niveles, tales como errores o desconexiones, estos pueden estar en niveles mas bajos, como se vera posteriormente.

La ESP32 maneja el uso de slash tanto para su envio de datos y node-red los utiliza para sus subscripciones, tal como se muestra en las siguientes imagenes:



Figura 4.5.2

```
if (valTpc == "tec1/a1/entrada/on") {  
  Serial.println("LED encendido: tec1.a1.entrada.on");  
  turnLed(1);  
}
```

Figura 4.5.3

Como primera parte tenemos la figura 4.5.2 la cual nos muestra el protocolo MQTT de suscripción en node-red el cual utiliza el formato antes mencionado, con el uso de la almohadilla el cual representa una obtención de datos generales, en este caso del último nivel, `on` y `off` para poder obtener información de los dos estados. Por otra parte, se tienen la figura 4.5.3 la cual es una extracción del código utilizado en la ESP32 el cual envía al tópico un estado de encendido, pero

lo importante es observar el uso de este formato el cual también cumple con las especificaciones antes mencionadas.

Cabe mencionar que para cada una de las áreas se necesitara una ESP32 dedicada que cumpla con su función, con una configuración determinada siguiendo este estándar, por eso la importancia de la figura 4.5.1.

Sin embargo, una comunicación entre la ESP32 y node-red no es suficiente, necesitamos un cluster el cual lleve registro de cada uno de los mensajes que se envíen, para esto utilizamos rabbitmq, el cual es una excelente opción como se mencionaba previamente a lo largo de este reporte.

El uso de rabbitmq es fundamental, anterior mente se mencionaba los exchanges, en este apartado se verán en detalle el uso de cada uno. Cada exchange tienen un nombre y un tipo el cual será topic. Sin embargo, todos y cada uno de los mensajes pasaran por exchange llamado amq.topic el cual estará ligado a cada una de las áreas mediante bindings o ligas tal como se muestra en el siguiente diagrama:

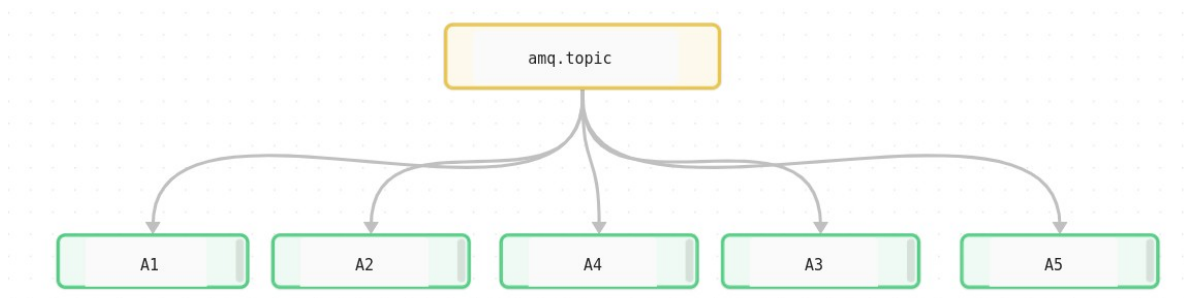


Figura 4.5.4

La figura 4.5.4 representa un diagrama general de como los mensajes de todas las ESP32 se repartiran en cada uno de sus exchanges, asi la informacion se repartira de manera adecuada, a continuacion se muestra la siguiente figura la cual es tomada directamente de rabbitmq.



Figura 4.5.5

La figura 4.5.5 representa el area 1 el la cual esta ligada con emq.topic, cabe mencionar la routing key la cual es representada con un * el cual representa en nuestra ruta original tec1, seguido del area en este caso la numero 1 y por ultimo una almoadilla la cual puede representar si es una entrada o una silida, su estado, al estar seguido de una almoadilla cualquier mensaje llegara a esa ruta mientras se cumpla la condicion del area uno como se espesifica en la figura.

Pasando a las queues, las cuales como se menciono anteriormente van a ser caracterizadas por la recepcion de datos tenemos las siguientes:

tec1.a1.entrada.error	classic
tec1.a1.entrada.off	classic
tec1.a1.entrada.on	classic
tec1.a1.salida.error	classic
tec1.a1.salida.off	classic
tec1.a1.salida.on	classic
tec1.a2.entrada.error	classic
tec1.a2.entrada.off	classic
tec1.a2.entrada.on	classic
tec1.a2.salida.error	classic
tec1.a2.salida.off	classic
tec1.a2.salida.on	classic
tec1.a3.entrada.error	classic
tec1.a3.entrada.off	classic
tec1.a3.entrada.on	classic
tec1.a4.entrada.error	classic
tec1.a4.entrada.off	classic
tec1.a4.entrada.on	classic
tec1.a4.salida.error	classic
tec1.a4.salida.off	classic
tec1.a4.salida.on	classic
tec1.a5.entrada.error	classic
tec1.a5.entrada.off	classic
tec1.a5.entrada.on	classic
tec1.a5.salida.error	classic
tec1.a5.salida.off	classic
tec1.a5.salida.on	classic

Figura 4.5.6

Como se puede observar en la figura 4.5.6 cada una de las queues sigue un formato o ruta definido con características similares entre ellos, como dato general todos tienen un inicio en tec1 el cual representa que todas estarán ubicadas en el instituto tecnológico de Chihuahua, seguido de las áreas de la 1 a la 5 como se menciona con anterioridad, seguido de entrada o salida el cual representa la salida de vehículos o la entrada de la misma por ultimo su estado, en este caso representando si la pluma esta arriba o esta abajo representado por un on y un off. Cabe mencionar que en rabbitmq se utilizaran puntos y asteriscos, los cuales representan el formato MQTT siendo los puntos los remplazos de los slashes y los

asteriscos en cierta medida los remplazos de las almohadillas, todas estas van ligadas a sus distintas áreas tal como se muestra en el siguiente diagrama:

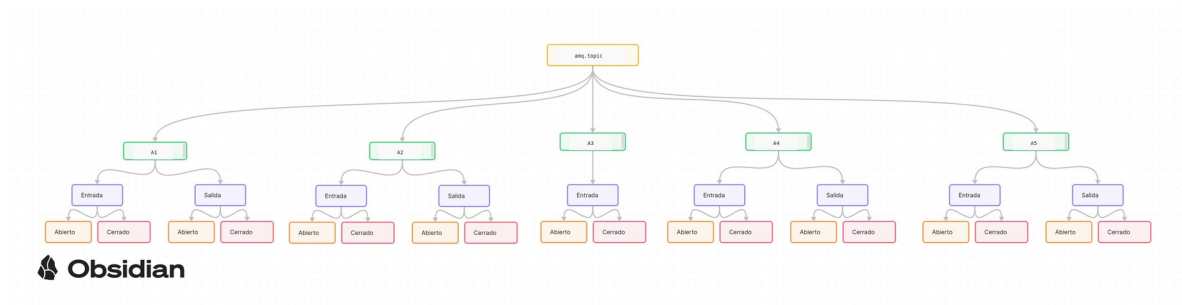


Figura 4.5.7

En el presente diagrama de la figura 4.5.7 se puede observar como se realiza el reparto de cada uno de los mensajes, desde amq.topic a cada una de las áreas y dependiendo de cada una de las áreas serán dirigidos a su área respectiva que a su vez identificará si esta es una entrada o una salida y por último el estado de cada una. Cabe mencionar que a pesar de que los mensajes sean enviados por parte de node-red estos siguen el formato de rabbitmq continuando con el uso de puntos y asteriscos, tal como se muestra en la siguiente figura:

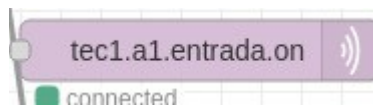


Figura 4.5.8

Tal como se muestra en la figura 4.5.8 la manera en la que node-red y rabbitmq se comunican es mediante el formato de puntos y asteriscos, pero la manera en la que rabbitmq y node-red se comunican es mediante el uso de slashes y almohadillas, esto es muy importante de mencionar ya que si no se tiene en cuenta esta forma de comunicación no se realizaría una conexión adecuada entre ambas entidades.

4.6 Diseño e implantación de la interfaz gráfica.

Una vez realizada una interconexión exitosa entre las tres principales plataformas, ESP32, node-red y rabbitmq y al estar utilizando docker compose en específico portainer se tiene el ambiente completo para realizar la programación de los ambientes de manera exitosa siendo en general configuraciones de node-red, una vez realizadas las pruebas de envío y recepción de un mensaje simple tal como se muestra en la siguiente figura:

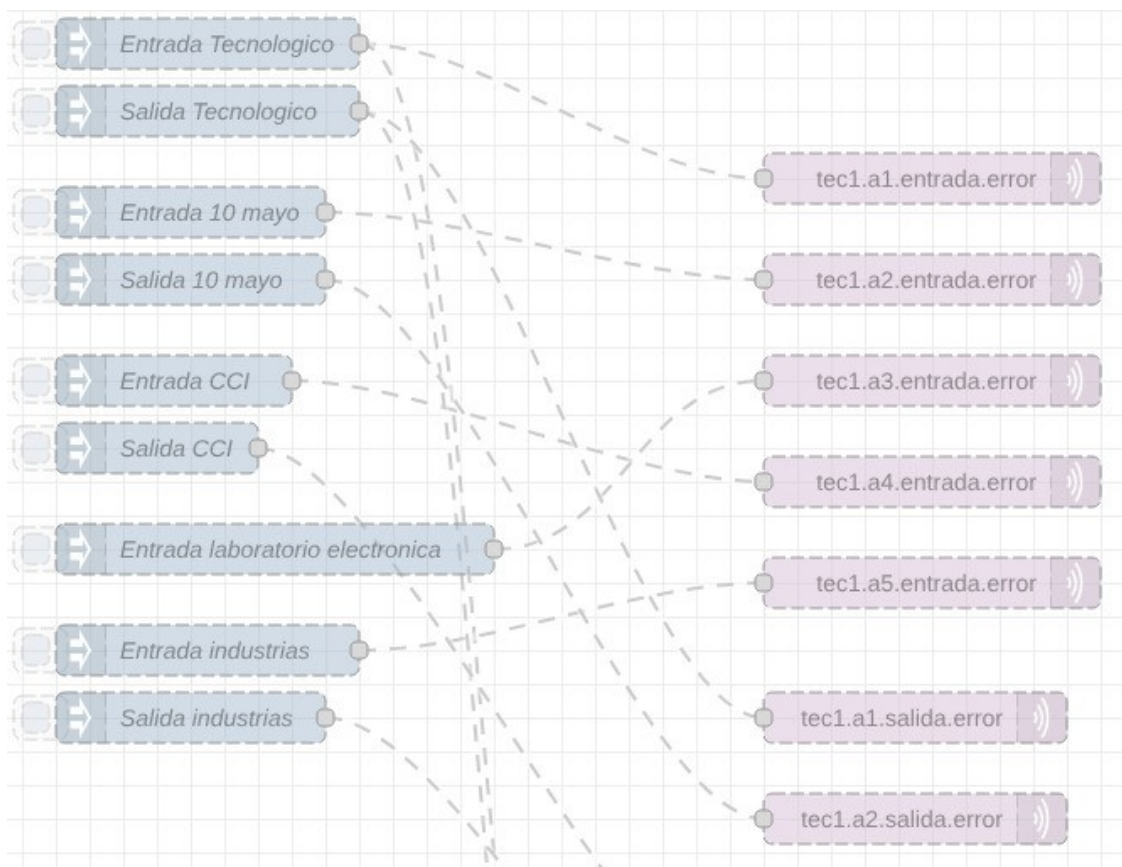


Figura 4.6.1

En el cual son mensajes simples los cuales se envían a rabbitmq en cada uno de los exchanges de manera individual para corroborar su funcionamiento tal como se muestra en la siguiente figura:

Queue tec1.a1.entrada.error

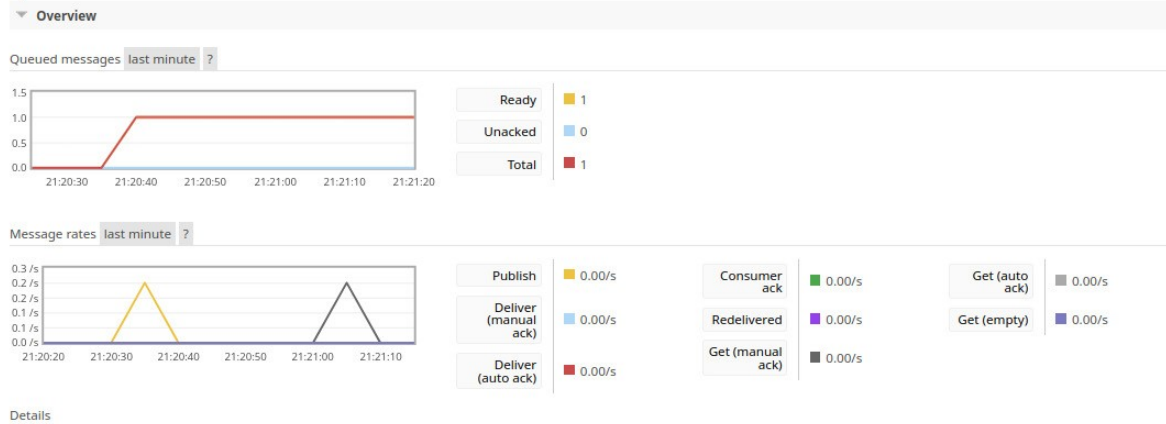


Figura 4.6.2

En la cual se puede observar una gráfica de la queue del área 1 específicamente de una entrada la cual está recibiendo el mensaje de manera adecuada, esta prueba se realizó con todas las áreas. Cabe mencionar que rabbitmq recibe los mensajes en orden de recepción, lo cual es una ventaja ya que los envía de la misma manera, entonces para entradas con alto flujo no representa un problema al momento de ser utilizado, una vez corroborado que los exchanges funcionan de manera adecuada y cada una de las queues, pase al siguiente paso el cual es realizar una interfaz gráfica, para el cual se instalaron ciertas librerías mencionadas en el apartado 4.4 del presente capítulo, las librerías mencionadas se utilizaron tal como se representa en la siguiente figura:

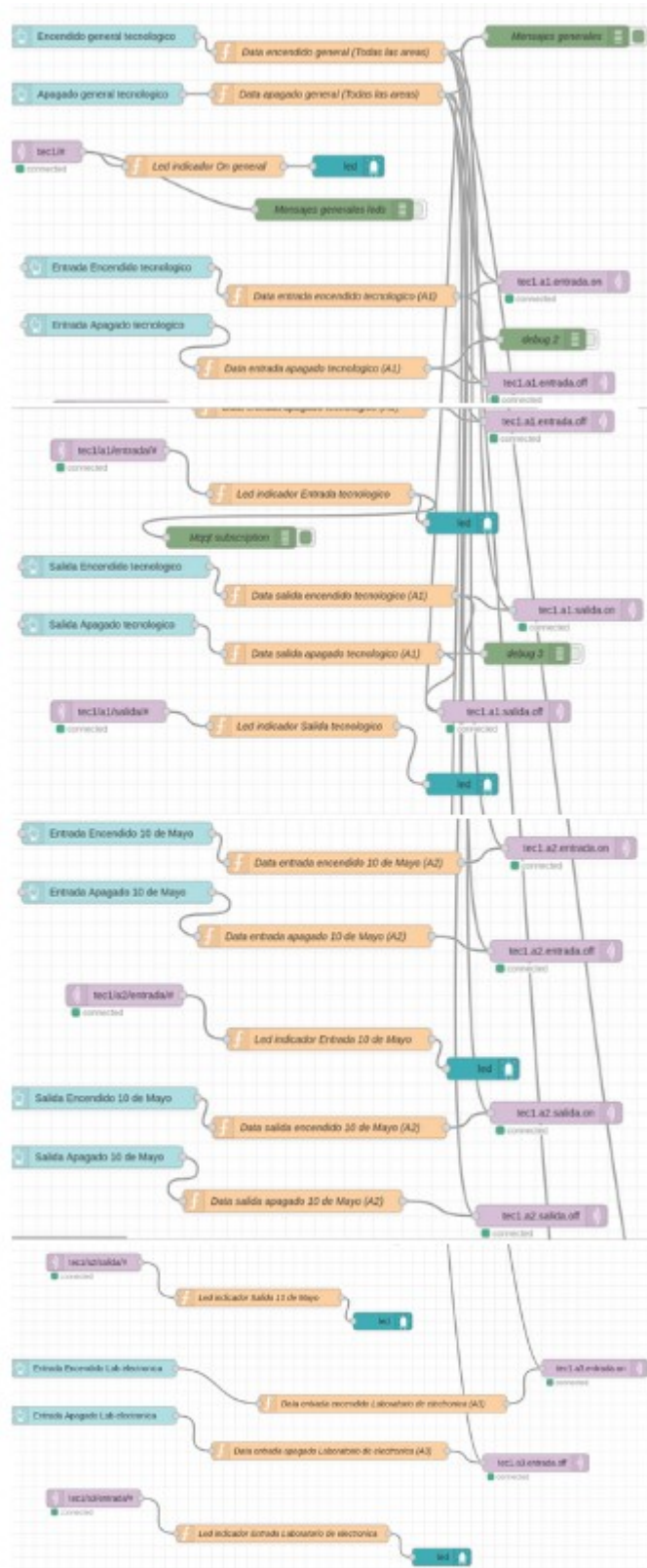


Figura 4.6.3

A grandes rasgos la figura 4.6.3 implementa 5 elementos importantes.

- Mqtt in
- Mqtt out
- Function
- Button
- Led

Estos elementos mencionados son la clave principal de la elaboración de la interfaz, mqtt in nos sirve para recibir mensajes por parte del cluster, en este caso un estado para mandarlo a un led indicador, cabe mencionar que node-red utiliza el lenguaje de JavaScript, el cual se parece a C++ o otros lenguajes de bajo nivel, al ser un control de accesos vehiculares, primero se tiene que saber el estado de la pluma, segundo la hora, el día y el año en que se realizó la acción, para esto se utilizó el nodo de función en su gran mayoría, tal como se muestra en la siguiente imagen:

```
1  const d = new Date();
2  if(msg.payload == "On general " + d.toLocaleString())
3    msg.payload = true;
4  else if(msg.payload == "Off general " + d.toLocaleString())
5    msg.payload = false;
6  return msg;
```

Figura 4.6.4

En la figura 4.6.4 se muestra la creación de una constante d la cual va a ser new y llamará la función Date o fecha, en el payload dependiendo de la condición que se cumpla se tendrá un mensaje de On u Off el cual representa el estado de la pluma más la constante antes mencionada siendo esta una cadena de datos los cuales se verán reflejados en rabbitmq de la siguiente manera:

Exchange	amq.topic
Routing Key	tec1.a1.entrada.on
Redelivered	<input type="radio"/>
Properties	delivery_mode: 1 headers: x-mqtt-dup: false x-mqtt-publish-qos: 0
Payload 43 bytes Encoding: string	On Entrada tecnologico 6/2/2025, 9:42:20 PM

Figura 4.6.5

En la figura 4.6.5 se tiene un ejemplo de cómo rabbitmq recibe los mensajes antes mencionados representados en el payload de la figura, de esta manera se puede tener un control preciso de los mensajes y del tiempo exacto en los cuales estos se envían ya que esta puede ser información muy importante en el futuro.

Luego se utilizaron distintos grupos para la creación de la interfaz gráfica, tales como el título principal llamado Control de acceso para barreras vehiculares en el Itch, la ventaja de utilizar node-red es que se pueden tener distintas interfaces coexistiendo entre si, el nombre antes mencionado es la interfaz principal, luego se creó un grupo llamado RabbitMQ el cual será el panel de control principal para el administrador el cual tendrá control total de cada una de las áreas mencionadas en la figura 4.5.1 siendo estas controladas por botones tal como se muestra en la siguiente figura:

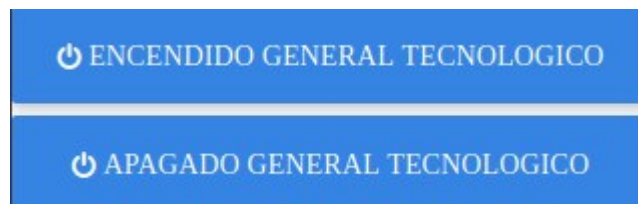


Figura 4.6.6

A su vez la interfaz necesita indicadores y para eso utilice los nodos de led los cuales tienen 3 estados siendo estos activado, desactivado y un tercer estado el cual puede ser cualquier tipo de variable, en este caso tal como se mencionó al

inicio de este capítulo, se tiene un control general y un control específico, el tercer estado es para el control general el cual desactiva los indicadores de las áreas específicas y activa el del control específico el cual a su vez cuenta con los mismos 3 estados ya que este pasa a tercer estado cuando el control de áreas específicas se está utilizando, tal como se muestra en la siguiente figura:



Figura 4.6.7

Para finalizar y teniendo cada uno de los elementos vistos en profundidad se tiene una interfaz gráfica amigable con el usuario u operador la cual se muestra en la siguiente figura:



Figura 4.6.8

Tal como se muestra en la figura 4.6.8 se tiene la sección principal o el panel de control con todos sus elementos subdivididos en este caso con sus nombres específicos y sus iconos de encendido y apagado con sus respectivos leds indicadores los cuales reciben respuesta directamente desde el cluster lo cual asegura que el mensaje o mensajes son recibidos correctamente por parte de la interfaz de usuario, sin embargo en el menú de la derecha se tienen más apartados tales como el área de subscripciones.

Pasando al área de subscripciones se realizó un código más simple ya que estas no requieren muchas operaciones tal como se muestra en la siguiente figura:

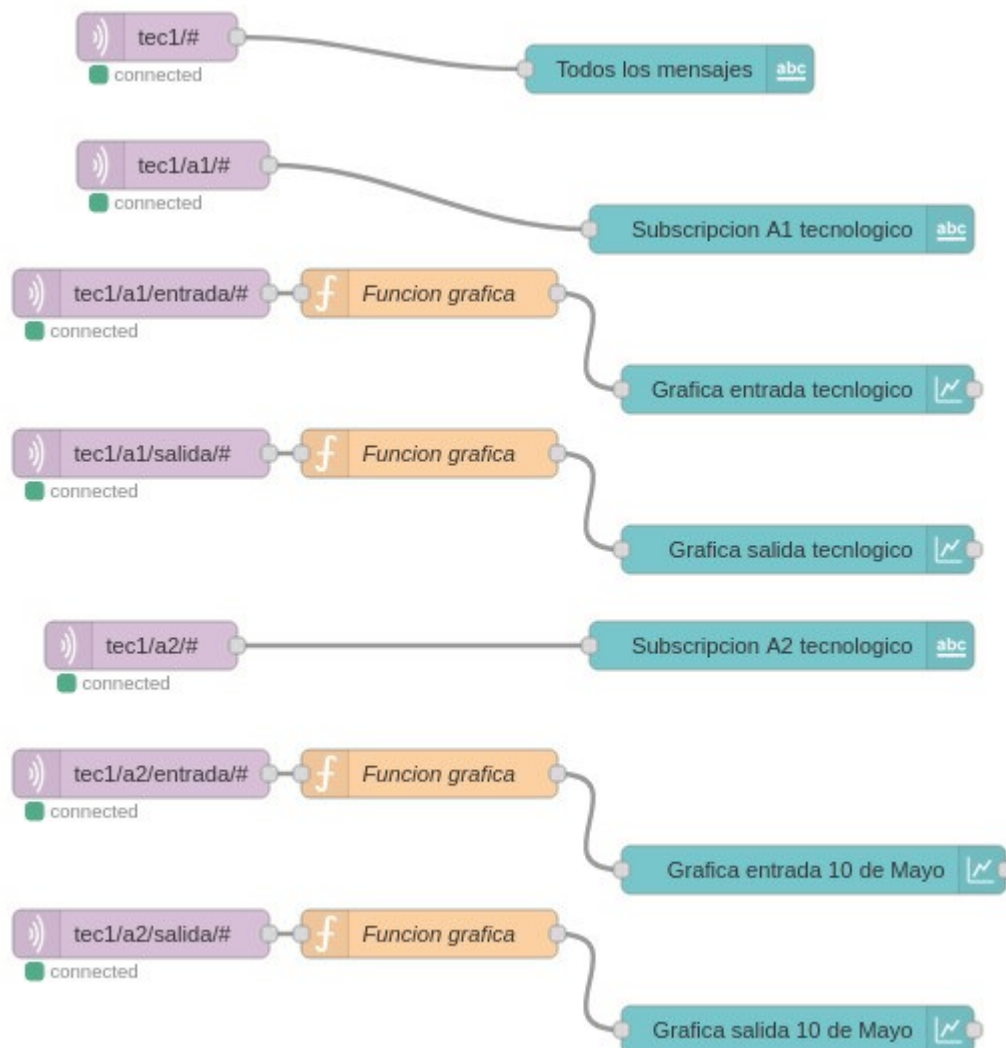


Figura 4.6.9

Principalmente se utilizaron 3 nodos.

- Mqtt in
- Function
- Cualquier nodo indicador, grafico, o mensaje

Se realizaron dos tipos de interfases de suscripción principales, siendo la suscripción general, que a mi consideración solo tiene que recibir mensajes de texto la cual luce de la siguiente manera:



Figura 4.6.10

Tal como se muestra en la figura 4.6.10 se muestra el último mensaje enviado de manera general, este cambia dependiendo del uso y se tiene un control en tiempo real de cada uno de los movimientos mostrando el estado de la pluma, el área en específico, el día, el mes y el año, así como la hora específica de la acción, en el cual se puede observar de manera detallada que es lo que pasa en tiempo real.

El segundo tipo de suscripción se representa en el área 2 el cual se muestra en la siguiente figura:

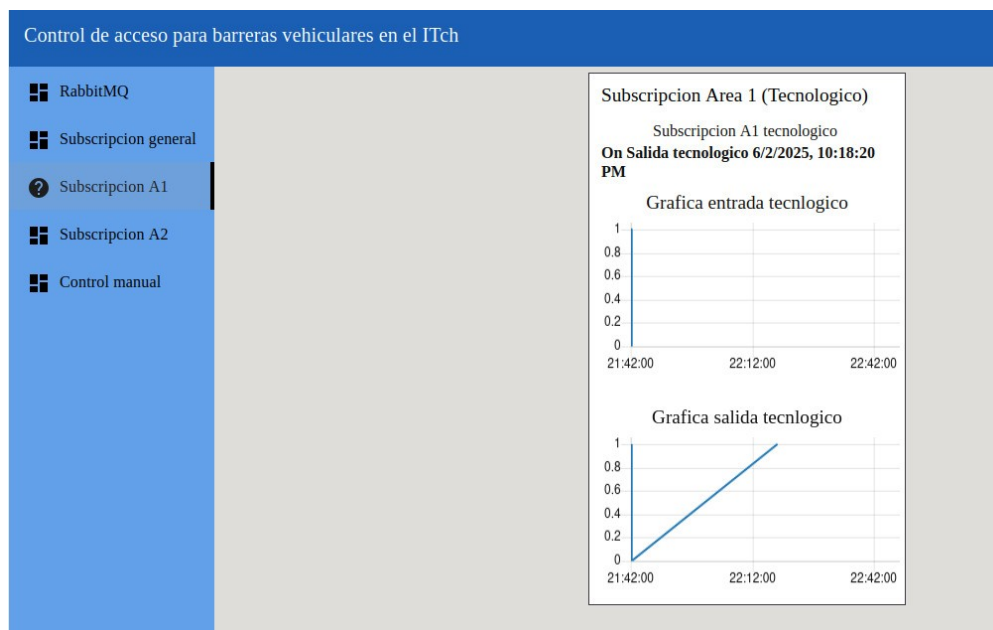


Figura 4.6.11

En la figura 4.6.11 tiene en común la representación del último mensaje enviado al cluster pero además se representa una gráfica en tiempo real de los accesos, el

cual nos brinda información importante siendo el 1 una activación y el cero la desactivación de la pluma tanto de la entrada como de la salida del área, así como su tiempo de respuesta o el tiempo en el que la pluma se mantuvo arriba o abajo, esto brinda información que puede ser utilizada en la capítulo 2 del presente informe, cabe mencionar que yo utilice estos dos tipos de interfaces debido a que considere que puede dar la mayor cantidad de información sin ser agresivo para el subscriptor.

4.7 Conexión serial entre la ESP32 y node-red

La interconexión entre node-red y ESP32 no solamente se limita en el protocolo MQTT, sino que a si vez puede tener una interfaz dedicada a la conexión serial, ya que la ESP32 cuenta con esta, estos nodos son los siguientes:

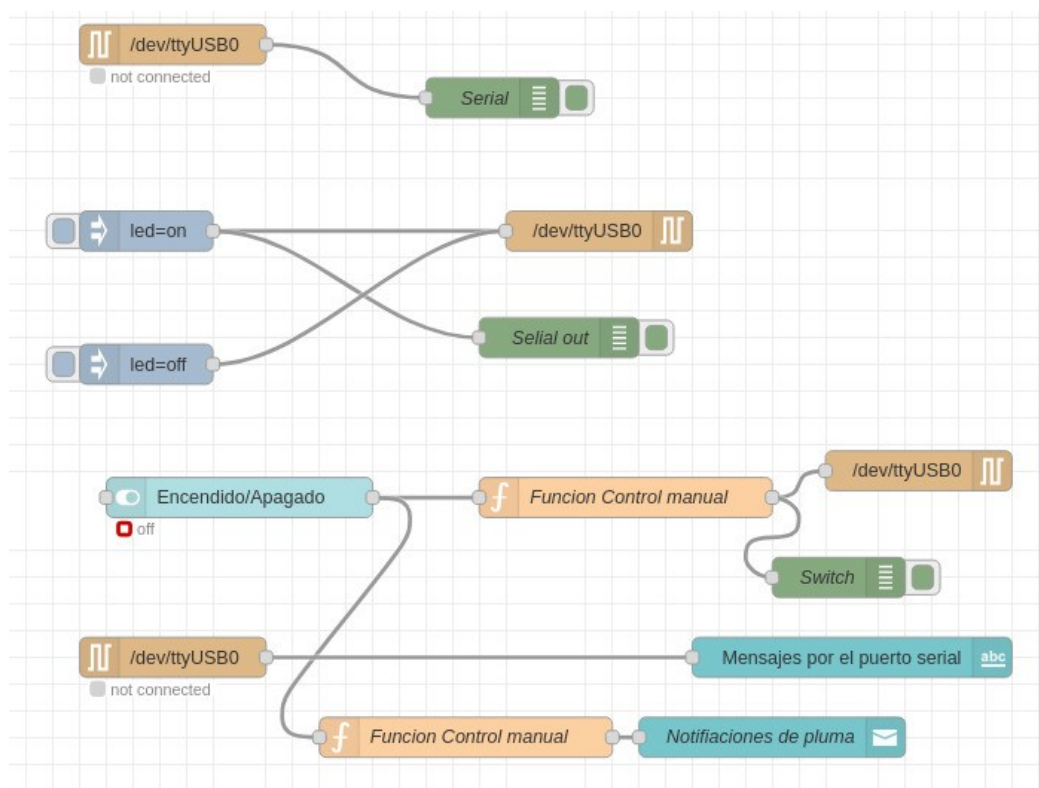


Figura 4.7.1

Tal como se muestra en la figura 4.7.1 se tienen los siguientes nodos:

- Serial in
- Serial out

- Inject
- Debug
- Notificaciones
- Mensajes de texto

Como prueba se colocaron los nodos de inject y debug para poder observar la respuesta específica de la ESP32 y ver si esta funcionaba correctamente una vez corroborado utilizaría un slider que hace la función de un switch el cual levanta y baja la pluma, lo utilice debido a que este es el más gráfico para un operador para conocer el estado de una pluma, tal como se mostrara más adelante, a su vez se utilizó un apartado de notificaciones las cuales se mostrarán en la parte superior derecha para indicar y dar más información del estado de la pluma tal como se muestra en la siguiente figura:

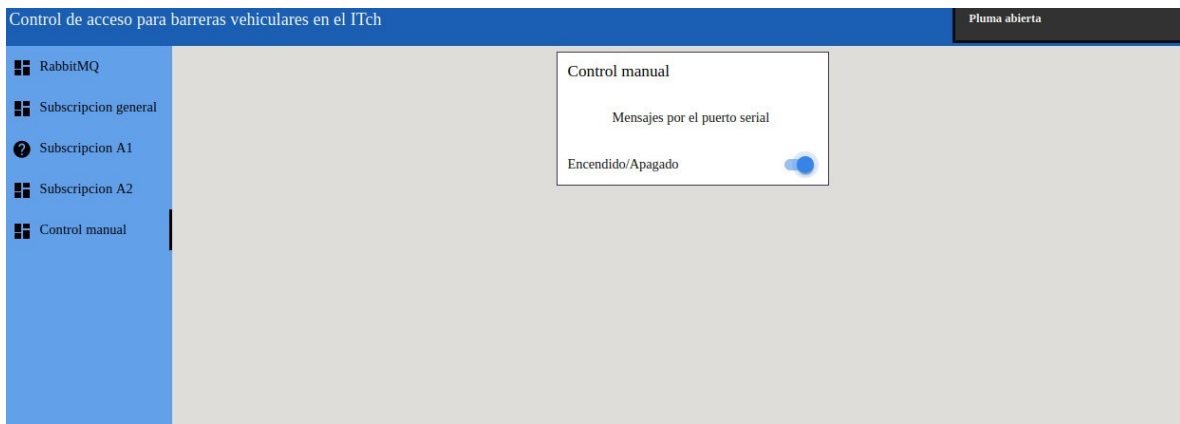


Figura 4.7.2

Tal como se observa en la figura 4.7.2 se tiene una interfaz de usuario minimalista contando solamente con la información necesaria para su operación. Con esto se concluye el uso total de cada uno de los apartados gráficos del control de barreras vehiculares.

4.8 Explicación del código implementado en la ESP32.

Como uno de los apartados finales decidí implementar la explicación del código utilizado, ya que este tiene un grado de complejidad medio, el código es el siguiente:

```
*****
** Conexion con un servidor MQTT utilizando una ESP32 Devkit V1_1      **
** Oscar Alberto Valles Limas                                         **
** Oscar Beltran                                                       **
** Instituto tecnológico de Chihuahua                                  **
*****
*/
#include <Arduino.h>
#include <WiFiManager.h>          //https://github.com/tzapu/WiFiManager WiFi
#include <PubSubClient.h>
#include <Controlled.h> //Libreria para el control de los leds

//MQTT Setup
const char* mqttServer = "192.168.1.79"; //esta es la Ip del broker rabbitmq,
const int mqttPort = 3883;
const char* clientId = "ESP_1"; // Client ID username
const char* topic = "tec1.a1.entrada.error"; // Publish topic

WiFiClient espClient;
PubSubClient client(espClient);

//----- Publicar errores -----
void publishError(const char* msg) {
  client.publish(topic, msg);
}

void reconnect() {
  while (!client.connected()) {
    Serial.println("Intentando conexión MQTT...");
    if (client.connect(clientId)) {
      Serial.println("MQTT conectado");
      client.subscribe("tec1.a1.entrada.error");
      client.subscribe("tec1.a1.entrada.on");
      client.subscribe("tec1.a1.entrada.off");
      client.subscribe("A1");
      Serial.println("Tópicos suscritos");
    }
    else {
      Serial.print("Fallo en conexión MQTT, rc=");
      Serial.print(client.state());
      Serial.println(" intentando de nuevo en 5 segundos");
      publishError("ERROR: Fallo en conexión MQTT");
      delay(5000); // wait 5sec and retry
    }
  }
}
```

```

void setup() {
    Serial.begin(115200);

    WiFiManager wifiManager;
    //wifiManager.resetSettings(); //Esto solo es para modo de prueba, en el codigo final, hay que comentar
    //first parameter is name of access point, second is the password
    wifiManager.autoConnect("ESP32_Prueba", "Itch1234"); //Nombre del hotspot que se va a crear y su contrasena
    Serial.println("Ya estas conectado"); //Impresion de mensaje

    if (WiFi.status() != WL_CONNECTED) {
        | publishError("ERROR: No hay conexión WiFi");
    } else {
        | publishError("INFO: Conectado a WiFi");
    }

    client.setServer(mqttServer, mqttPort);
    client.setCallback(callback); // Define function which will be called when a message is received.
    setup_(); //Invoca a setup_ que se encuentra en la libreria Controlled.h

    publishError("INFO: ESP32 iniciada correctamente");
}

void loop() {
    if (!client.connected()) { // If client is not connected
        | publishError("ERROR: Cliente MQTT desconectado");
        | reconnect(); // Try to reconnect
    }

    client.loop();

    //-----Entrada serial-----
    if (Serial.available() > 0) { //Si el puerto serial esta disponible
        String str = Serial.readString(); //Se lee una cadena de caracteres
        str.trim();
        Serial.println(str);
        publishError("INFO: Mensaje recibido por puerto serial");

        if(str == "led=on") //Si la cadena de caracteres es led=on
            | turnLed(1); //Enciende el led

        if(str == "led=off") //Si la cadena de caracteres es led=off
            | turnLed(0); //El led se va a apagar
    }

    //-----Entrada del boton -----
    evalBtn(); //Se invoca la funcion para evaluar el boton
}

```

Figura 4.8.1

Como se puede observar el código a simple vista no parece tan complejo para el uso que se da a lo largo del presente informe sin embargo mientras más profundicemos en el tendrá más sentido, como mención este código no solo es de mi autoría, si no que mi asesor de residencias me apoyo en el aspecto de programación y limpieza de código, el cual se menciona en el primer apartado de la figura 4.8.1, pasando al apartado de las librerías utilizadas en el presente código se tienen las siguientes:

- Arduino.h

- WiFiManager.h
- PubSubClient.h
- ControlLed.h

Arduino.h al estar utilizando visual studio code como interprete se utilizó una extensión llamada platformio el cual nos permite utilizar el lenguaje de programación de arduino en la ESP32 facilitando su uso a nivel general, luego en la segunda librería WiFiManager esta se explica con más detalle en el apartado 4.3 del presente informe, luego tenemos PubSubClient, así como el nombre de esta librería nos indica Client siendo cliente, pub es una abreviación de publisher o publicador y sub el cual es una abreviación de Subscriber o suscriptor, en términos simples esta librería nos permite implementar el protocolo MQTT en la ESP32, por último se tiene ControlLed la cual es una librería creada por mí el cual permite el control de la pluma.

El apartado de configuración tenemos que tener en cuenta que se va a utilizar un cluster privado por lo cual se tiene una constante llamada mqttServer en el cual se pondrá la ip de nuestro servidor de rabbitmq, esta puede cambiar dependiendo de la maquina o el servidor dedicado, luego tenemos el puerto, en este caso el 3883 ya que este puerto lo definimos en el archivo .yaml el cual se puede ver detenidamente en la figura 4.4.12 del presente informe por último el clientId este puede ser cualquiera como ESP_A1, por último el tópico que vamos a publicar, esto se explica en el apartado 4.5, pasando a la función callback, esta será una función muy importante ya que esta se comunicará con el cluster y con node-red al mismo tiempo, en el apartado 4.5 se menciona como la ESP32 interpreta los mensajes, por esto se tuvo que tener muy claro como es el envío y recepción de mensajes de manera detallada, también se imprime un mensaje por el puerto serial y se utilizan palabras reservadas de la librería ControlLed la cual se explicará más adelante.

Por último hay que mencionar la librería que se creo especial para el uso y control de las plumas, la cual es llamada como ControlLed, pero antes hay que mencionar como estas funcionan en platformio.

La creacion de librerias es muy similar a la creacion de librerias de C o C++ ya que se apoya de un archivo .h el cual tendra el mismo nombre antes mencionado, ControlLed, sin embargo platformio pide una gerarquia especifica para ser utilizada de la forma en la que se utilizo en el presente informe, siendo la siguiente:

```
2 This directory is intended for project specific (private) libraries.
3 PlatformIO will compile them to static libraries and link into executable file.
4
5 The source code of each library should be placed in an own separate directory
6 ("lib/your_library_name/[here are source files]").
7
8 For example, see a structure of the following two libraries `Foo` and `Bar`:
9
10 |--lib
11 | |
12 | | |--Bar
13 | | | |--docs
14 | | | |--examples
15 | | | |--src
16 | | |   |-- Bar.c
17 | | |   |-- Bar.h
18 | | |-- library.json (optional, custom build options, etc) https://docs.platformio.org/page/librarymanager/config.html
19 | |
20 | | |--Foo
21 | | | |-- Foo.c
22 | | | |-- Foo.h
23 | | |
24 | | |-- README --> THIS FILE
25 | |
26 |-- platformio.ini
27 |--src
28 | |-- main.c
29
```

Figura 4.8.2

La figura 4.8.2 es la explicación de cómo funciona la creación de librerías privadas, la cual nos dice que se tiene que crear una carpeta que en mi caso llame Led dentro de la carpeta lib la cual se crea junto con el proyecto de manera automática y dentro de esta se creara el archivo .h que en este caso fue ControlLed.h tal como se muestra en la siguiente figura:

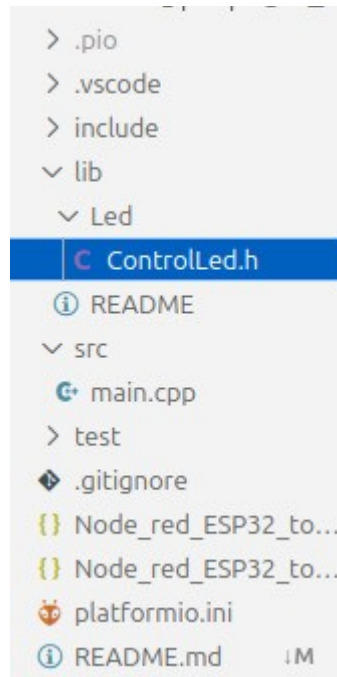


Figura 4.8.3

Una vez creado el archivo se tiene el siguiente bloque de código:

```
1 #include <Arduino.h>
2
3 bool rebote = 0; //Variable para el anti rebote
4 bool toggle = 0; //Variable para sujetar
5
6 int led = 5; //Led en el pin 5 de la ESP32
7 int led_2 = 18; //Led en el pin 18 de la ESP32
8 int btn = 21; //Boton en el pin 21 de la ESP32
9
10 void setup() {
11
12     pinMode(led, OUTPUT); //Led configurado como salida
13     pinMode(led_2, OUTPUT); //Led 2 configurado como salida
14     pinMode(btn, INPUT); //Boton configurado como entrada
15     digitalWrite(led, 0); //Estado inicial del led apagado
16     digitalWrite(led_2, 1); //Estado inicial del led encendido
17 }
18
19 void turnLed(int v) { //Funcion para encender el led va a recibir un valor entero v
20
21     if(v == 1){
22         digitalWrite(led, 1);
23         digitalWrite(led_2, 0); //Si v es igual a 1 el led se va a encender
24     }else{
25         digitalWrite(led, 0);
26         digitalWrite(led_2, 1); //Si el v no es igual a 1 el led se va a apagar
27     }
28 }
29
30 void evalBtn() { //Funcion para evaluar el estado del boton
31
32     int valBtn = digitalRead(btn); //Evaluamos el estado del boton
33
34
35     if(valBtn == 1 && !rebote) //Si el boton es presionado y ademas el valor del antirebote es 1
36     {
37         rebote = true; //El rebote se va a volver verdadero
38         Serial.println("On"); //Se imprime el mensaje de encendido
39
40         if(toggle == 0) //No se va a enganchar
41             turnLed(1); //Se enciende el led
42         else
43             turnLed(0); //En caso contrario el led se va a pagar
44
45         toggle = !toggle; //El enganche vuelve a su valor inicial
46     }
47     else if(valBtn == 0 && rebote) { //Pero si el valor del boton es igual a cero o no esta presionado
48         rebote = false; //No hay rebote
49         Serial.println("Off"); //Se muestra un mensaje de apagado
50     }
51
52     delay(100); // Retraso de 100 milisegundos
53 }
54
```

Figura 4.8.4

La figura 4.8.4 es la explicación por la cual el tamaño del código es tan reducido ya que se configura el control de la pluma completamente por separado y no es necesario configurarla más de una vez en la parte principal del código, ya que

como se observa en la figura 4.8.4 declaramos las variables principales, estas están representadas desde la línea 3 hasta la 8 en la cual tenemos una variable tipo booleana la cual se llama rebote, ya que el uso de botones provoca un fenómeno llamado rebote el cual manda más de una señal y el uso de esta variable lo elimina, luego tenemos la siguiente variable llamada toggle la cual sujeta el uso de nuestros botones físicos para no perder información, por último tenemos el uso de los leds los cuales están mapeados en los pines 5 y 18 de la ESP32, además del mapeo del botón físico en el pin 21, se decidió hacer esta función con un solo botón por practicidad y uso del operador.

Se tiene una función llamada `setup_` en la cual se configura el uso de los pines siendo los leds configurados como salidas ya que se les va a mandar una señal, el botón como entrada ya que este va a mandar una señal eléctrica a la ESP32 para el control de la pluma y a su vez se configuran los estados iniciales de cada uno de los leds siendo `led` con un estado 0 o apagado y `led2` con un estado de 1 el cual indica que estará encendido de manera inicial.

Se tiene otra función llamada `turnLed` la cual es la función la cual se muestra como palabra reservada en la figura 4.8.1 la cual evalúa el estado del led siendo este encendido o apagado.

Por último se tiene la función `evalBtn` el cual evalúa el estado del botón por medio de otra palabra reservada llamada `digitalRead` la cual evalúa el estado del pin en el cual se mapeo el botón por se tienen las condiciones de `toggle` y `rebote` las cuales están comentadas en la figura 4.8.4 siendo las líneas 37 a la 49 del código. Con toda esta configuración se tiene el uso principal del protocolo MQTT, botón físico y comunicación serial funcionando sin que una interrumpa a la otra, las tres coexistiendo sin problemas, con esto sería la explicación del código que cada

ESP32 tendría en cada una de las áreas las cuales se mencionan en el apartado de la figura 4.5.1.

4.8 Control de versiones con git y uso de la plataforma github.

Como último apartado se tiene el control de versiones que se utilizó a lo largo del proyecto, como herramienta principal es el uso de git, el cual me permite tener un control sobre cada modificación importante del proyecto así como un seguimiento de este para la elaboración del presente informe, el uso de este es esencial para los programadores ya que se puede contribuir al proyecto de manera remota sin tener ningún inconveniente para esto se utilizaron los siguientes comandos esenciales de git.

- Git init
- git status
- git add
- git commit
- git log
- git push
- git clone
- git branch

Estos comandos son básicos para el control de versiones ya que nos permite inicializar git desde cualquier bash para poder controlar cada una de las versiones, a su vez tenemos git status el cual permite saber si algún archivo ha recibido alguna clase de modificación git add para agregar esas modificaciones git log para ver cada vez los commits que se han realizado git clone para clonar cualquier repositorio en la plataforma github, git branch el cual nos permite crear ramas o subramas del proyecto principal con diferencias o agregados importantes, git push el cual sirve para poder enviar los cambios importantes a la plataforma github.

El uso de la plataforma github permite a mi asesor llevar registro de mis avances, contribuir si es necesario y tomar registro de mi avance, además el uso de git es parte de las habilidades que todo programador debe de tener al menos a nivel

básico para tener un buen flujo de trabajo todo el progreso de la realización del proyecto se encuentra en mi perfil de github para que pueda ser consultado en cualquier momento.

CAPÍTULO V.

ANÁLISIS DE RESULTADOS

Los resultados obtenidos a lo largo de la elaboración de mis residencias profesionales son aceptables, ya que se integraron las áreas de electrónica y sistemas coexistiendo en una sola. Mis conocimientos y mi formación como ingeniero electrónico me permitieron desarrollar el proyecto de manera satisfactoria desempeñando áreas de sistemas, manejo de hardware y software, siendo el hardware el uso de la ESP32, LEDs, botones, etc., y el software, programación en distintos lenguajes como Arduino, C#, JavaScript, el uso de Docker Compose, Node-RED, Portainer, Docker Hub, GitHub, distribuciones de Linux como Linux Mint y Debian, editores de texto como Nano en el caso de Linux, Visual Studio Code, extensiones específicas como PlatformIO para la programación de la ESP32. En este apartado también obtuve conocimientos más profundos de sistemas como el protocolo MQTT y AMQP, así como el uso de brokers.

Realizando una combinación de cada uno de los softwares y hardware antes mencionados, generando una coexistencia agradable, el desarrollo de interfaces de usuario agradables para un operador, así como el análisis de todo un sistema de entradas y salidas con posibles ramificaciones. El manejo de diagramas de flujo para obtener la lógica correcta, la interpretación de todo un sistema, evaluación de posibles situaciones como desconexión, recolección de datos, posibles problemas técnicos y de entendimiento general, así como el uso adecuado de entradas digitales tales como la comunicación serial y de manera remota debido al uso del protocolo MQTT, y entradas físicas de hardware como botones, el uso y manejo configuración de ambientes portables para un flujo de trabajo en equipo en el desarrollo de sistemas y/o software el cual agiliza el flujo de trabajo.

Corrección de malos hábitos de programación, limpieza de código, administración de recursos y librerías para programar de manera adecuada. Uso y manejo de

documentación así como la creación de mi propia documentación con GitHub, también el control de versiones de mi trabajo, uso de Markdown para la realización de documentación limpia y coherente entendible para cualquier persona del área de electrónica con conocimientos de programación, así como la creación de distintas ramas a la rama principal de trabajo con agregados únicos, el uso exhaustivo de la consola y el entorno de trabajo de Linux, la creación de contenedores a base de imágenes y archivos con extensión YAML para que coexistan de manera adecuada en el espacio de trabajo teniendo más de una versión del sistema trabajando al mismo tiempo sin que estas se interrumpan, el manejo adecuado de los puertos en una computadora para que puedan ser utilizados de manera eficiente, lecturas relacionadas a sistemas y programación.

5.1 Commits a lo largo de las residencias.

Al utilizar GitHub como herramienta de control de versiones, se cuenta con un registro detallado de cada uno de los avances realizados durante el desarrollo de las residencias profesionales, tal como se muestra en la siguiente figura:

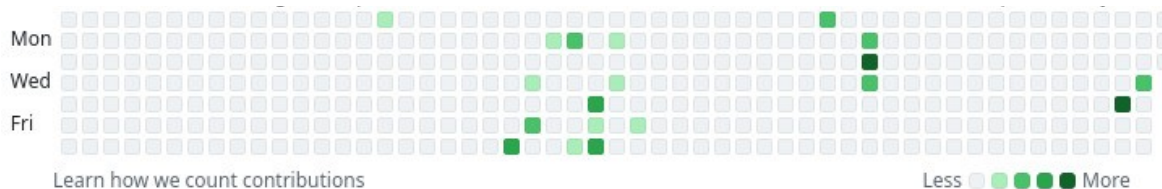


Figura 5.1.1

Como se puede observar, se tiene un historial completo de las contribuciones realizadas a lo largo del periodo de residencia, así como de los repositorios creados, los cuales se detallan a continuación.



Figura 5.1.2

En esta figura se muestran los repositorios principales utilizados durante las residencias, abarcando desde prácticas iniciales hasta versiones finales del proyecto. También se incluye la documentación elaborada en formato Markdown, con el objetivo de proporcionar una comprensión clara y estructurada tanto para el asesor como para futuras generaciones que consulten este informe como referencia.

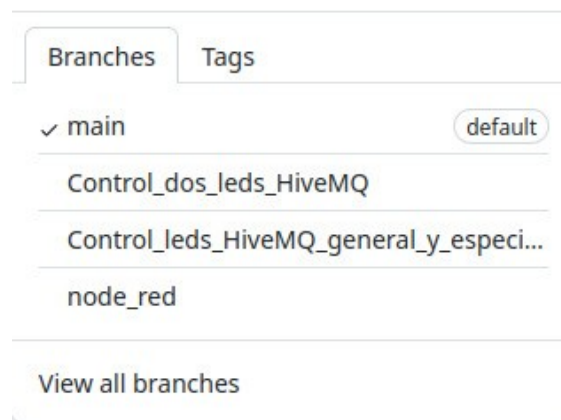


Figura 5.1.3

Cada repositorio cuenta con distintas ramas (branches), las cuales fueron creadas para subdividir el desarrollo en componentes clave. Estas ramas se separan del apartado principal (main), pero contienen aportes únicos que, en muchos casos, fueron integrados posteriormente al repositorio principal.

```

1  |## Conexión ESP32 con servidor MQTT v2.1
2
3  |![[ESP32]](https://img.shields.io/badge/ESP-32-000000.svg?longCache=true&style=flat&colorA=CC101F)(https://www.espressif.com/en/products/socs/esp32)
4
5  |![[Wokwi]](https://img.shields.io/badge/Wokwi-Simulador-blue)(https://wokwi.com/projects/423788809188749313)
6
7
8  **Material**
9  - ESP32
10 - Node-Red
11 - 2 led
12 - Calbes
13 - 2 resistencia 220 ohms
14 - 1 resistencia 1k ohms
15 - Docker compose
16 - Portainer
17 - RabbitMQ
18
19 ## **Diagrama**
20 -----
21 |![[image]](https://github.com/user-attachments/assets/7a2fea84-d13b-461e-be00-d6bc56f2a0b8)
22
23 ## **Circuito físico**
24 -----
25 |![[image]](https://github.com/user-attachments/assets/23d05e68-7ae0-4ed4-a42b-7ccb562aa570)
26 |![[image]](https://github.com/user-attachments/assets/c0b6a82f-25c6-48b6-8550-6b40e379ae2f)
27
28 ## **Librerías necesarias para node red**

```

Use **Control + Shift + m** to toggle the **tab** key moving focus. Alternatively, use **esc** then **tab** to move to the next interactive element on the page.

Attach files by dragging & dropping, selecting or pasting them.

Figura 5.1.4

Como se mencionó anteriormente, a lo largo del desarrollo del proyecto se utilizó el lenguaje Markdown para la elaboración de la documentación técnica. En esta figura se puede apreciar el uso de dicho lenguaje para documentar cada uno de los repositorios y sus respectivas ramas.

Conexion_ESP32_con_servidor_MQTT_v2_1

ESP32

Wokwi Simulator

Material

- ESP32
- Node-Red
- 2 led
- Calbes
- 2 resistencia 220 ohms
- 1 resistencia 1k ohms
- Docker compose
- Portainer
- RabbitMQ

Diagrama

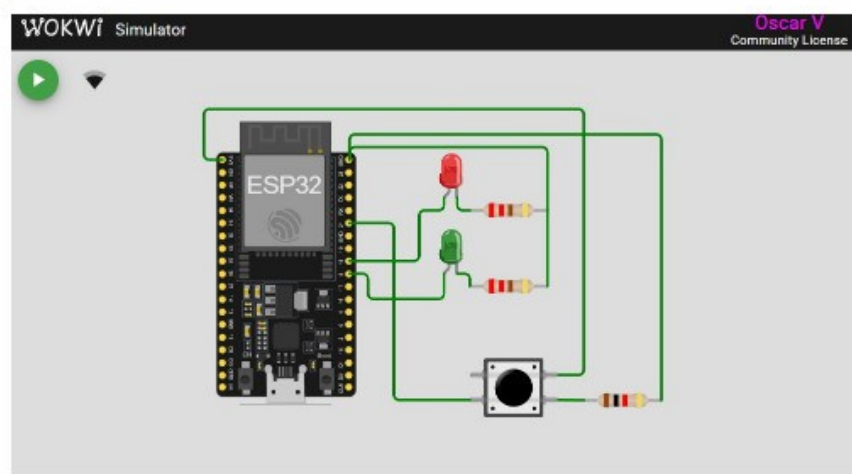


Figura 5.1.5

La figura 5.1.5 muestra un extracto de la documentación mencionada anteriormente. En ella se incluyen enlaces a páginas relevantes, listas de materiales utilizados, capturas de pantalla del funcionamiento del sistema y evidencia del progreso en el desarrollo del sistema de control de accesos.

CONCLUSIONES

Se creó un sistema para el control de accesos vehiculares en el Instituto Tecnológico de Chihuahua, implementando herramientas mencionadas en los objetivos. Sin embargo, no se utilizaron todas, ya que me centré en las adecuadas para realizar un sistema funcional. Se tienen ciertas carencias, como el respaldo de información o el guardado de datos en caso de alguna desconexión, aspectos que son importantes para el desarrollo del sistema en cuestión. Las limitaciones siguen siendo las mismas, ya que es un control de accesos vehiculares. Sin embargo, se puede extrapolar a otro tipo de sistema, como un control de acceso de personas o el registro de operadores en una fábrica, teniendo un sistema muy parecido, utilizando el protocolo MQTT así como las diversas herramientas ya mencionadas.

El uso de recursos es realmente accesible en la creación de estos sistemas, ya que al utilizar software libre como Linux no se tiene que pagar una licencia como en el uso de Windows. También es una de las razones por las cuales las grandes empresas utilizan este sistema operativo, ya que ahorra gastos a largo plazo al no tener que renovar una licencia cada cierto tiempo o actualización. El conocimiento de estas distribuciones fue trascendental para la realización de este proyecto, ya que implementé la portabilidad a distintas distribuciones comprobando que el sistema puede funcionar en diversos ambientes mientras se cumplan con condiciones.

El uso de brokers fue clave, siguiendo con la línea del software gratuito. Muchas empresas venden servicios de servidores preconfigurados con ciertas opciones de prueba tales como HiveMQ, pero al estar limitado a ciertos logins, a largo plazo estas opciones gratuitas no son tan rentables. Sin embargo, una de las competencias fue el conocimiento de servidores y se cumplió perfectamente al instalar, configurar y utilizar RabbitMQ, el cual tiene logins infinitos, exchanges infinitos y tópicos infinitos, así como seguridad del sistema debido al mapeo de puertos de Docker Compose y la asignación de credenciales personalizadas.

Desafortunadamente no se utilizó la Raspberry Pi, pero el uso del sistema Linux lo compensa, ya que el sistema operativo Raspbian es en cierta parte una derivación de Linux y al estar utilizando este sistema puedo utilizar una Raspberry Pi sin ningún tipo de problema, al ser a fin de cuentas una computadora en miniatura.

La obtención de cada uno de estos conocimientos es provechosa, ya que abre áreas de oportunidad laborales más allá de mi formación inicial en el Instituto Tecnológico de Chihuahua. El estudio de cada uno de estos conocimientos es totalmente recomendado para cualquier ingeniero electrónico.

RECOMENDACIONES

Los resultados obtenidos a lo largo de la elaboración de mis residencias profesionales son aceptables, ya que se integraron las áreas de electrónica y sistemas coexistiendo en una sola.

- Mejor administración del tiempo.
- Publicación de actualizaciones con mayor frecuencia, ya que hubo momentos en los que se realizaron cambios importantes, pero se perdió el registro de estos debido al paso del tiempo y a la omisión de los commits correspondientes.
- Mejor documentación del código. A pesar de que se realizó documentación, considero que se debió poner mayor énfasis en este aspecto para lograr los objetivos de manera más concreta.
- Comentar con mayor frecuencia los códigos, lo cual facilita la comprensión y el mantenimiento del sistema.
- Mejorar la limpieza del código, especialmente en Node-RED, donde se pudieron utilizar herramientas adicionales para optimizar su estructura y legibilidad.

FUENTES DE INFORMACIÓN

Se deberá anexar al final del reporte todas las fuentes de información que el residente consultó para llevar a cabo su Residencia Profesional consignada en el reporte.

De acuerdo con la American Psychological Association (APA), al final se enlistan las referencias consultadas, en estricto orden alfabético bajo el título de “referencias”.

Algunos ejemplos de la forma de presentar éstas referencias se presentan a continuación:

Libros:

Apellido del autor, la inicial de su nombre, año de edición entre paréntesis, nombre del libro, país de edición y editorial.

1. Anderson, D. R., Sweeny D.J., Williams T.A. (2003) Estadística para Administración y Economía. México: Thomson.
2. Goldratt, E.M. , Cox J. (2003). *La meta*. México: Castillo

En el caso de publicaciones periódicas, los datos a presentar, todos a renglón seguido, son:

- ✍ Apellido del autor, (coma)
- ✍ Inicial del nombre. (punto)
- ✍ Año de publicación entre paréntesis. (punto)
- ✍ Título del artículo. (punto)
- ✍ Nombre de la publicación en cursiva, (coma)
- ✍ Número del volumen en cursiva
- ✍ Número del ejemplar entre paréntesis, (coma)
- ✍ Número de la (s) página (s). (punto)

Ejemplo:

1. Ascanio, A. (1988). Competencias de los docentes para el desarrollo del proceso de aprendizaje en instituciones de educación superior. *Revista de Investigación Educativa*, 15(32), 1-8.

Hasta seis autores se nombran todos, tal como se indicó.

En el caso de revista exclusiva de internet:

1. Fredrickson, B. L. (2000, 7 de marzo). Cultivating positive emotions to optimize health and web-being. *Prevention and Treatment*, 3, Artículo 0001a. Recuperado el 20 de noviembre de 2000, de <http://journals.apa.org/prevention/volume3/pre0030001a.html>

Nota: pueden consultar las siguientes páginas:

<http://lia.unet.edu.ve/ant/EstiloAPA.htm>

<http://alejandria.ccm.itesm.mx/biblioteca/digital/apa/APAelectronicas.html>

<http://www.portesasiapacifico.com.mx/content/archivos/>

[Referencias_APA_2012.pdf](#)

y ver en Word. Barra de tareas. Referencias. En el software de office.

ANEXOS

Son todos aquellos documentos que nos permiten tener un soporte a la información reportada. No se paginan, se separan por hojas con su título y se ordenan usualmente con letras, ejemplo:

ANEXO A “Nombre del documento”.

<https://github.com/VALO64>

GLOSARIO

Variables booleanas

Representan valores lógicos: verdadero (true) o falso (false). Se usan para tomar decisiones en los programas.

Enteras

Tipo de variable que almacena números enteros, sin decimales.

Dobles

Tipo de variable que almacena números con decimales de doble precisión.

Lenguaje de programación de bajo nivel

Lenguaje cercano al lenguaje máquina, como el ensamblador. Ofrece control directo sobre el hardware.

Librerías en programación

Conjunto de funciones y recursos reutilizables que facilitan tareas comunes en el desarrollo de software.

Driver en programación

Software que permite que el sistema operativo se comuniquen con hardware específico.

UI en programación

Interfaz de Usuario (User Interface): parte visual de una aplicación con la que interactúa el usuario.

Distribución de Linux

Variante del sistema operativo Linux que incluye el núcleo y un conjunto de herramientas y software.

Bash en Linux

Intérprete de comandos por defecto en muchas distribuciones de Linux. Permite ejecutar scripts y comandos del sistema.

Fish en Linux

Shell interactiva alternativa a Bash, conocida por su facilidad de uso, autocompletado inteligente y sintaxis clara.

Red local

Conjunto de dispositivos conectados entre sí dentro de un área limitada, como una casa u oficina.

Puertos en computadoras

Puntos de conexión lógica que permiten la comunicación entre dispositivos o aplicaciones a través de una red.

Comunicación serial

Método de transmisión de datos bit a bit a través de un solo canal. Común en microcontroladores y dispositivos embebidos.

Credenciales de acceso

Información (como usuario y contraseña) que permite autenticar a un usuario en un sistema o red.

Broker

Servidor intermediario que gestiona la distribución de mensajes entre dispositivos en protocolos como MQTT.

MQTT tópicos

Canales jerárquicos donde se publican y suscriben mensajes en el protocolo MQTT.

Slash en protocolo MQTT

El carácter / se usa para separar niveles jerárquicos en los tópicos MQTT.

Almohadilla en protocolo MQTT

El símbolo # se usa como comodín para suscribirse a todos los subniveles de un tópico en MQTT.

Nodos en Node-RED

Componentes visuales que representan funciones o dispositivos en un flujo de programación visual.

Contenedor en Docker

Entorno ligero y aislado que ejecuta aplicaciones con todas sus dependencias.

Imagen en Docker

Plantilla inmutable que contiene todo lo necesario para crear un contenedor: código, librerías, configuraciones.

Queue en RabbitMQ

Cola donde se almacenan mensajes hasta que son procesados por un consumidor.

Exchange en RabbitMQ

Componente que recibe mensajes y los distribuye a las colas según reglas definidas.

Binding en RabbitMQ

Enlace entre un exchange y una cola que define cómo se enrutan los mensajes.

Cluster en RabbitMQ

Conjunto de nodos RabbitMQ que trabajan juntos para ofrecer alta disponibilidad y escalabilidad.

LED

Diodo emisor de luz. Componente electrónico que emite luz cuando pasa corriente a través de él.