UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

"DISEÑO Y EVOLUCIÓN EN LA FABRICACIÓN DEL HARDWARE Y EL SOFTWARE PARA LA AUTOMATIZACIÓN DE CENTROS DE TRASTEROS"

TRABAJO FIN DE GRADO

Marzo -2024

AUTOR: Andrés Brotóns Martínez.

DIRECTOR: Fernando Rodríguez Mas.

1. ÍNDICE

2. ÍNDICE DE FIGURAS

3. RESUMEN

En el siguiente documento se detallan los pormenores del diseño, fabricación e instalación de un proyecto creado para la empresa Keemetrix Systems S.L. La empresa Keemetrix Systems se dedica a la automatización de centros de trasteros de una forma innovadora, simplificando los alquileres de los mismos y su utilización. Agilizan la capacidad del usuario para empezar a usar un trastero y dan seguridad al dueño del centro, ya que no necesita dar llaves a extraños ni preocuparse de tener que cambiar candados una vez finalizado el alquiler.

En el trabajo se habla de cuáles fueron las necesidades de la empresa y de que recursos disponían. Se describe detalladamente cuál fue la base de la que se partió en el diseño y que se quería conseguir con todo ello. Este proyecto se centra en el diseño y fabricación del software y hardware implantado en una PCB que permite a cada usuario controlar el acceso al centro y su trastero de forma remota y le permite a la empresa saber en todo momento el estado de los dispositivos y de dichos accesos.

Para la creación de los dispositivos se emplearán los softwares de EASY EDA, VISUAL STUDIO CODE y AWS IOT. Para los test del equipo se empleará el programa VISUAL STUDIO CODE y para la solución de problemas físicos en el mismo un multímetro (o tester) así como soldador, estaño, quita estaño y diversos tipos de cableado y conectores.

El cerebro de los dispositivos estará basado en la tecnología ESP32 y en su conectividad Wi-Fi, también se empleará el framework de Arduino para su programación disponible para esta familia de SoCs.

4. MOTIVACIÓN PARA EL PROYECTO

En el año 2021, entre en contacto con la empresa Keemetrix Systems S.L. mediante unas prácticas extracurriculares a través del observatorio ocupacional de la universidad Miguel Hernández. Las competencias de dichas prácticas eran el diseño de placas de circuito impreso (PCB), programación de módulos Wi-Fi, así como investigar la implementación de la tecnología desarrollada.

Tras pasar por una entrevista de más de 20 inscritos, fui seleccionado para el puesto en el que se me revelaron mis cometidos, buscaban implementar una tecnología existente pero para un propósito concreto, la automatización de trasteros. Tras acostumbrarme a las herramientas proporcionadas por la empresa, así como la compra del material de laboratorio necesario, pasé a desarrollar la tecnología que se me requería. En agosto de ese mismo años tras varios meses de desarrollo se llevó a cabo la instalación del material en el primer centro completamente automatizado, aquí pude comprobar de primera mano el proceso de instalación y los problemas que planteaban, tomando notas de los mismo para su posterior mejora. Todos los problemas que pudieran ocurrir no se podrían solucionar acercándose rápidamente a la instalación.

El problema planteado por la empresa fue el cómo adaptar e implementar una tecnología que sabían que existía pero que desconocían como trabajar, para conseguir la reducción del cableado necesario para una instalación de este tipo, pudiendo hacer así más atractiva la inversión principal y más cómoda toda la instalación.

5. MARCO TEÓRICO

Para la correcta automatización de las entradas y las puertas de un centro de trasteros, la tecnología actual, era actuar por medio de centralitas. Una centralita está compuesta por diversos elementos que trabajan juntos, suelen estar compuestos por unos circuitos cuyos procesadores controlan la información para efectuar las aperturas a través del uso de relés integrados, estos elementos son las controladoras, un elemento de conexión a la red, ya sea un router o un hub conectado al internet proporcionado al centro, un trasformador de tensión y un temporizador electrónico que permite reiniciar el sistema en caso de mal funcionamiento del mismo y recuperarse, también conocido como watchdog. Los distintos problemas pueden ser desde una caída de red, pasando por saturación de mensajes por demasiadas llamadas consecutivas etc.

Esta centralita se instalaba en el cuarto técnico del local a automatizar. Las cerraduras de todas las puertas de los trasteros y las de las entradas del establecimiento se conectaban hasta la centralita, en un principio no supone ningún problema hasta que conforme nos alejamos de la centralita, los cables pasan a ser demasiado largos, incrementando significativamente el coste de la instalación de la instalación así como la cantidad de esfuerzo que requiere la misma.

Ante el problema identificado, Keemetrix System S.L. consideró la necesidad de encontrar una solución más eficiente. Esta solución no solo debía enfocarse en la reducción de los costes comentados, sino también en proporcionar un control integral sobre los accesos. Esto se volvía especialmente relevante debido a que la administración de las centralitas tiende a ser externalizada a terceros. Esta externalización tiene el efecto de generar un aumento significativo en el presupuesto tanto de la instalación como del mantenimiento asociado.

En la búsqueda de esta tecnología encontraron información sobre el módulo ESP32.

ESP32 es una poderosa familia de microcontroladores de bajo costo y bajo consumo desarrollada por Espressif Systems. Se ha ganado una reputación sólida en la comunidad de desarrolladores debido a su versatilidad y rendimiento[8].

El sistema ESP32 viene con un procesador de doble núcleo Xtensa LX6, lo que significa que puede manejar múltiples tareas simultáneamente[7]. Esto es particularmente útil para aplicaciones que requieren multitarea y alta capacidad de respuesta. Este elemento nos ha resultado muy útil a la hora de separar todos los elementos de conexión con las respuestas a las interrupciones.

También cuenta con una memoria flash integrada para almacenar programas y datos. Tiene una memoria RAM para almacenamiento temporal durante la ejecución del programa. La capacidad de memoria varía según la versión específica del ESP32 utilizada. Es conocido por su capacidad para proporcionar conectividad inalámbrica a través de Wi-Fi y Bluetooth. Es compatible con los estándares 802.11 b/g/n Wi-Fi en la banda de 2.4 GHz y puede alcanzar velocidades de hasta 150 Mbps. Además, cuenta con Bluetooth v4.2 y BLE (Bluetooth Low Energy) para aplicaciones de bajo consumo de energía. Está equipado con una variedad de interfaces periféricas, incluyendo I2C, SPI, CAN y PWM[1,Anexo I]. Estas interfaces permiten la conexión con una amplia gama de sensores, dispositivos y actuadores, lo que lo convierte en una opción versátil para proyectos de electrónica. Ha sido diseñado para ser eficiente en términos de energía, lo que lo hace adecuado para dispositivos alimentados por baterías y otras aplicaciones de bajo consumo de energía. Tiene modos de bajo consumo que permiten conservar energía cuando el dispositivo está inactivo. Es compatible con diversas plataformas de desarrollo, incluyendo el entorno de desarrollo de Arduino y el framework de desarrollo de Espressif, el Espressif IoT Development Framework (ESP-IDF). Esto facilita a los desarrolladores escribir código para el ESP32 utilizando herramientas familiares.

Debido a su popularidad, el ESP32 tiene una comunidad de desarrolladores activa que comparten conocimientos, proyectos y soluciones a través de foros en línea (como github) y otros recursos[8]. Esto facilita el aprendizaje y la resolución de problemas para los usuarios.

En resumen, el ESP32 es una opción sólida para una variedad de aplicaciones, desde proyectos de bricolaje y prototipos hasta productos comerciales de IoT. Su potente combinación de características, conectividad inalámbrica y bajo consumo de energía lo

convierten en una opción preferida para muchos desarrolladores. Todas las capacidades de este chip vienen especificadas en su hoja de datos adjunta en el anexo 1.

Para la implementación de la tecnología ESP32 se adquirió un circuito impreso de la casa Diy Mall.

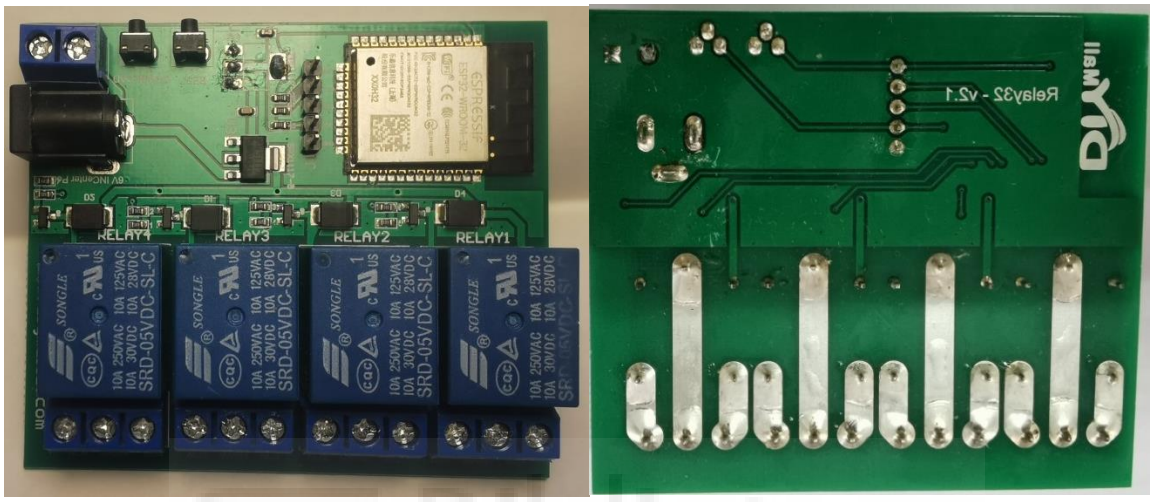


Figura 5.1 placa DIY MALL

Este circuito impreso se compone de 4 relés activables a 5 V conectados a unos bloques terminales de 3 salidas (como se puede apreciar en la parte inferior de la figura 5.1). Los cuales nos permiten conectarnos a los pines normalmente abierto (NO por sus siglas en ingles), normalmente cerrado (NC) y común (el pin que proporciona alimentación a los dos anteriores). Estos se activan gracias a un transistor que a su vez es activado por un optoacoplador controlado por el ESP32 WROOM-32, que es el chip de la familia ESP32 que incluye este modelo de circuito impreso. Esta PCB está alimentada de 5 V por lo que es necesario un conversor DC-DC para reducir el voltaje a los 3,3 V necesarios para el sistema en chip (SoC por sus siglas en ingles system on chip). La alimentación se recibe a través de una clavija de tonel estándar para cargador o por un bloque terminal con positivo y negativo, como se puede ver en la parte superior izquierda de la figura 4.1.

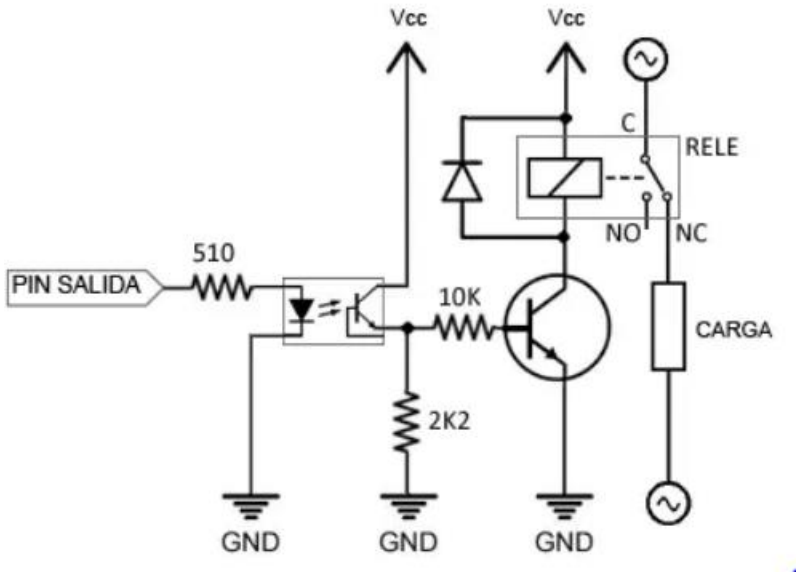


Figura 5.2 Funcionamiento.

Para la apertura de una cerradura el funcionamiento sería el siguiente[5]. El GPIO correspondiente se pondría en alto (pin de salida de la imagen 4.2) haciendo que el optoacoplador pusiera a su vez en alto la base del transistor completando el circuito que accionaria el relé. La función del optoacoplador es aislar los elementos de corriente del ESP32 pero es redundante.

Aunque en la figura 5.2 la carga está situada en el pin NO, este circuito también permite el funcionamiento opuesto, es decir conectarlo en NC, esto haría que la corriente siempre circulara por la cerradura y solo cuando el relé cambiase de estado se cortaría la corriente. Para un sistema así haría falta otro tipo de cerradura que necesita corriente para mantenerse cerrado. Este tipo de cerraduras se usan en entornos donde haya que garantizar la salida, en caso de fallo eléctrico, por un incendio u otras condiciones la puerta quedaría abierta. Esto para centros de trasteros sin embargo es inapropiado.

6. SOFTWARE EMPLEADO

El software empleado para el diseño esquemático y físico fue EASY EDA, para la programación de la PCB el Visual Studio Code y para enlazarse al servidor la herramienta web de AWS IOT.

6.1 EASY EDA:

Este es un software gratuito, de fácil acceso e implementación muy similar a EAGLE programa empleado en la asignatura de Diseño y calidad de Circuitos electrónicos. El software puede ser descargado desde su página web[9].

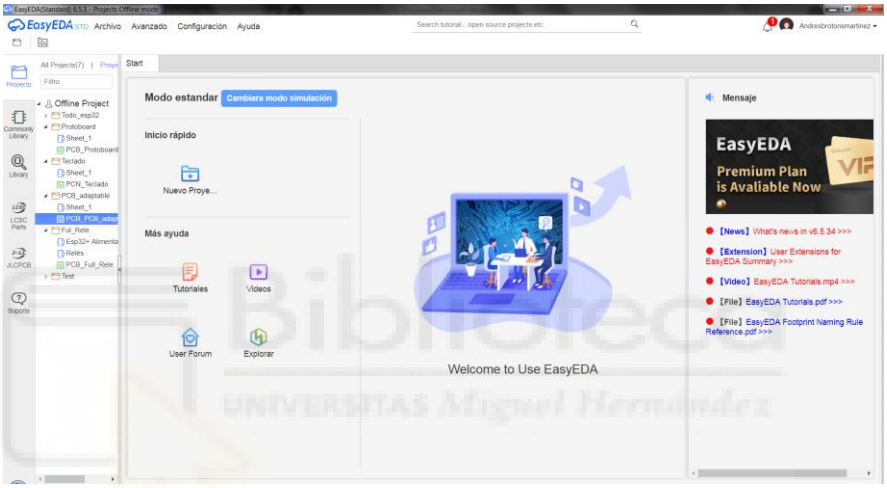


Figura 6.1 Entorno Easy EDA

O puede emplearse de forma online ya que existe un editor web.

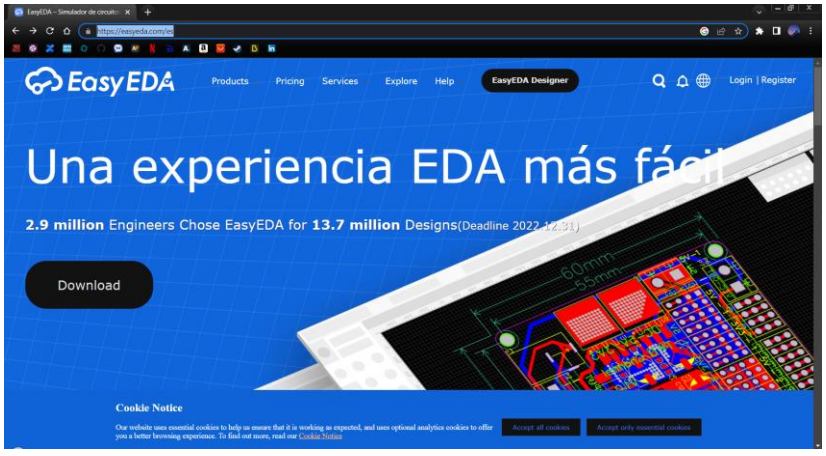


Figura 6.2 Entorno WEB de Easy EDA

El software permite almacenar los proyectos de manera offline, tanto en el propio dispositivo como en la nube del sistema. Esto facilita la difusión del trabajo y proporciona flexibilidad a la hora de trabajar desde otros dispositivos. Además el editor web está enlazado con el entorno descargado lo que aumenta su integración. Asimismo, este software se encuentra enlazado con la empresa JLPCB y su entidad hermana, LCS Componentes. Esta conexión proporciona acceso a una librería de componentes reales que incluye información detallada, como medidas y diseños originales de componentes. Esto permite acceder a datos relevantes sobre el fabricante de PCBs (JLCPCB), el inventario de los componentes deseados y sus respectivos precios unitarios, lo que simplifica notablemente la fase de preproducción y la subsiguiente fabricación. Además, el software posibilita la realización de simulaciones (aunque no se han utilizado en este caso) y la exportación de archivos GERBER, BOM y Pick & Place, que son esenciales para su fabricación.

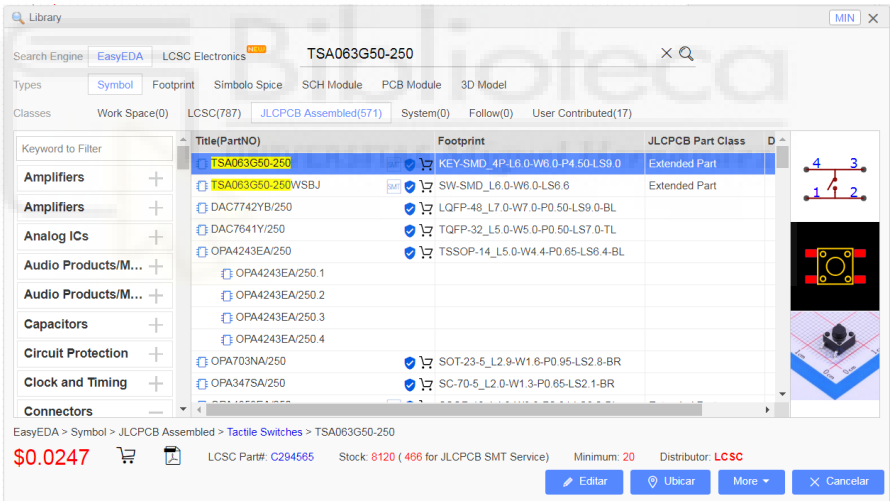


Figura 6.3 Ejemplo para botón TSA063G50-250

Diseñado el esquemático y el layout es necesario subir los archivos Gerber, pick and place y el BOM a la web de JLPCB para la fabricación del circuito impreso. Pero en lugar de tener que hacer esto, se puede acceder al generador del proyecto en dicha web directamente desde la opción fabricación del layout, JLPCB deja muchos parámetros a elegir, así como el tipo de sustrato, pudiendo llegar a fabricarse incluso en láminas

flexibles, de la misma manera nos permite decidir el tipo de cobre y color de la resina protectora y junto a otras opciones de fabricación.

Este entorno nos permite añadir los componentes por un precio extra, pedir un extensil para soldar nosotros mismos (u otra empresa) los componentes EN CASO DE SER NECESARIO. Si elegimos que JLCPCB añada los componentes, opción muy recomendable ya que evita errores de soldadura, tendríamos que escoger entre dos opciones. En nuestro caso siempre usamos la versión estándar, esta nos obliga a escoger por qué cara se deberán colocar los componentes, siempre elegimos la cara TOP (la de arriba) por comodidad a la hora de diseñar el layout, una vez elegido esto se pasa a otra pantalla en la que se añaden (en caso de no estar ya debidamente incorporados a través de la plataforma de EASY EDA) y nos mostrará un ejemplo tridimensional de la placa con los componentes colocados.

Otra herramienta de EASY EDA es que una vez diseñado el layout con todos los componentes se puede ver un modelo 3D del producto final, pudiendo confirmar que todo esté debidamente colocado y la serigrafía colocada sea legible. Este modelo también se puede exportar en caso de ser necesario.

EASY EDA tiene dos modelos para el apartado de esquemático, un modo simulación, que nos permite detallar circuitos, sus conexiones y estudiar sus salidas, voltajes y amperajes entre otros factores. El esquemático de uso común nos permite seleccionar componentes de una librería propia, exportar una librería, usar modelos, o emplear las librerías de componentes de LCS o JLCPCB, que tienden a ser las mismas, como se podía observar en la imagen se detalla que tienda lo proporciona, que precio unitario tiene y si es soportado por JLCPCB para su montaje en fabrica.

Estas librerías nos dejan seleccionar componentes y añadirlos a favoritos para poder tener un acceso rápido a componentes de uso frecuente haciendo que los nuevos proyectos sean más rápidos de generar ya que no tendríamos que buscar el componente deseado entre una gran variedad de opciones. Esta librería también cuenta con un buscador, si conocemos el nombre o número del componente se podrá encontrar facilmente.

6.2 VISUAL STUDIO CODE:

Visual Studio Code (VS Code) es un editor de código fuente desarrollado por Microsoft. Es de software libre y multiplataforma, disponible para Windows, GNU/Linux, macOS y web. Cuenta con soporte para depuración de código, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código, es personalizable permitiendo cambiar temas, atajos de teclado y preferencias. Además, ofrece una gran variedad de extensiones, posibilitando la capacidad de escribir y ejecutar código en cualquier lenguaje de programación [10].

VS Code está construido sobre el framework Electron y se integra con Azure y GitHub. Admite casi todos los lenguajes y tipos de aplicaciones, incluidas aplicaciones web estáticas, las aplicaciones Azure Functions sin servidor y los clústeres de Kubernetes.

En este caso se ha empleado la extensión PlatformIO (PIO). Es un entorno de desarrollo para sistemas embebidos muy utilizado, que ofrece un entorno de trabajo unificado, al margen del hardware o bien software que se emplee. Facilitará la gestión de la descarga de bibliotecas, la descarga de toolchains, la compilación del código fuente y la descarga del código a la placa entre otras acciones.

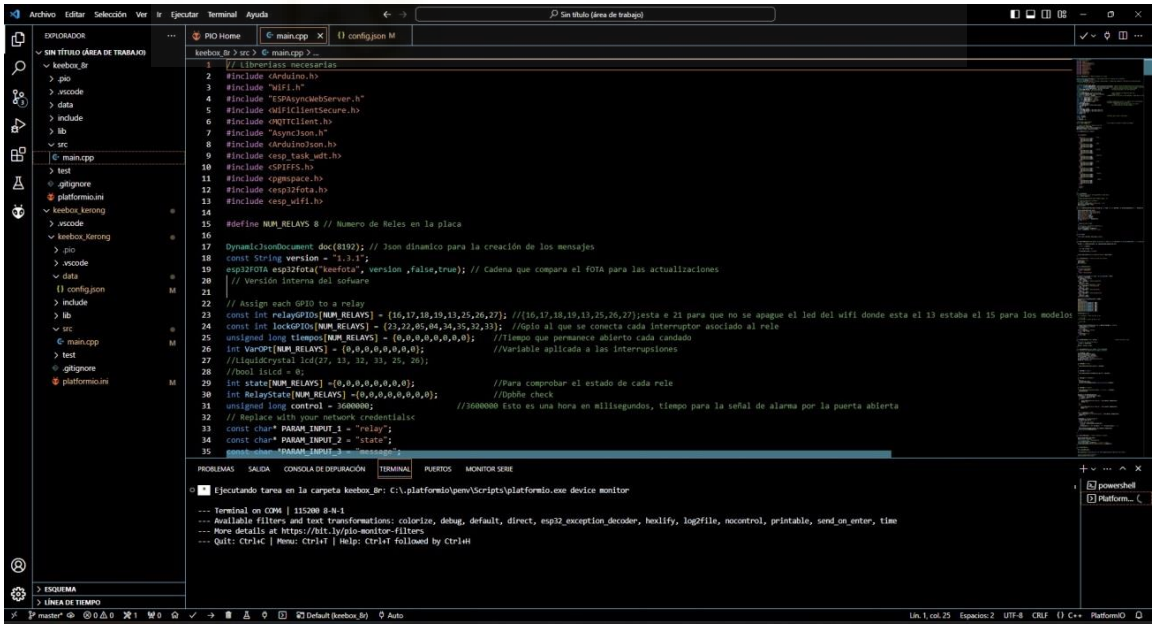


Figura 6.4. Entorno Visual Studio Code

6.3 AWS IOT

AWS IOT ofrece servicios en la nube y asistencia para dispositivos que se pueden utilizar para implementar soluciones de internet de las cosas (IOT por sus siglas en ingles Internet of things). Amazon Web Services (AWS) proporciona una variedad de servicios en la nube para respaldar las aplicaciones basadas en IOT [11].

En el contexto de IOT, AWS IOT ofrece servicios que respaldan los dispositivos que interactúan con el entorno y gestionan los datos que se intercambian entre ellos y AWS IOT.

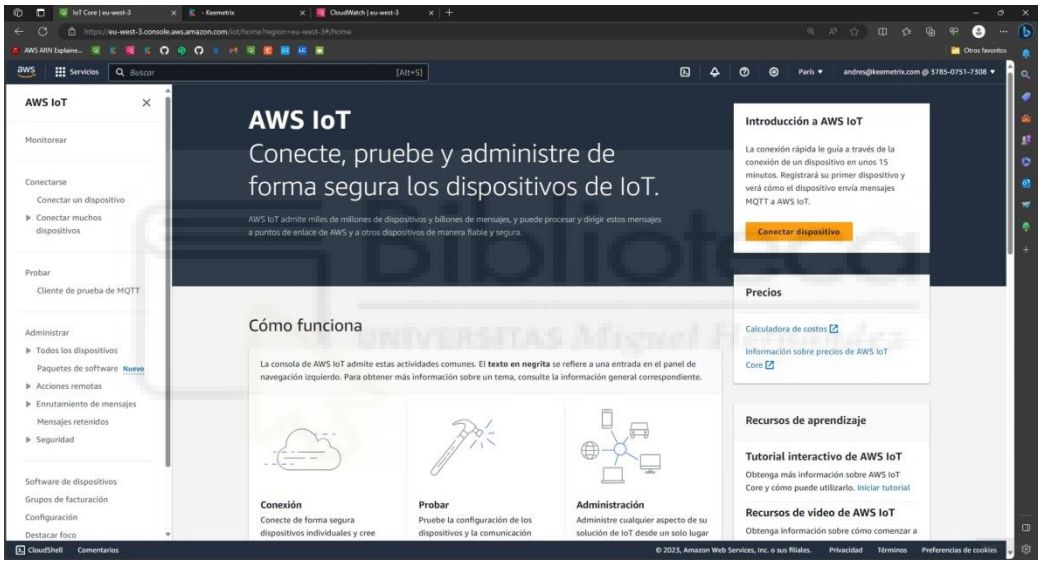


Figura 6.5. Portada AWS IOT

AWS IoT Core ofrece una gama completamente administrada de características de mensajería basadas en MQTT, que pueden ayudarlo a crear arquitecturas de IoT adaptables que sean rentables, fiables y estén optimizadas para la escalabilidad.

Transmite de forma segura mensajes hacia y desde todos los dispositivos y aplicaciones IoT con baja latencia y alto rendimiento.

Con millones de dispositivos y temas simultáneos, AWS IoT Core permite escalar automáticamente su solución conectada para procesar billones de mensajes y, al mismo tiempo, evitar los costos de infraestructura, las tarifas de licencia y otros gastos operativos.

7. ETAPAS DEL DISEÑO

7.1 REQUERIMIENTOS DEL SISTEMA

Para adaptar la PCB original a lo que necesitábamos está debía cumplir con:

1. No tener que introducir fuentes de alimentación externas.
2. Una conexión dedicada para las cerraduras y sus sensores (en caso de tenerlos).
3. No tener que puentear los relés entre ellos para alimentarlos a ellos o a las cerraduras.

Con este primer diseño se solucionaran los puntos 2 y 3 ya que en primera instancia se usaran 2 fuentes diferentes.

7.2 PRIMER DISEÑO

7.2.1 SOLUCIÓN DE PROBLEMAS

Con el primer prototipo se solucionaron los puntos comentados en el apartado anterior de la siguiente forma:

Figura 7.1 Prototipo Nº1

Para no tener que proporcionar una alimentación externa a las cerraduras se optó por modificar los relés originales de 5 V por unos accionados a 12 V, agrupando así, las

alimentaciones. En esta etapa de diseño se propuso mantener dos alimentaciones separadas, la de 3.3 V para el ESP32 y los 12 V ya mencionados para el resto de elementos.

Los pines de salida para los relés de la imagen 6.1 son: el pin común, normalmente abierto (NO por sus siglas en ingles) y normalmente cerrado (NC), con esto solventamos el 2º punto del apartado anterior, lo que no solo ayuda al funcionamiento del sistema sino también a su instalación. En este diseño se agrupan las alimentaciones de los relés quedando solucionado el tercer y último punto del apartado anterior.

7.2.2 DETALLES DEL MODELO

En este primer modelo se elimina el convertidor DC-DC que transforma los 5 V a 3,3 V, esto se debe al pensamiento de que, si en la instalación se usaban dos fuentes separadas una para alimentar toda la línea de chips y otra separada para alimentar la parte de cerraduras y relés, se ahorraría tener que diseñar una parte de transformación. Esto evitaría problemas de sobrecalentamiento además de cumplir directamente la normativa establecida de baja tensión sobre la transformación de los 230 V que llegan a una instalación de vivienda o local.

Los optoacopladores se conectaban directamente al chip ESP32 el cual, al poner el pin correspondiente en alto hacia que el optoacoplador pusiera la base del transistor en ese nivel con lo que este último activa el relé. Este proceso se puede anular ya que conectando el pin del ESP32 a la base del transistor produce el mismo efecto que todo el sistema de los optoacopladores, con lo que se ahorran costes y se simplifica el diseño.

El funcionamiento de la apertura de puertas (accionar el relé) por parte del ESP32 es la siguiente.

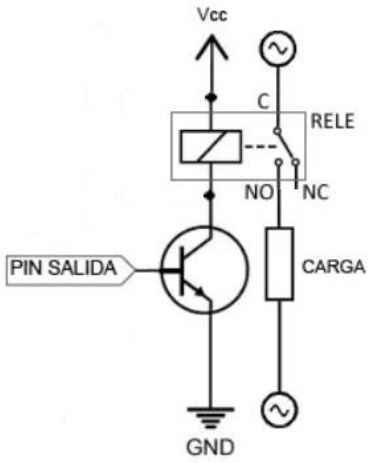


Figura 7.2 Conexión a relé

Cuando el pin de salida (pin conectado a GPIO del ESP32) se pone en alto, 3,3V, el flujo de corriente del transistor pasa hacia GND accionando el relé que cambia de estado, proporcionando la tensión necesaria a la carga.

Se escogieron conectores Cixi Kefa Elec KF2EDGR-5.08 de 2 y 3 pines para la conexión de las cerraduras y de la alimentación dado a su extendido uso entre electricistas e instaladores, ya sea de alarmas u otros técnicos especializados, lo que permitía que, una vez una instalación estuviera completada, en caso de necesidad la sustitución de la electrónica por un modelo nuevo fuera relativamente sencilla. Se eliminaron pines de sobra y solo se dejaron accesibles los necesarios para programar el ESP32, estos son un pin de GND, RX0 y TX0.

Se añadió un pull-up junto a cada relé conectados a 4 GPIOS del SoC esto se implementó dado que las cerraduras incorporan un interruptor interno que nos permite saber si la misma está cerrada o abierta.

El funcionamiento de un pull-up (o un pull- down) es muy sencillo.

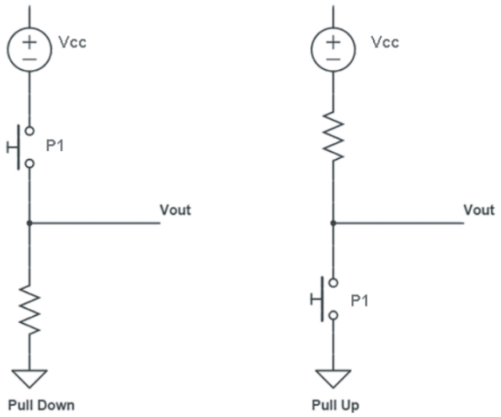


Figura 7.3 Pull-up/Pull-down

Como se puede ver en la figura 7.3 esto sirve para poner en bajo o en alto, según el caso, un pin concreto, en nuestro caso el pull-up pone en bajo el pin del ESP32 cuando el interruptor se cierra, devolviéndonos un 1 o un 0 a nivel de software permitiéndonos conocer su estado.

Se añadió un plano de masa en cada capa del diseño dejando libre la parte donde se encuentra la antena del chip y se añadieron pequeñas vías para conectar mejor las dos capas de GND y se acercó el condensador de protección a la entrada de 3,3V al ESP32.

Señalar que aunque el chip principal es un ESP32 este es un modelo superior que en la placa de DIY MALL que se pasa del ESP32-WROOM-32 al ESP32-WROOM-32D.

7.2.3 PROBLEMAS SURGIDOS

En este prototipo se cometieron una serie de errores que se enumeraran a continuación.

1. No se conectaron los pulsadores de reset y boot a una resistencia cada uno y está a 3.3 V, con lo que la programación del SoC era inviable.

Figura 7.4 Error en pulsadores

2. El pio IO2no se conectó debidamente en la etapa de diseño a GND por lo que esto también impedía la correcta programación de la PCB.

Figura 7.5 Error en Pin IO2

7.3 SEGUNDO DISEÑO

En este nuevo diseño se recolocaron los componentes y se ajustaron los errores del modelo anterior.

Una vez comprobada su funcionabilidad se diseñó un producto más maduro para su prueba sobre el terreno y su instalación en el primero de los centros.

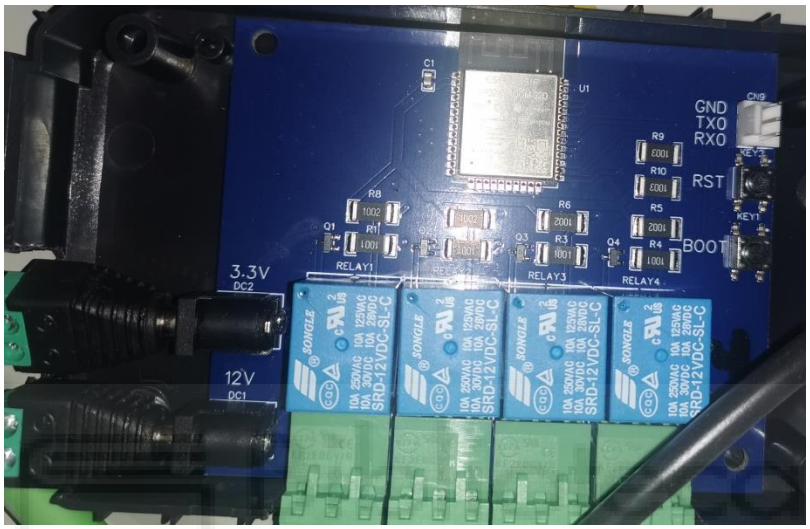


Figura 7.6 Placa funcional

7.3.1 SOLUCIÓN DE PROBLEMAS

Los errores en la figuras 7.4 y 7.5 que se han comentado en el punto 6.2.3 impedían que el ESP32 entrase en modo programación.

Para comprobar que efectivamente rediseñando esta parte se solucionarían los problemas se soldó a la misma placa los componentes necesarios y se puenteó el pin IO2a masa y las resistencias a 3.3 V. Una vez acabado la reestructuración sobre el mismo modelo y ver que la PCB funcionaba correctamente se aplicaron los cambios al diseño.

Estos cambios se pueden apreciar en las mismas figuras 7.4 y 7.5.

Comprobados los cambios se implementaron conectando correctamente el pin superior de cada pulsador a resistencias de 100 KΩ, estas a su vez se conectaban a la línea de 3.3 V, lo que nos permitía poner en alto los pines EN y IO0 al presionarlos.

Junto a la correcta colocación del pin IO2 a masa se solventaron los principales problemas del anterior modelo.

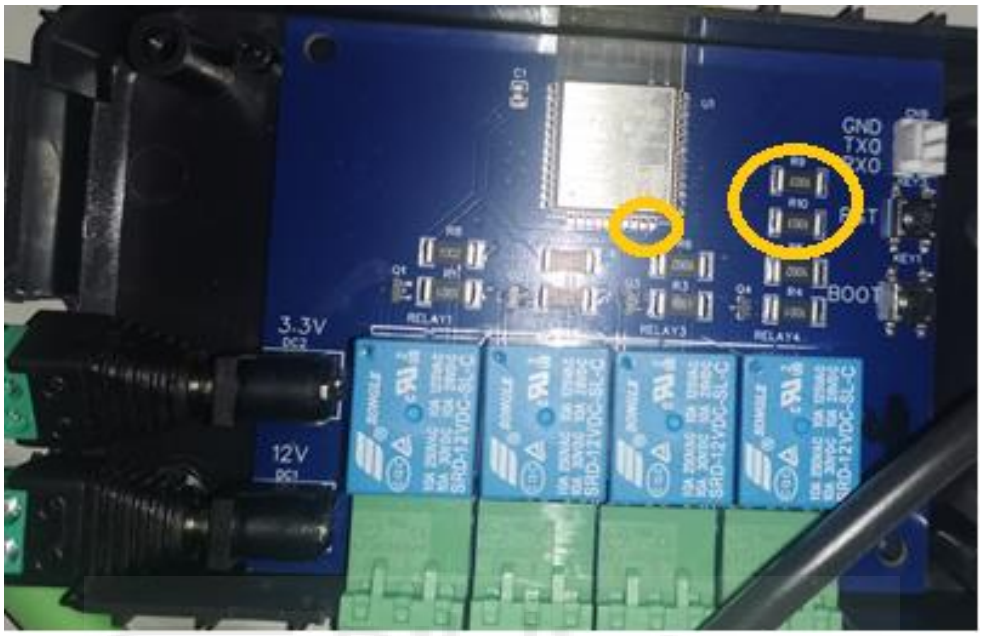


Figura 7.7 solución de errores del apartado 6.2.3

Como podemos apreciar en la figura 7.6 en este diseño se cambió el protector verde por el azul por propósitos estéticos, esto no influía en el funcionamiento o en el precio final, lo que si implicaba es que para pedidos de mayor tamaño no estaría disponible en este nuevo color (azul), ya que la empresa JLPCB solo tiene disponible el verde para pedidos de gran tamaño. Se recolocaron los componentes para mayor comodidad y se sustituyeron los conectores de alimentación por clavijas de tonel. Siendo estas más comunes.

Este modelo demostró ser totalmente funcional y eficiente y en agosto del año 2021 se instalaron en lo que sería el primer centro automatizado por Keemetrix Systems. La instalación de los cables de las cerraduras, los cables de alimentación de 12 y 3,3V, así como las fuentes correspondientes y las placas las llevé yo acabo, ayudándome así a tener mucho más claro el proceso completo.

A día de hoy estas placas han estado funcionando de forma ininterrumpida durante más de dos años, en los que los únicos 3 incidentes fueron por causas ajenas a la compañía.

7.4 TERCER DISEÑO

Teniendo un producto cuya única problemática residía en su instalación, se modeló un nuevo prototipo.

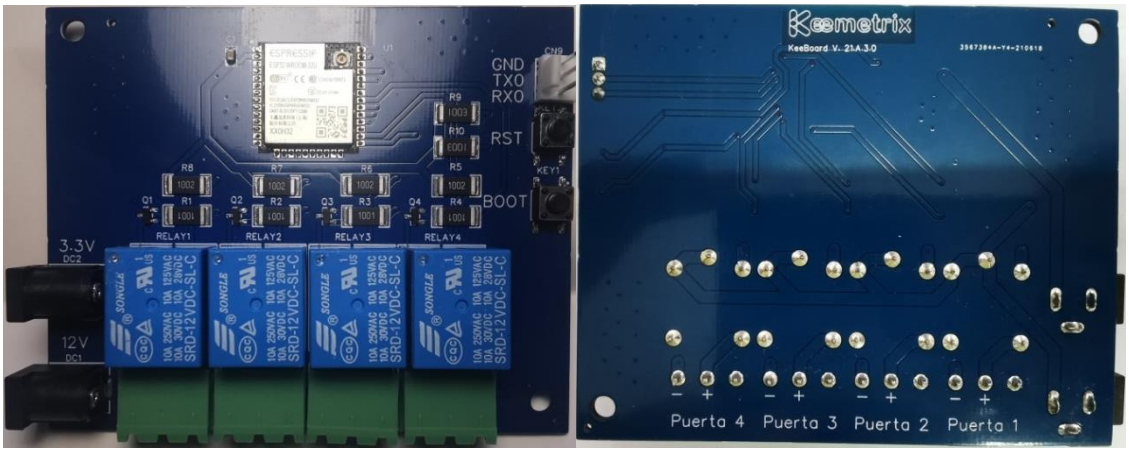


Figura 7.5 Funcional sin antena

7.4.1 AJUSTES AL ÚLTIMO DISEÑO

La única diferencia real entre este modelo y el anterior es que se modificó el SoC ESP32-WROOM-32D por el ESP32-WROOM-32UD, lo que implica la falta de antena impresa e integrada en el chip. Esto se modificó para comprobar si con otro tipo de antena el alcance de la señal Wi-Fi mejoraba o no.

Tras diversas pruebas de alcance y de funcionamiento no se apreciaron mejoras, sin embargo, esto afecto en que el montaje de las mismas se complicaba y al tener que pedir conectores y antenas fuera del proceso de fabricación de las PCBs se incrementaba el precio del producto final.

7.4 CUARTO DISEÑO. ADAPTACIÓN DEL DISEÑO A CUBIERTA PREFABRICADA

Tras la correcta instalación del modelo visto en el punto 6.3 la empresa recibió una oferta para la instalación del sistema en una ciudad catalana. Para esta instalación se quería

llegar con un producto más elaborado y se encargaron cubiertas en las que debía encajar la electrónica.

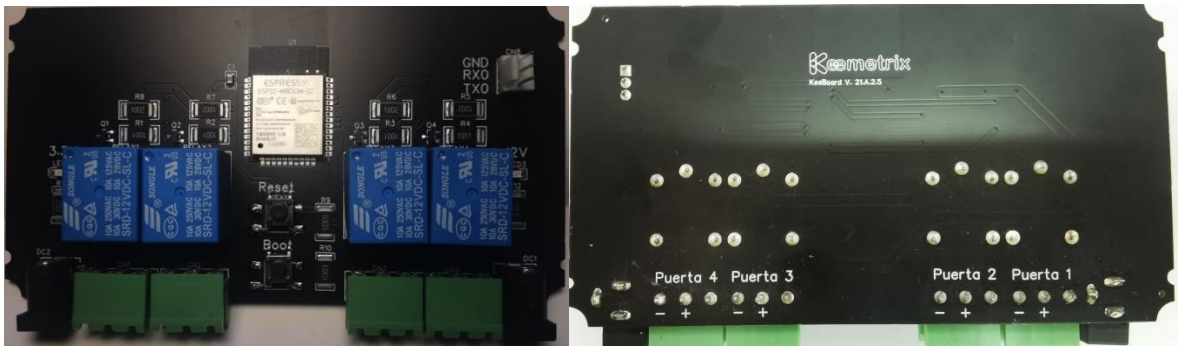


Figura 7.6 Modelo cubierto

7.4.1 REESTRUCTURACIÓN DEL DISEÑO

Se reestructuró el modelo visto en el punto 6.3 adaptándose al ancho y al largo de la cubierta seleccionada. Se dejaron espacios donde para colocar la tornillería que mantendría fijas las PCBs.

7.4.2 PROBLEMAS SURGIDOS

Este modelo presentó un problema de diseño. Se dio el caso de que el modelo de pulsador empleado recibió una actualización en la herramienta easy EDA, cambiando en el esquemático los pines que le afectaban directamente. Para salir del paso y poder seguir con los plazos para la instalación hubo que modificar manualmente la dirección de cada uno desoldando manualmente y volviendo a soldar el componente con un giro de 90º.

Este modelo fue instalado finalmente tras un largo y costoso proceso y estuvieron funcionando durante más de 6 meses, fueron sustituidas por un modelo más moderno que alivió problemáticas que generadas durante la instalación.

7.5 PRUEBAS PARA MEJORA DE FUNCIONALIDADES

Volviendo nuevamente al modelo de la figura 6.5 se añadieron 3 variaciones.

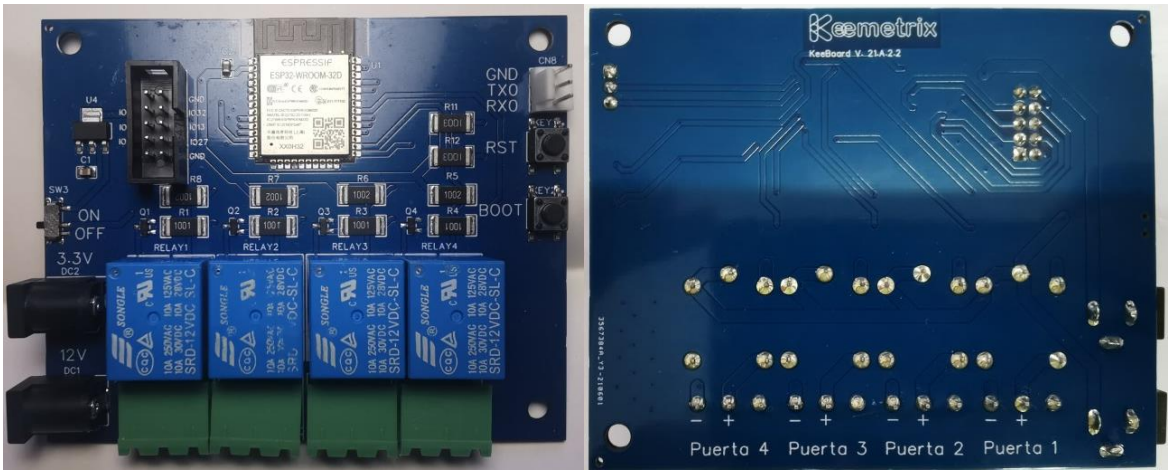


Figura 7.7 Modelo con pantalla

7.5.1 CAMBIOS EN EL DISEÑO

Los cambios más evidentes se dieron al añadido de un conector para emplear una pantalla LCD.

Figura 7.8 CONECTOR PARA PANTALLA LCD

Para facilitar la instalación también se decidió hacer pruebas con un convertidor DC-DC para transformar los 12V de la entrada de los relés y cerraduras a 5V, tensión en la que funcionaba dicha pantalla LCD.

Se instaló también un interruptor para el apagado y encendido del dispositivo.

Figura 7.9 Interruptor y conversor DC-DC

7.5.2 RESULTADOS DE LAS PRUEBAS

La pantalla LCD presentó la problemática de que aunque por separado la pantalla y el sistema funcionaban correctamente al unificar ambos procesos la conectividad de la red entraba en conflicto con la pantalla y viceversa.

La idea se acabó abandonando puesto que la electrónica se coloca fuera de la vista y sus fallos de conexión se reciben a través de los servidores AWS. Como esto encarecía y complicaba el producto se descartó su implementación.

El interruptor funcionó correctamente, este permitía efectuar una instalación del dispositivo con la seguridad de que hasta que no se accionara el componente no llegaría la tensión al SoC principal.

Todos los elementos de la instalación se colocan con los elementos de corriente desactivados, es decir, el magnetotérmico dedicado a la línea esta bajado por seguridad. Esto implicaba que una vez alimentada la línea, hubiera que activar cada elemento de forma individual y dado que los dispositivos tienen que operar de forma continua y autónoma existía la posibilidad de que alguien los desactivara manualmente. Por todo ello fue descartado.

El convertidor funcionó correctamente, no hay comentarios ni negativos ni positivos al respecto, fue una toma de contacto para tener la capacidad de simplificar la instalación.

7.6 QUINTO DISEÑO

Todos los modelos anteriores se habían diseñado teniendo en cuenta que en la instalación habría dos fuentes separadas, una para 3,3V y otra de 12V. Esto implicaba instalar 2 líneas separadas de corriente de alimentación. Esto unido al hecho de que hasta ahora se habían estado utilizando adaptadores a conectores de barril, que servían para conectar todas las placas en serie, implicaba que la conexión con la alimentación fuera frágil y complicada, haciendo que se produjeran intermitencias en las conexiones de la última instalación.

Figura 7.10 Adaptador de conector de barril

Esto se trató de solucionar con el siguiente rediseño.

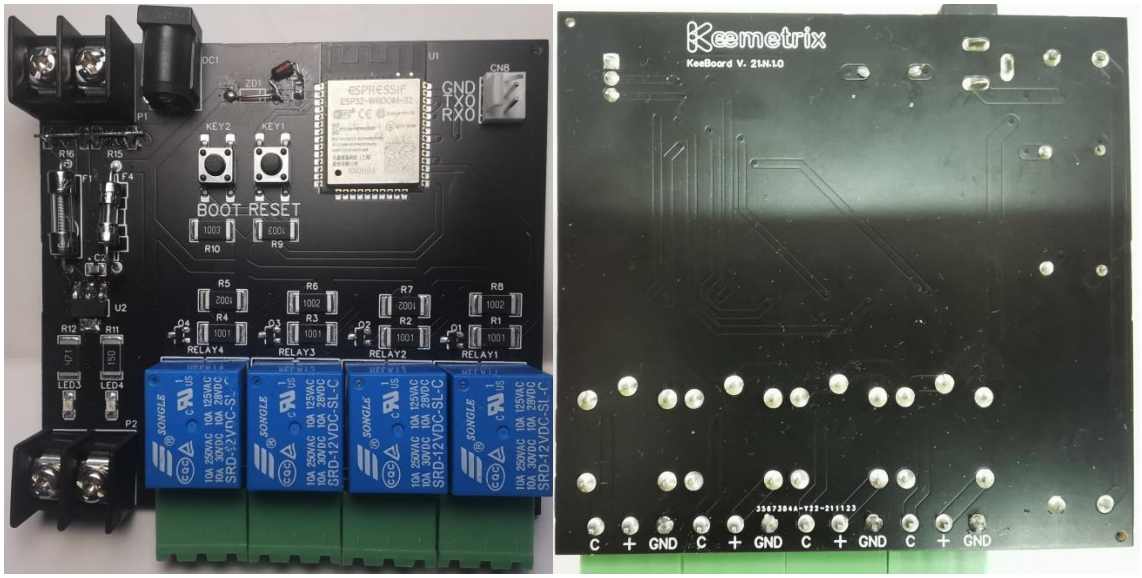


Figura 7.11 Rediseño cuasi integral

7.6.1 CAMBIOS EN EL DISEÑO

Aplicando lo aprendido sobre el convertidor DC-DC en el apartado anterior, se añadió un conversor de 12 V a 3,3 V con lo que pudimos simplificar la instalación y dejar un único conector de barril para la conexión de una alimentación de 12V. Se añadieron dos bloques terminales para que al conectar la electrónica a la instalación estas pudieran conectarse en serie todas a un mismo hilo. Aunque una placa fallase o se estropease, mientras esta se mantuviera conectada no entorpecería el paso de corriente a la siguiente de la línea.

Se añadieron dos fusibles en la placa, uno a la entrada de los 12V de entrada y otro conectado a los 3,3 V que salían del convertidor DC-DC ya mencionado. Por último, en lo que respecta a los cambios en la alimentación, se introdujo un diodo zenner de 3,3V a la entrada de la alimentación del ESP32 para frenar posibles subidas de tensión.

Consideramos añadir una resistencia para limitar la corriente de entrada al dispositivo, debido a la capacidad calculada de la misma y no disponer del valor exacto por el fabricante se decidió poner una pareja del mismo valor en paralelo.

Se diseñó la colocación de dos leds verdes para indicar la correcta conexión de la alimentación de 12V y el correcto funcionamiento del convertidor de 3,3V. Esto servía para

que si se vieran ambos apagados se supiera que habría un problema con la alimentación principal y si solamente estuviera desconectado el de 3,3V saber que el problema procedía del convertidor, haciendo que la detección de problemas fuera más sencilla para el técnico o instalador.

7.6.2 PROBLEMAS SURGIDOS

Como se puede apreciar en la figura 6.12 se retiraron las resistencias que limitaban la corriente de entrada.

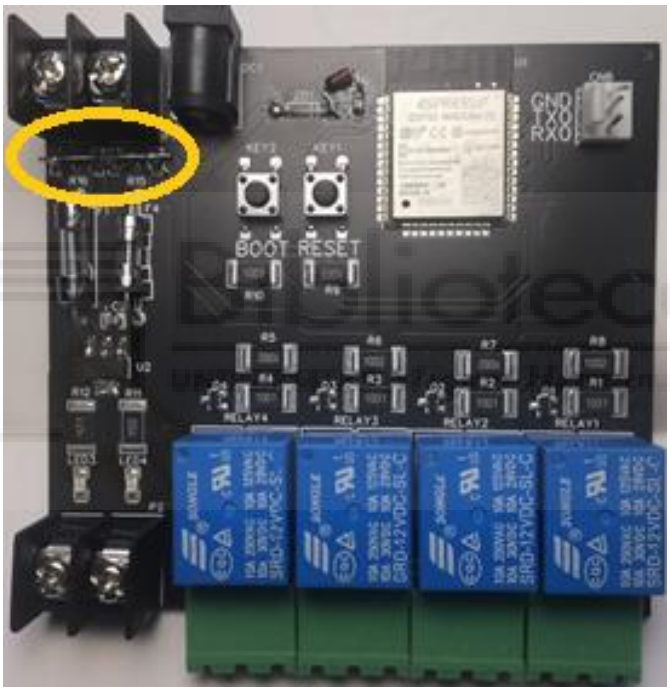


Figura 7.12 Problema con las resistencias

Esto se debe a que las mismas impedían demasiado bien el paso de la corriente y el dispositivo no podía llegar a conectarse.

Se tuvo que replantear la colocación del diodo Zenner, que se puso en serie con el condensador a la entrada del SoC en lugar de en paralelo.

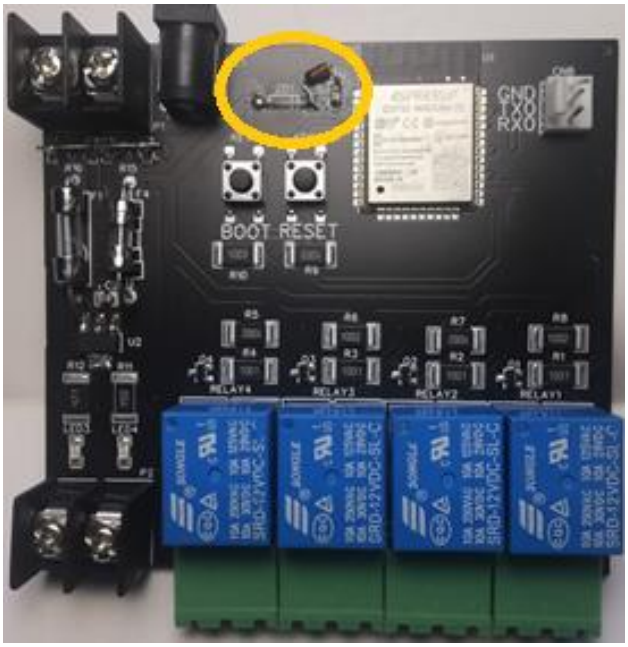


Figura 7.13 Problema con diodo ZENNER

Estos problemas se solucionaron de forma manual.

Se programó el dispositivo y se le hicieron diversas pruebas de funcionamiento, siendo estas derivadas del funcionamiento normal del dispositivo, conexión a la red Wi-Fi y al servidor AWS, probando diferentes ubicaciones y obstáculos. También se comprobó la capacidad para la apertura de las diversas cerraduras, las propias de los trasteros y las diferentes empleadas en las puertas principales.

7.6.3 RESULTADOS DE LAS PRUEBAS

Una vez hechas todas las comprobaciones y anotando los errores principales de diseño las pruebas resultaron en un gran éxito, ya que el conversor DC-DC y los nuevos conectores facilitaban mucho la instalación de numerosos dispositivos y el funcionamiento parecía mantenerse de forma correcta.

7.7 SEXTO DISEÑO

Revisados todos los fallos vistos en el apartado 6.6.2 el rediseño los corrigió y se añadieron ligeros cambios para mejorar el diseño.

Figura 7.14 2º Prototipo funcional

7.7.1 CAMBIOS EN EL DISEÑO

Las resistencias a la entrada se eliminaron definitivamente y se corrigió la colocación del diodo Zenner en la alimentación del ESP32.

Se sustituyeron los fusibles integrados por un soporte para los mismos, esto implica la cómoda sustitución de los mismos en caso de que dejasen de funcionar. Los fusibles ahora se pedirían a la tienda por separado.

El modelo demostró un alto nivel de funcionabilidad y fue empleado para sustituir elementos averiados en otras instalaciones y como dispositivo principal en nuevas instalaciones.

7.8 MINIATURIZACIÓN DEL SITEMA

Dado que el último modelo fue considerado un éxito, ya que no solo ayudaba a reducir el coste y complejidad de la instalación, reduciendo las líneas de alimentación necesarias a la mitad, si no que añadía elementos de seguridad a la electrónica y al no tener

que usar 2 fuentes de alimentación abarataba los costes de la instalación total, se usó como nueva base para los futuros rediseños abandonando así el prototipo del punto 6.4.

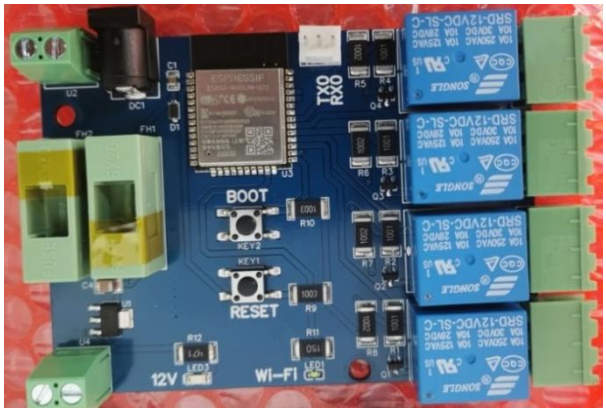


Figura 7.15 Circuito "miniaturizado"

7.8.1 CAMBIOS EN EL DISEÑO

Se sustituyó el diodo Zenner por uno de menor tamaño de tipo SMD, se modificó el cartucho contenedor de los fusibles haciéndolos menos aparatosos y se cambiaron los bloques terminales de alimentación por otros de menor tamaño y considerados más elegantes estéticamente hablando, este cambio no afecta a su funcionabilidad.

También se eliminó el led de 3,3 V y se colocó un led azul conectado al GPIO15 del Esp32, el cual se encendía una vez el código detectaba que estaba conectada correctamente a la red Wi-Fi. Esto último se pensó para ayudar a resolver problemas durante la instalación o en una reparación de forma más rápida y eficiente.

7.8.2 REAJUSTE DE EFICIENCIA

Más adelante se observó que el segundo fusible colocado no era relevante y se eliminó, reduciendo ligeramente el tamaño de la PCB y el coste total.

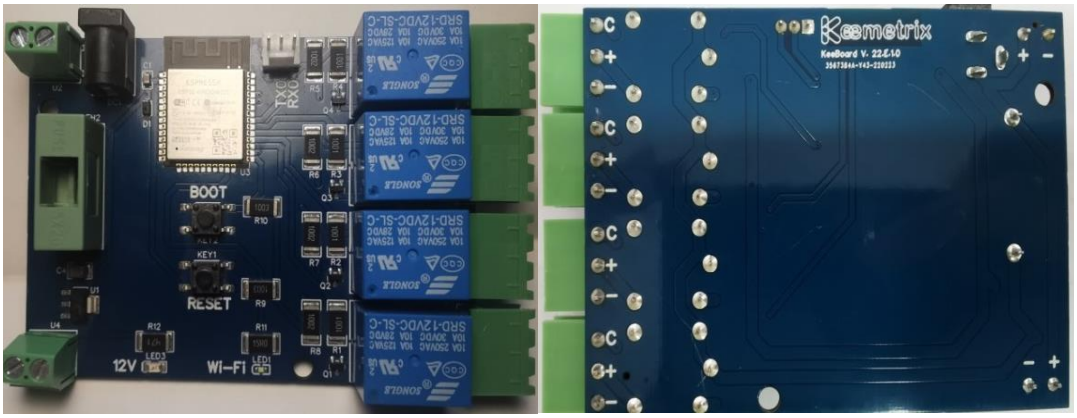


Figura 7.16 Circuito "miniaturizado" simplificado

7.9 SEPTIMO DISEÑO. REDUCCIÓN DEL NÚMERO DE DISPOSITIVOS

Este último modelo resultó ser completamente funcional, eficiente y cómodo de instalar gracias a sus conectores y reducido tamaño.

El problema que se plateo a continuación fue debido a que según se incrementaba el tamaño de los centros automatizados se aumentó el número de dispositivos conectados a la red del mismo y esto empezó a generar desconexiones y cuellos de botella.

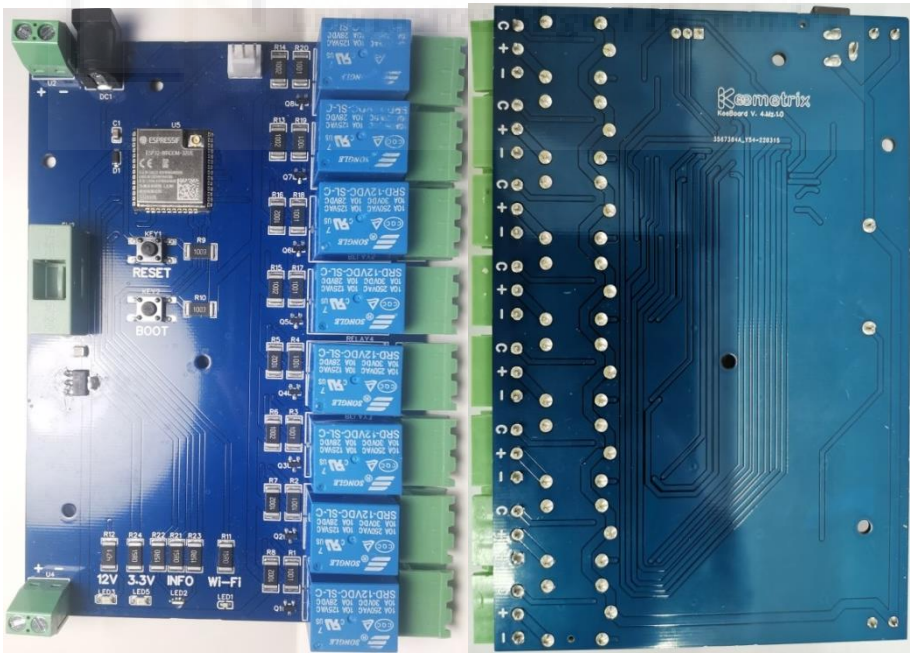


Figura 7.13 Modelo 8 relés

7.9.1 CAMBIOS EN EL DISEÑO

Para reducir el número total de elementos conectados se estudiaron todos los pines disponibles de nuestro SoC ESP32 y se diseñó este último modelo, añadiendo 4 relés más reduciendo a la mitad los elementos conectados necesarios.

En este modelo también se volvió a introducir el led de alimentación de 3,3V, se mantuvo el led "Wi-Fi" (encargado de informar cuando estaba conectado a esta red o no) y un led RGB que se implementó para proporcionar información adicional sobre errores o lo que le ocurría al procesador durante su funcionamiento, lo que mejoró nuestro conocimiento al respecto durante las instalaciones.

7.9.2 PROBLEMAS SURGIDOS

El modelo funcionó correctamente salvo por el pin GPIO21 que controlaba la apertura del quinto relé, el primero de los recién implementados, este no respondía a los intentos de funcionamiento a pesar de que la hoja de datos del ESP32 y el resto de información conocida indicaban que era utilizable para dicha labor.

Se retornó a la idea de añadir antenas externas a este modelo, por un intento de ampliar su rango efectivo de conexión, durante el periodo en que se emplearon existieron a la vez ambos modelos, con y sin antena impresa.

7.9.3 RESULTADOS Y OBSERVACIONES

Con los problemas relacionados con el relé 5 y su pin de control GPIO21 este modelo de "8" relés pasó a ser de 7 ya que el quinto relé no era funcional al margen del tipo de programación intentada. Se comenzó el estudio sobre que pin podría reemplazarlo.

Por otra parte ya que la eficiencia de las placas con antena integrada y sin esta eran equivalentes se volvió a descartar, esta vez definitivamente, la idea de añadir las antenas de forma externa a la fabricación.

Estas PCB aún con sus fallos se emplearon en diversas instalaciones, teniendo en cuenta que la funcionalidad real era de 7 cerraduras por dispositivo.

Aún siguen operativas hasta la fecha.

7.10 APROXIMACIÓN AL MODELO DEFINITIVO

Este último diseño aunque funcional, tuvo que ser modificado posteriormente debido a que un cambio en las especificaciones del convertidor DC-DC de 12V a 3,3V que hacía que no soportasen la potencia empleada hasta la fecha.

7.10.1 EXPLICACIÓN DE LA PROBLEMÁTICA

El cambio en las especificaciones, las cuales modificaban la potencia máxima disipada por el componente, provocó que en la siguiente instalación nada más conectarse los dispositivos estos no disipasen correctamente la potencia quemando así sus convertidores y destruyendo a su vez el chip principal de los mismos, haciendo imposible su reparación.

Esto obligo a una retirada forzada del nuevo material.

7.10.2 SOLUCIÓN Y REDISEÑO

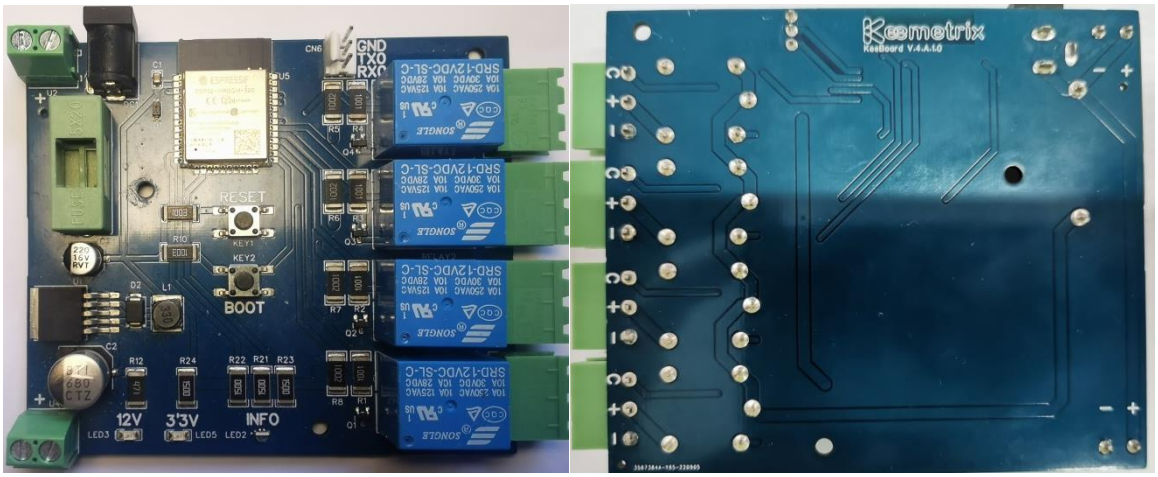


Figura 7.14 Modelo 4 relés definitivo

El cambio más evidente, y necesario, es el cambio del conversor de la alimentación, este conversor cambia a 3,3 V tensiones desde 4 V hasta 24 V, haciendo que los problemas de disipación de potencia desaparezcan y que las subidas de tensión dejarán de ser un problema.

Entré otros cambios se observa la eliminación del pin "Wi-Fi" (GPIO15) ya que el led RGB instalado se emplea para mostrar los distintos estados del dispositivo, haciendo redundante la información proporcionada por dicho led.

Más adelante se modificó el conector del fusible por otro más común para solventar problemas de stock del mismo.

El SoC ESP32 que empezamos utilizando fue el ESP32-WROOM, que fue modificado por el ESP32-WROOM-32D en el punto 7.2 y para ajustarnos a los últimos modelos fabricados en masa por Espresif sustituimos este último por el modelo ESP32-WROOM-32E, este modelo como sus versiones anteriores comparte el mismo PINOUT (configuración de pines del microprocesador) por lo que no modifica el diseño pero añade compatibilidades para diferentes versiones. También nos asegura que no es descontinuado.

7.10.3 RESULTADOS Y OBSERVACIONES

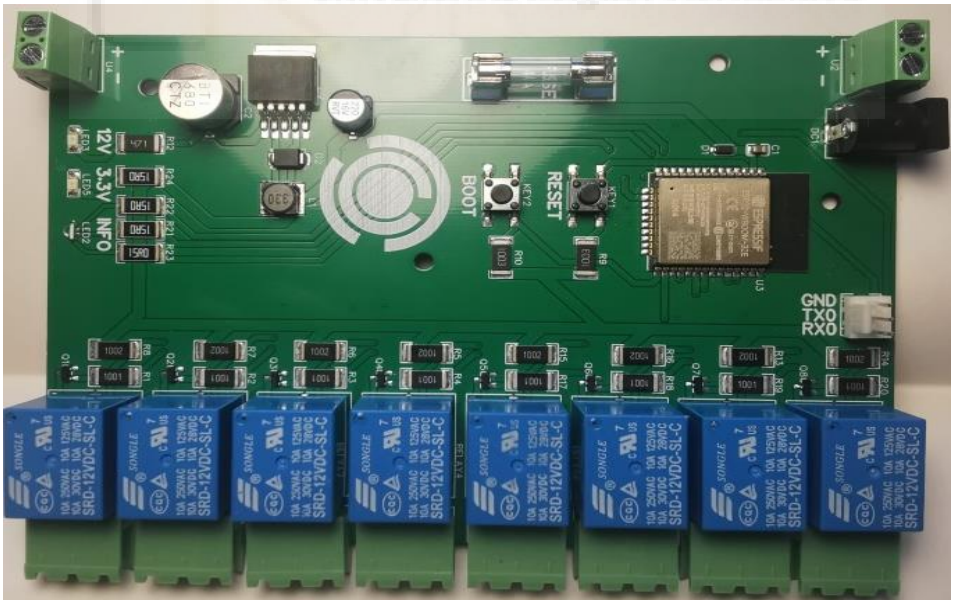


Figura 7.15 Modelo 8 relés casi definitivo

En el modelo de 8 relés de la figura 7.15 ya se puede apreciar el nuevo conector para el fusible y la eliminación del led que se consideró innecesario.

También se intercambió el pin GPIO21 por el GPIO15, dado que este activaba correctamente el led Wi-Fi se sobreentendió que también podía activar correctamente el pin del relé 5.

Pudimos apreciar que gracias a la interconexión del nuevo conversor que el circuito demostraba una nueva capacidad de resistencia a malas conexiones para con la cerradura. Anteriormente si el instalador conectaba equivocadamente el pin positivo de la cerradura, haciendo que los 12 V proporcionados por la fuente en el momento de la apertura se derivasen a la placa por un pin incorrecto, solía suponer el fin de la misma. Ahora tras un malfuncionamiento la electrónica volvía a conectarse a la red tras un fallo imprevisto, sobreviviendo a la primera apertura de esta incorrecta conexión, dando tiempo al técnico a modificar su instalación.

7.10.4 NUEVAS PROBLEMATICAS

En este modelo, completamente funcional y que a día de hoy sigue instalado en algún centro, se descubrió que el pin IO15, era de hecho un Straping pin, es decir, en el momento en el que la placa se reinicia, ya sea a propósito como por un mal funcionamiento, este pin se pone en alto, haciendo que la puerta asociada se abra generando un problema de seguridad para el trastero alquilado.

7.10.5 SOLUCION DE DISEÑO PARA EL ÚLTIMO PROBLEMA

El pin GPIO15 se intercambió por el GPIO13 asociado al led RGB, así todos los stapping pins quedaban asociados al led RGB haciendo que su funcionamiento durante el reseteo de la electrónica no genere ninguna clase de problema.

7.14 EXPLICACIÓN DETALLADA SOBRE EL MODELO ADAPTABLE

El análisis detallado de todos los componentes utilizados, del esquemático y del diseño del PCB se harán sobre el modelo adaptable por ser el más completo y definitivo.

Todos los datos técnicos y físicos de los componentes se encuentran en sus respectivos enlaces en el anexo I.

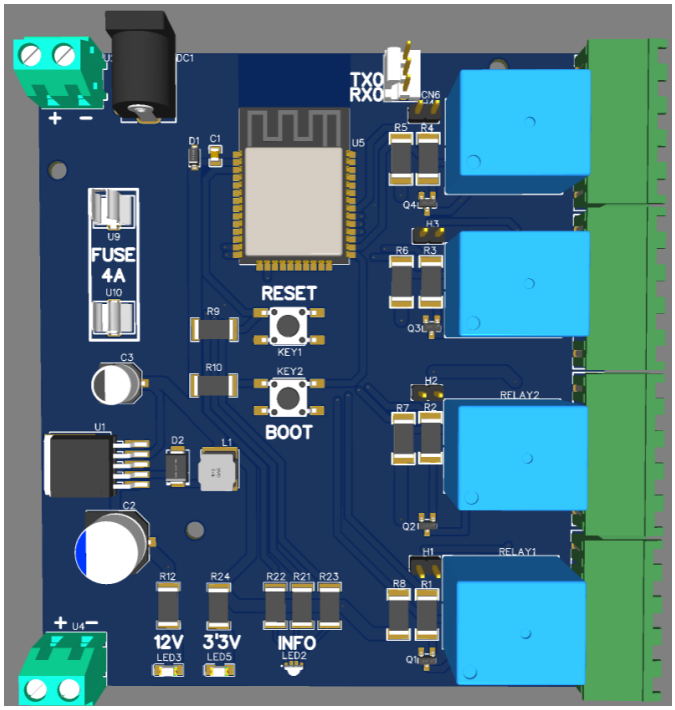


Figura 7.16 Resultado final del diseño

Se irán explicando los distintos pines empleados así como los componentes acoplados a los mismos usando como referencia el esquemático de la figura 7.17.

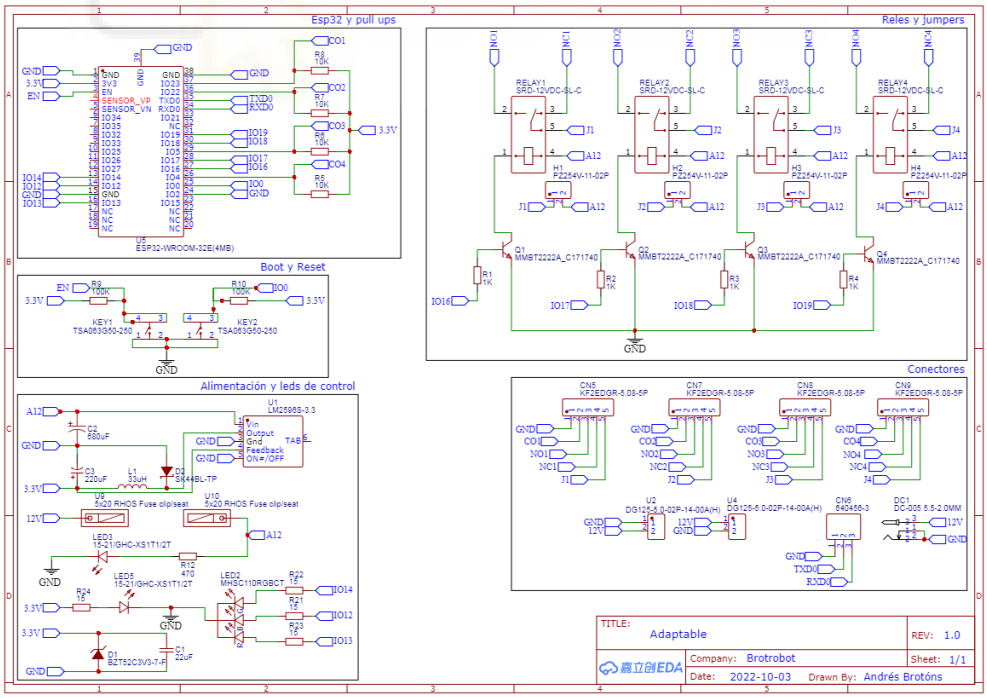


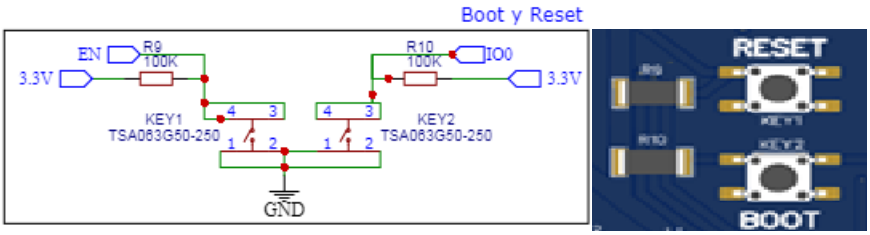
Figura 7.17 Esquemático del diseño adaptable

7.14.1 CONEXIONES CON ESP32

Este diseño está centrado en el *ESP32-WROOM-32E(4MB)* cerebro y parte principal del proyecto. Este módulo ampliamente utilizado se caracteriza por ser un dispositivo programable que lleva integrados las tecnologías Wi-Fi y bluetooth, estas tecnologías no pueden emplearse de forma paralela ya que comparten la salida de la antena y al propagarse a la misma frecuencia (2’4 GHz) hace inviable este método. Aunque pueden usarse de forma alterna desconectando uno u otro según convenga. Para este proyecto nos centraremos solo en la tecnología Wi-Fi como ya hemos comentado antes.

El ESP32 alcanzo el pico de usuarios debido a la capacidad de emplear el framework de Arduino en lugar del propio del desarrollador (Espresif) lo que permitió a la comunidad DIY documentar librerías y ejemplos de código de forma libre y eficiente. Nosotros también emplearemos este framework.

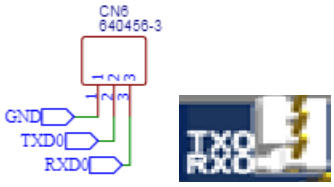
Empezaremos por explicar las conexiones que hacen posible el funcionamiento básico del circuito, cada pin GND fue conectado directamente a la red del mismo nombre, así como el pin GPIO02 el cual tiene que estar a nivel bajo para que el ESP32 pueda entrar en modo programación. Para entrar a este modo se necesita pulsar una combinación concreta de los botones BOOT y RESET, para el cual empleamos el componente *TSA063G50-250* al que conectamos a uno de sus extremos al pin GPIO01 y ENABLE respectivamente, y en el otro extremo del pulsador una resistencia de 100KΩ (*UNI-ROYAL(Uniroyal Elec) 25121WF1003T4*), conectada a su vez a 3,3 V, para poner en alto los pines correspondientes al pulsar.



7.18 Pulsadores

Ya que hemos hablado de que es necesario para poner el chip en modo programación hablaremos ahora de los pines destinados para este propósito. Los pines TXD0 y RXD0 (GPIO01 y GPIO03) se conectan a un conector *TE Conectivity 640456-3* al

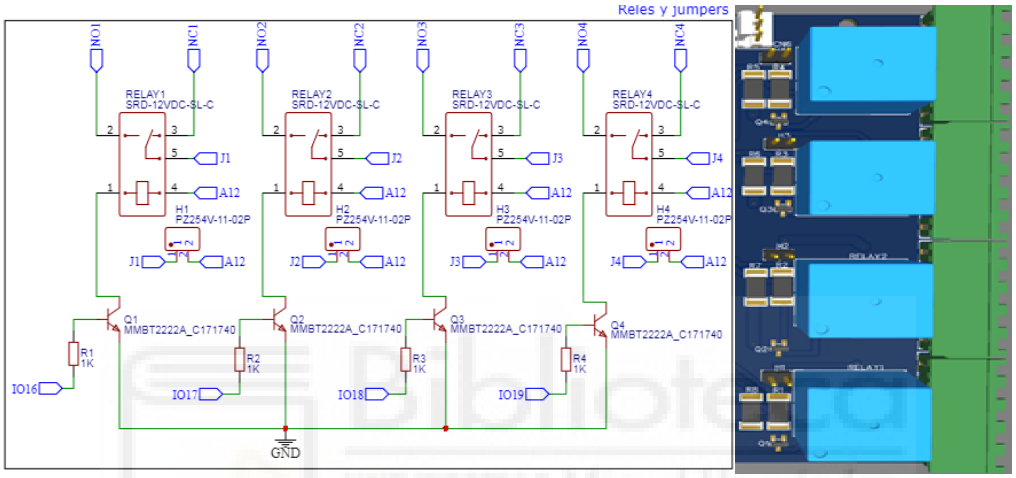
igual que un pin a GND, esto servirá para conectarse atreves de un cable a un convertidor TTL CP2102, el cual se distribuye libremente, no se añadió al diseño para que a la hora de un intento de jaqueo no hubiera una entrada USB evidente a la placa.



7.19 Conector para programador

Continuamos ahora con el objetivo principal del proyecto, la capacidad de suministrar corriente para el funcionamiento de las cerraduras, esto se hizo mediante un sistema sencillo de un transistor BJT *Shikues MMBT2222A* cuya base está conectada a una resistencia *UNI-ROYAL(Uniroyal Elec)25121WF1001T4E* de 1 kΩ que conecta con los GPIO16, 17, 18 y 19, uno para cada relé implementado, siendo estos los relés 1, 2, 3 y 4 respectivamente. El emisor de los transistores está conectado directamente a GND y el colector a uno de los pines de la bobina de los relés *Ningbo Songle Relay SRD-12VDC-SL-C* la cual en su otro extremo está conectada a 12 V, así cuando el ESP32 ponga el pin conectado a la base en alto permite el flujo de corriente hacia GND haciendo así que el relé cambie de estado normalmente cerrado (NC) a normalmente abierto (NO por sus siglas en inglés Normaly Open). El pin común del relé (llamados J1, 2, 3 y 4 en el esquemático) va conectado a un pin *XCFN PZ254V-11-02P* el cual a su vez va conectado a 12 V, esto nos permite la configuración manual de la entrada del dispositivo, es decir, si queremos que actúe como un relé libre de tensión dejaremos los pines J separados, que queremos que el pin NC y NO pasen directamente 12 V sin tener una fuente externa que los entregue, colocaremos un jumper entre ellos uniendo los 12 V con dichas salidas. El relé se seleccionó de 12 V por que al tener que alimentar las placas con 12 V si o si para poder abrir las cerraduras estándar para las que está destinado este dispositivo así como una amplia gama de cerraduras para puertas más comerciales, se consideró poco práctico que el relé se activara a una tensión diferente que la que se le iba a suministrar directamente ya que ello habría requerido una conversión propia de la tensión entregada a la placa.

Los GPIO23, 22, 05 y 04 se conectan tanto a una resistencia de 10 kΩ *UNI-ROYAL(Uniroyal Elec)25121WF1002T4E* como a los pines CO1, 2, 3, 4 del conector *Cixi Kefa Elec KF2EDGR-5.08-5P*, esta configuración es la de un Pull-Up que nos permite conocer el estado de un pulsador/sensor conectado al mismo, a este mismo conector se conectan GND y los pines NO, NC y común (J1, 2, 3 y 4 en el esquemático) que sirve para que puedan conectarse las cerraduras con sensor, cerraduras sin sensor, el sensor por sí mismo, o una cerradura que no necesite tensión o una diferente a la de 12 V.



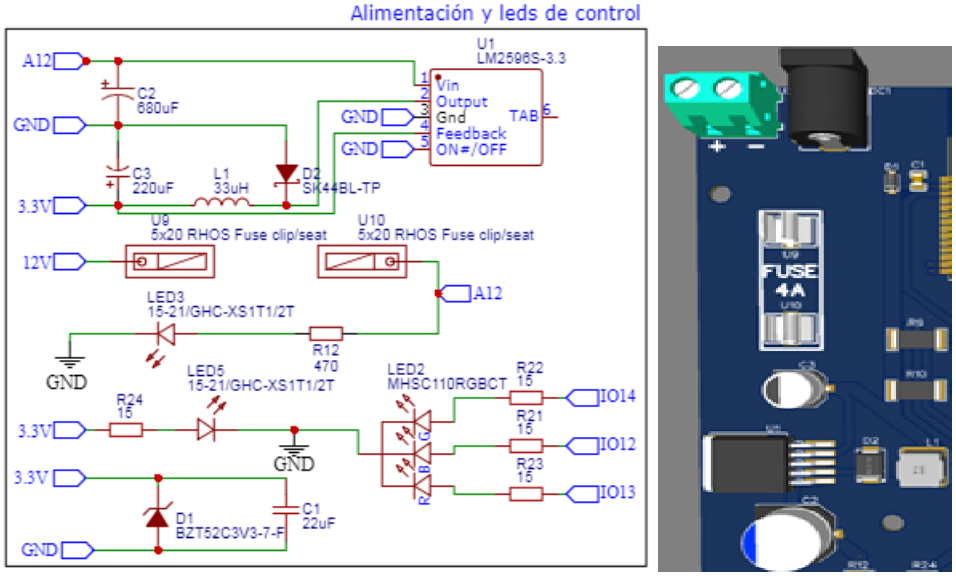
7.20 Relés

Ahora solo queda por explicar lo que permite que todo funcione, la alimentación. La alimentación de entrada es de 12 V en continua, los conectores que se han implementado en la placa para su recepción son un conector de barril *BOOMELE(Boom Precision Elec) DC-005 2.0* por si empleáramos una fuente de alimentación individual para cada placa y un par de bloques terminales *DEGSON DG125-5.0-02P-14-00A(H)* por si usáramos una fuente de mayor potencia para conectar todos los elementos de una instalación en serie. Estos tres componentes están interconectados entre sí, es decir, no importa en cual conectemos la alimentación, siempre que estén conectados correctamente todo funcionará como debe. Tras esta primera entrada encontramos dos sujeta fusibles *Xucheng Elec C3130* que están pensados para sujetar un fusible de 4 A, esto es así ya que aunque el ESP32 consume 3,3 V, 0’5 mA los relés y las cerraduras se alimentan a 12 V y en un pico de uso (abrir los 4 relés a la vez y sus correspondientes cerraduras) se pueden observar picos de hasta 4 A, más que esto suele deberse a un fallo en alguna de las cerraduras, por lo que el fusible se fundirá y protegerá al sistema de esta demanda

inesperada. Se colocaron dos de ellos a una distancia acorde al tamaño del fusible empleado.

Tras el fusible encontraremos un regulador de voltaje paso-bajo *UMW (Youtai Semiconducto Co., Ltd.) LM2596S-3.3* que nos convertirá los 12 V en 3,3 V para el funcionamiento del ESP32, este componente soporta tensiones de hasta 24 V por lo que es muy robusto y por sí mismo protege de sobretensiones que pudieran producir las fuentes de alimentación sobre el circuito. La configuración para su correcto funcionamiento se compone de un condensador electrolítico de 680µF (*Rubycon 16TZV680M10X10.5*, conectado entre Vin y GND, un condensador electrolítico de 220 µF *KNSCHA RVT220UF16V67RV0015* entre la red 3,3 V y GND, un diodo *MCC(Micro Commercial Components) SK44BL-TP* (la hoja de datos del componente se puede encontrar en el anexo 15) entre la salida del regulador y GND y una bobina *Sunltech Tech SLs5D28S330MTT* de 33 µH entre la misma salida del regulador y la red 3,3 V y el pin Fedback del regulador se conecta a la misma red 3,3 V. Esta red de 3,3 V se llama así porque todos los elementos que se conectan a dicha tensión se conectan a partir de este punto y no directamente a la salida del regulador.

A la entrada de tensión del ESP32 se añaden en paralelo un diodo zenner *Diodes Incorporated BZT52C3V3-7-F*, que se bloqueará a 3,3 V en caso de subida de tensión tras el regulador y un condensador cerámico de 22 µF *CCTC TCC0805X5R226M160FT* que filtrará los picos de corriente en caso de oscilamiento.



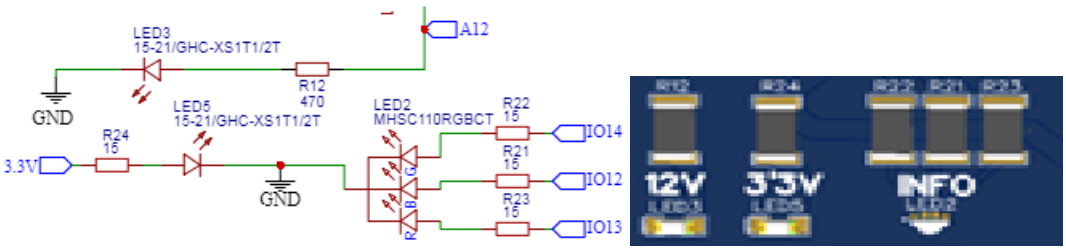
7.21 Alimentación

Los únicos componentes restantes del diseño por explicar son los 3 leds implementados que aportan información relevante para su instalación y para su mantenimiento.

Una resistencia de 470 Ω *UNI-ROYAL(Uniroyal Elec)25121WJ0471T4E* conectado a 12 V provenientes directamente del fusible, y su otro extremo conectado a un led verde *Everligth Elec 15-21/GHC-XS1T1/2T* que se conecta a GND. Este led permitirá saber si la alimentación principal llega correctamente al dispositivo, en caso de no estar encendido se comprobarían o el fusible o la alimentación general.

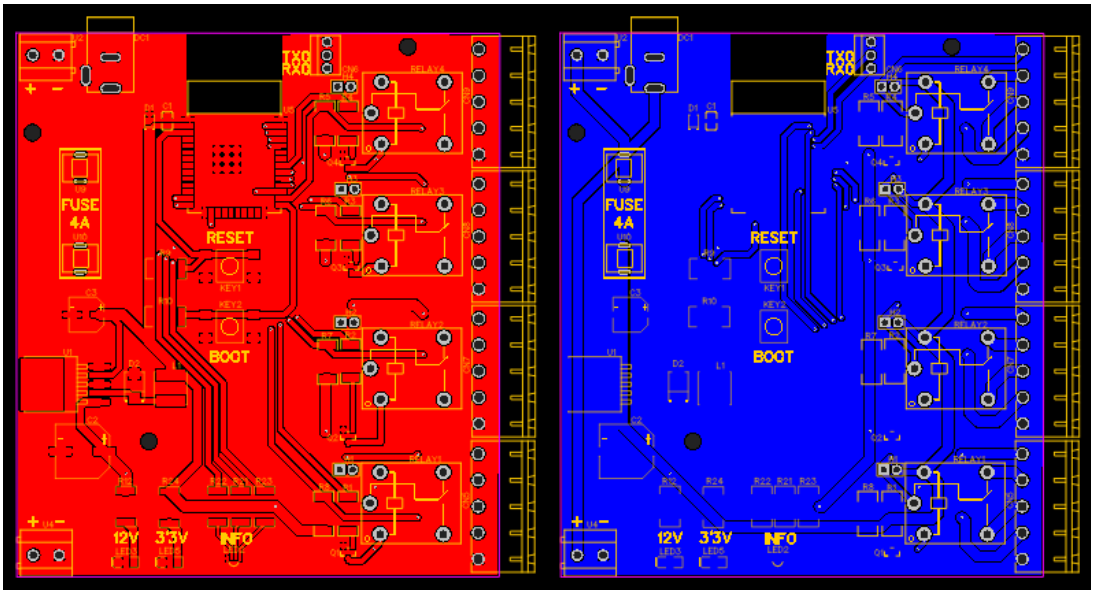
Una resistencia de 15 Ω *LIZ Elec CR2512J10150G* conectada a la red 3,3 V justo después del sistema regulador, y en su otro extremo otro led verde *Everligth Elec 15-21/GHC-XS1T1/2T*, conectado este a GND. Este led nos permitirá saber si algo le ha ocurrido al regulador de 3,3 V.

Por último conectaremos un led RGB *MEIHUA MHSC110RGBCT* a GND y sus pines R (rojo), G (verde) y B (azul) irán conectados a tres resistencias de 15 Ω *LIZ Elec CR2512J10150G* y a los pines GPIO del ESP32 13, 14 y 12 respectivamente. Con este led podremos programar distintas salidas lumínicas para informar del estado de la placa así como informar de errores conocidos ya sea una desconexión del servidor o de la propia red Wi-Fi.



7.22 Leds informativos

7.14.2 LAYOUT



Figuras 7.23 Diseño completo del PCB

El diseño físico de la placa se diseñó tratando de ser lo más cómodo posible de manipular, teniendo las entradas de tensión arriba (formando una línea horizontal entre los dos bloques terminales) quedarían los conectores de los relés enfrente del técnico, viendo así fácilmente el estado de los leds, si los jumpers están colocados o no y como están conectadas las cerraduras a los relés.

Se trató de colocar los condensadores lo más cerca posible de las entradas y salidas de tensión y se agruparon los elementos por conjuntos claramente diferenciados.

Estos vendrían a ser:

- Los relés con sus transistores y resistencias junto a la resistencia correspondiente a cada Pull-Up de su sensor (conjunto de cerradura).
- Los pulsadores al lado de sus propias resistencias (conjunto de pulsadores).
- La alimentación que esta toda alineada en un lateral de la PCB para ayudar con su instalación y para separar en la medida de lo posible el ruido que esta generase del chip principal (Conjunto de entrada).

- Los leds están debidamente alineados en la parte inferior del PCB para su fácil lectura e identificación, además de estar cerca de sus propias resistencias (conjunto informativo).
- El conector de 3 pines destinado a la programación del ESP32 está lo más cerca posible del mismo para evitar en lo posible la entrada de ruido o distorsiones.

Comentar que para las líneas de alimentación se usaron pistas de hasta 2mm para favorecer el flujo de corriente y que el resto de son de 1mm que es el ancho del pad del ESP32, el cual es óptimo para el control de sus 3,3 V.

Debido al tamaño de sus pines la conexión entre las resistencias del led RGB y el mismo son de 0,5 mm.

En todo momento se evitaron codos de 90º y se cubrió todo el circuito con planos de masa por ambas caras del diseño excepto en la zona de la antena del chip por motivos de transmisión de señal.

8. GENERACIÓN DE UN OBJETO EN AWS IOT

Entendemos por objeto en AWS el elemento (o tipos de elementos) que se asociaran al dispositivo físico para identificarlo en los servidores de Amazon y conocer sus funciones e intercambios de datos con los mismos.

8.1 Creación del objeto:

Antes de programar el dispositivo es necesaria la creación del mismo en el servidor AWS así como sus certificados y sus políticas para que una vez conectado al servidor este sea reconocido y la comunicación sea segura. Para esto seguiremos los siguientes pasos:

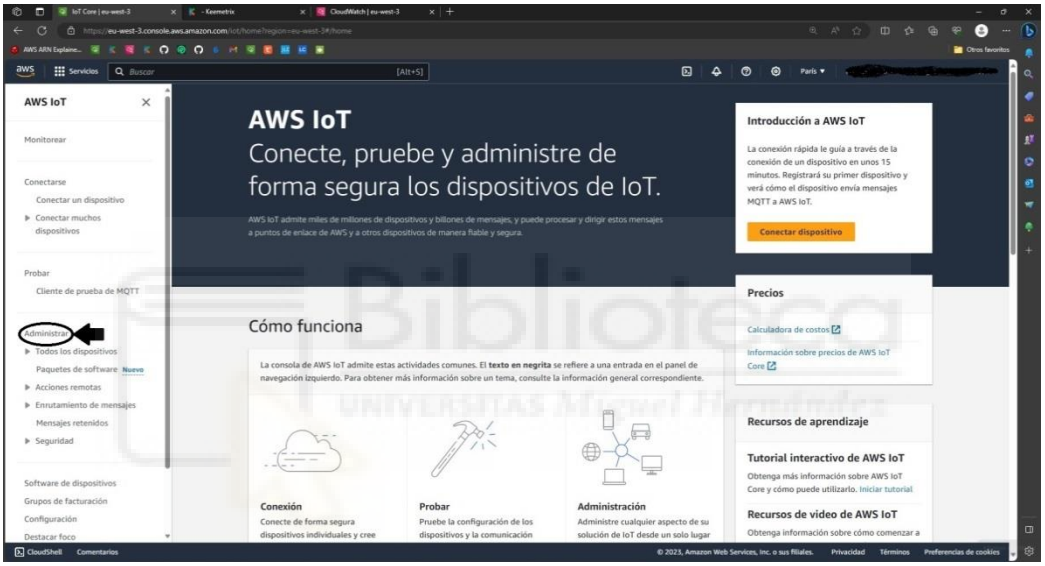


Figura 8.1 Inicio de AWS IOT

Comenzamos por entrar en AWS IOT una vez aquí iremos a Administrar > Todos los dispositivos > Objetos como se puede ver en la figura 8.2.

Figura 8.2 Administrar

Nos aparece entonces en pantalla el menú de los objetos, hacemos click en **crear objetos.**

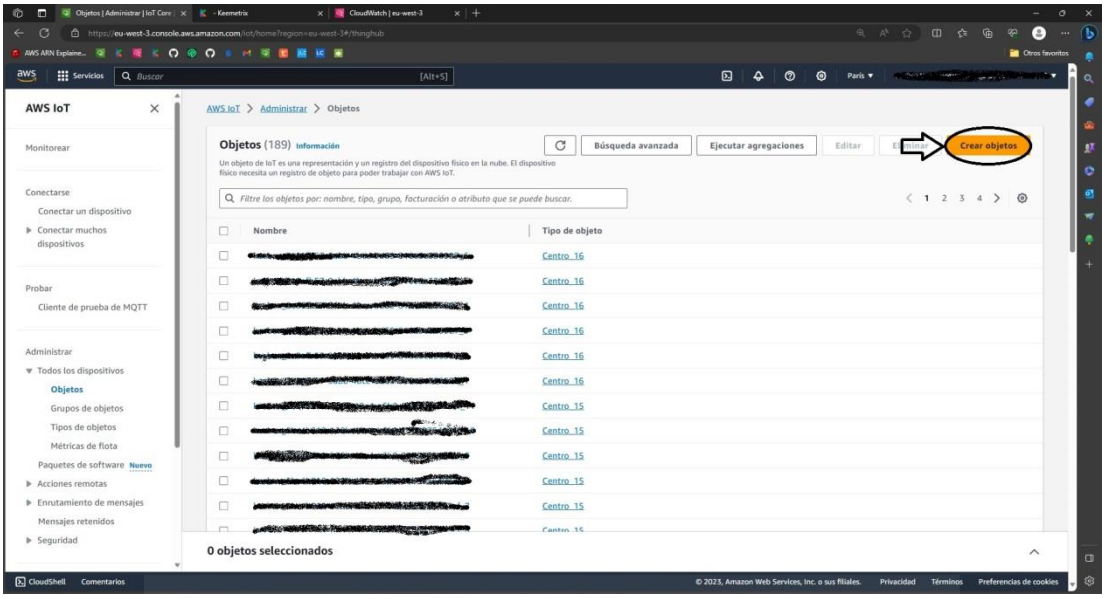


Figura 8.3 Índice de objetos de AWS IOT

Seleccionamos Crear un único objeto y presionamos siguiente.

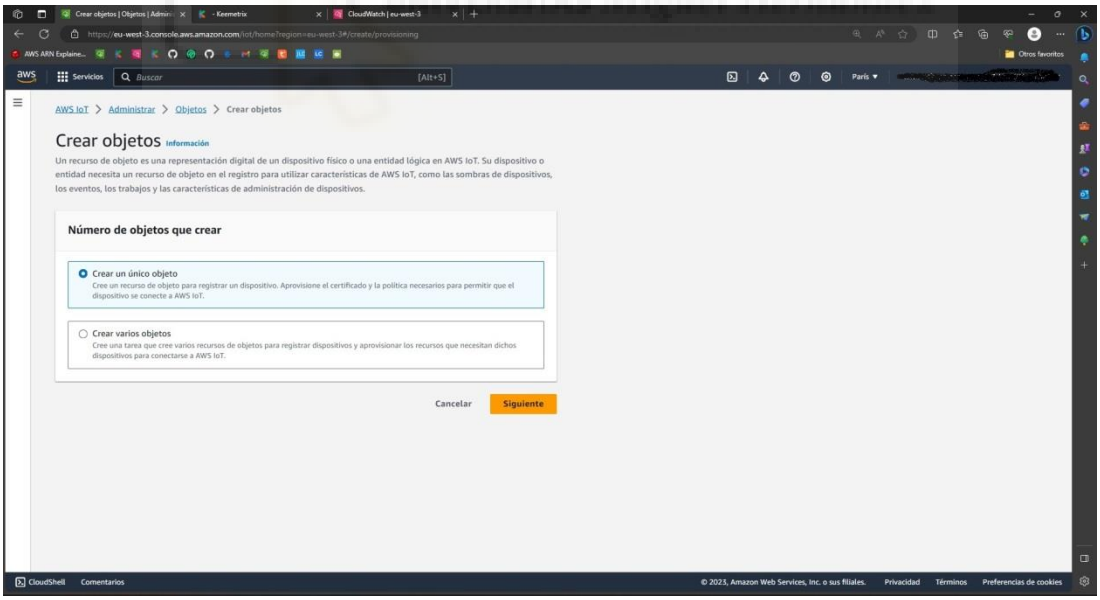


Figura 8.4 Primer paso para crear un objeto

En el campo Nombre del objeto escribimos el nombre que se le quiere proporcionar al mismo, recomiendo a nivel práctico dar un identificador común para los mismos tipos de objetos y un ID separados por _ más otro identificador numérico esto, en la aplicación que

nos ocupa nos sirve para nombrar las placas, saber a qué centro pertenecen y saber que placa es dentro del mismo, quedando algo así: Placa_5254rgf65sdf4g6d4_1.

Podemos, además añadir distintos campos al mismo objeto, en esta aplicación solo hemos creado el tipo de objeto, Centro_Nº, para identificar más rápidamente dentro de las herramientas de AWS el conjunto de elementos de cada lugar. Al acabar pulsamos en Siguiente.

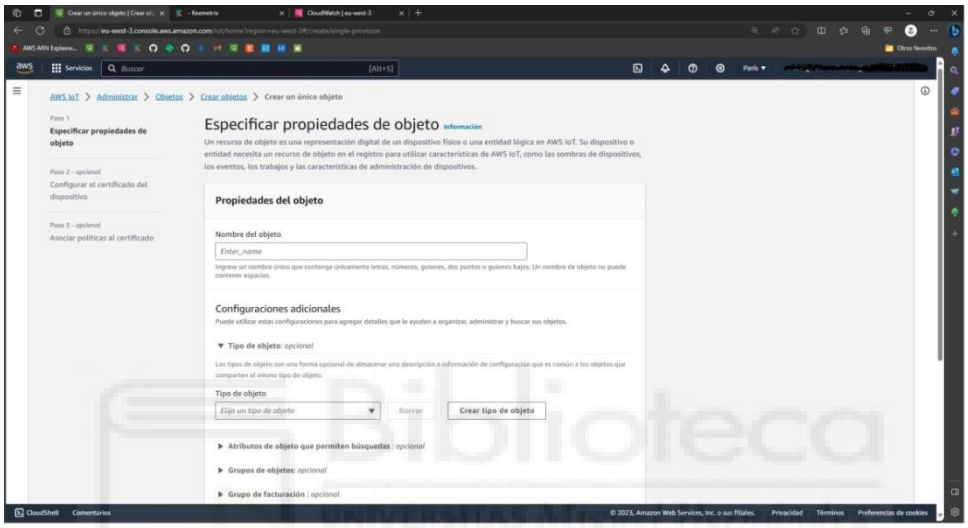


Figura 8.5 Segundo paso para crear un objeto

Seleccionamos Generar automáticamente un certificado nuevo (esta preseleccionado por defecto) y pulsamos en siguiente.

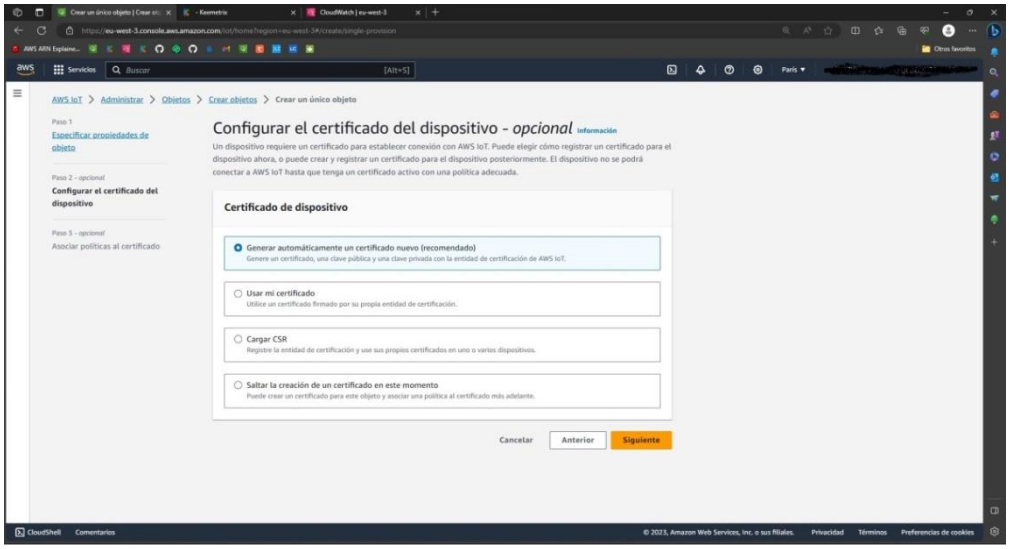


Figura 8.6 Tercer paso para crear un objeto

En esta última pestaña seleccionaremos la política asociada al objeto, hasta el momento cada objeto tiene una única y política creada para el mismo por lo que se le suele asociar a la política el mismo nombre que al objeto, ya que no solo no genera conflicto ya que son tipos de archivos diferentes si no que ayuda a encontrarlos y enlazarlos en caso de tener que modificarlos más adelante.

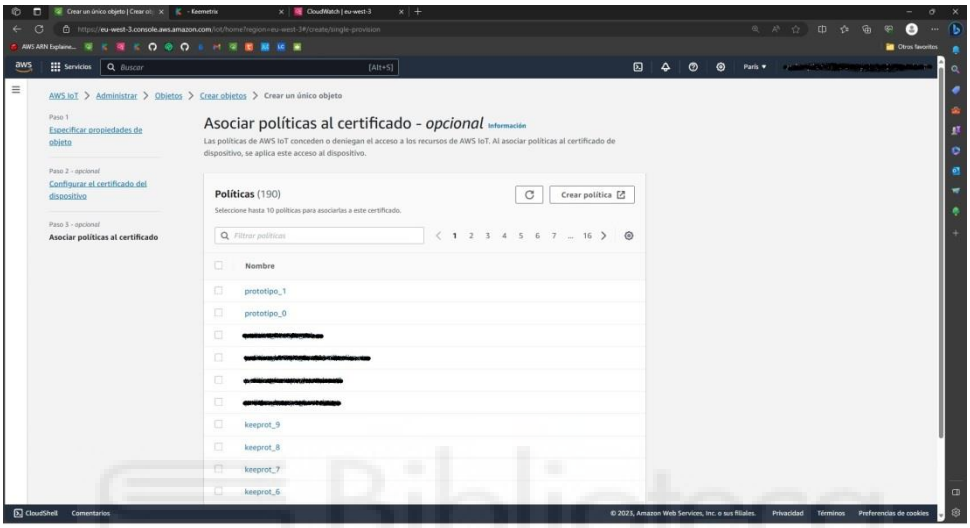


Figura 8.7 Asignación de políticas

Una vez asociada la política (que podría hacerse posteriormente en caso de no querer generarla o no tenerla generada en el momento de la creación del objeto) se mostrara una nueva pestaña donde se podrán descargar 5 certificados, uno de ellos es propio a la cuenta de AWS y nunca cambia, otro es la versión publica del mismo y hasta la fecha no ha sido empleado en el proyecto, los otros 3 son certificados propios del objeto y una vez cerrada la pestaña no se podrán volver a generar lo que nos deja la única opción de borrar el objeto e caso de que se perdieran, estos certificados son el propio certificado del objeto, el certificado privado del objeto y un certificado público del mismo, este último no se ha empleado en el proyecto.

8.2 Creación de la política:

Entenderemos por política el conjunto de instrucciones que nos permitirán identificar el dispositivo e intercambiar información con él desde nuestros servidores.

Explicaré paso a paso todo el proceso para la creación de la misma.

Volviendo a la página principal de AWS IOT hacemos click en Seguridad > Políticas

Figura 8.8 Primer paso para generar políticas

Hacemos click en Crear política en el margen superior derecho de la pantalla.

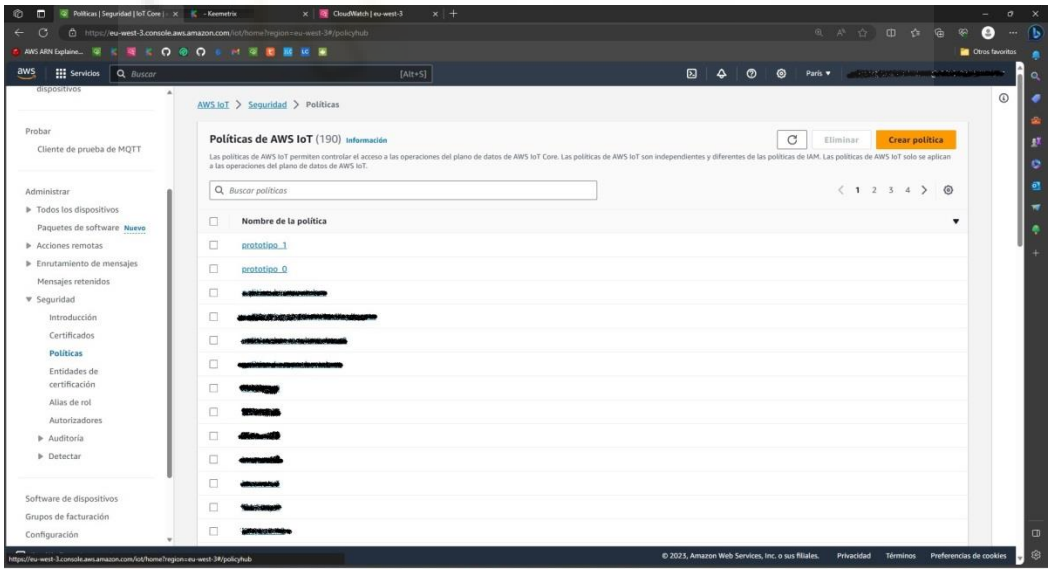


Figura 8.9 Segundo paso para generar políticas

En esta pestaña seleccionamos JSON (que es en el formato en el que se forman los archivos) y tras darle un nombre escribimos el código de la política, el cual se encuentra en el anexo II, pero se le han eliminado todos los elementos que podrían generar fallos de seguridad.

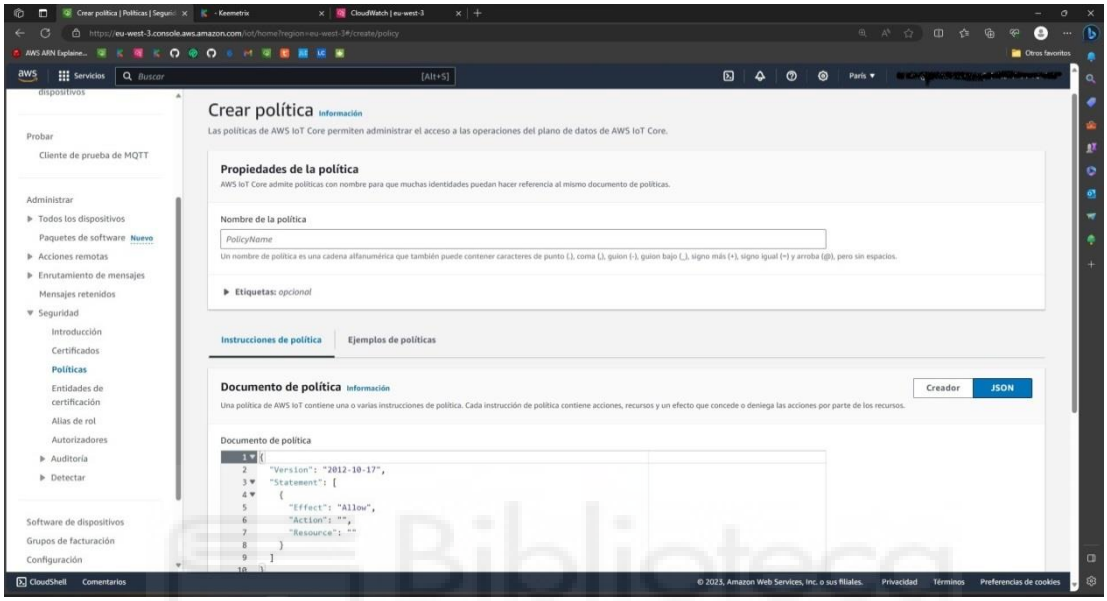


Figura 8.10 Tercer paso para generar políticas

9. PROGRAMACIÓN

Dividiremos el código en dos partes, el **config.json**, sección de información donde se guarda el nombre del objeto, los certificados relevantes de AWS IOT, así como el SSID de la red Wi-Fi del centro en el que se vaya a hacer la instalación como su contraseña, adjunto en el anexo III, se han borrado los elementos identificativos por razones de seguridad.

La segunda parte del código es el **src** (según los archivos de Visual Studio CODE) que es el script principal que hace funcionar la electrónica, este se encuentra en el anexo 23.

9.1 Config.json:

Este código como ya se ha comentado graba las credenciales necesarias para el reconocimiento de la placa tanto para el Wi-Fi del centro como para el servidor de AWS,

esta información relevante se almacena en la memoria no volátil del Esp32, para conseguir esto seguiremos estos pasos:

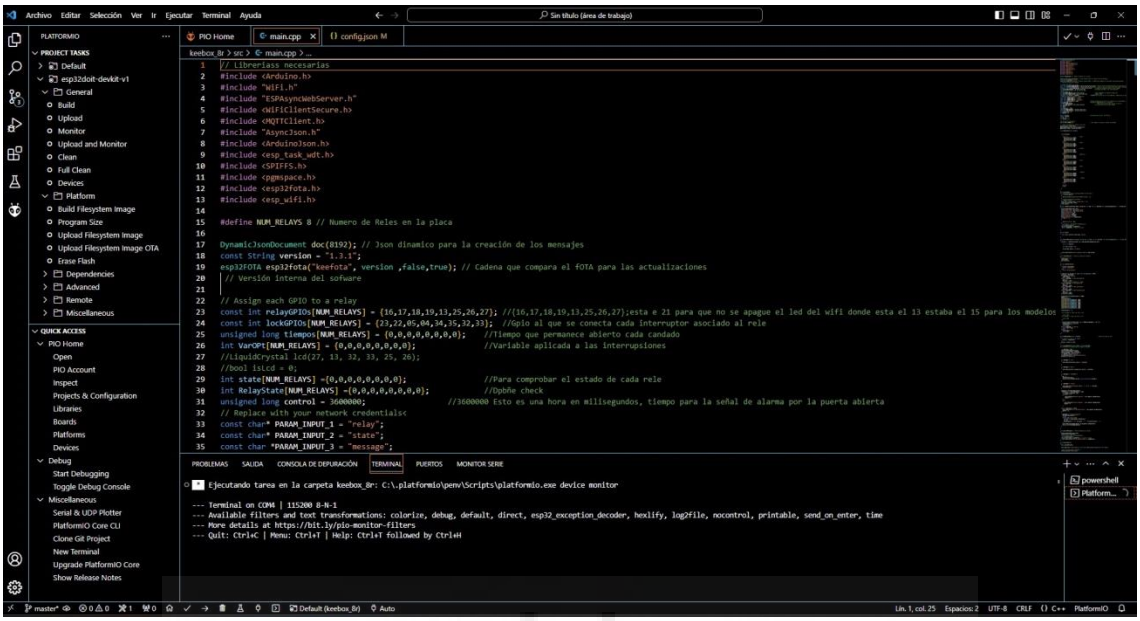


Figura 9.1 Entorno VS CODE con PLATFORM IO

Desde la pantalla de inicio del Visual Studio CODE seleccionamos la extensión de Platform.IO y con el config.json actualizado para la placa correspondiente (este se encuentra en la pestaña data dentro del proyecto como se aprecia en la imagen de inicio de Visual Studio CODE de la figura 5.4) haremos clic en Build filesystems imagen lo que preparará el archivo para ser grabado en la memoria. Una vez finalizado el proceso, puede llevar unos segundos, haremos click en Upload filesystem image, lo que cargará el archivo en la memoria no volátil del chip, como ya se ha comentado.

La relevancia de este proceso radica en que ya que gracias el método FOTA (actualización inalámbrica de firmware) la electrónica puede actualizarse de forma remota, mejorando así el rendimiento de la misma o arreglando bugs no resueltos de antemano, el problema radicaba en que este FOTA borraba toda la memoria del chip, al emplear la memoria no volátil para la información indispensable no es necesario la recuperación de la electrónica para su mejora de funcionamiento (siempre y cuando esta no tenga que ver con una actualización física).

9.2 Src(o main):

Simplemente hablaremos de las funciones más relevantes del código ya que el anexo IV está debidamente comentado.

Empezaremos por el setup: Inicializamos la velocidad del monitor serie a 115200 para poder visualizar correctamente por el serial del programa.

Generamos xTaskCreate(loop2,"mi_tarea",1000,NULL,1,NULL), esta función propia del ESP32, nos permite generar otro bucle loop infinito en el segundo núcleo del procesador del Esp32 lo que nos permite llevar a cabo dos procesos separados sin que ni8nguno interfiera con el otro, dejando las interrupciones siendo gestionadas por este segundo loop y la conectividad por el principal.

Activamos la función SPIFFS que nos sirve para leer el json grabado de antemano. También inicializamos las interrupciones así como diversas funciones propias del Esp32, Junto a esto marcamos 4 pines como input por razones de precaución dado que 3 de los seleccionados son únicamente utilizables de esta forma, asegurando así una buena funcionabilidad. En el caso de las interrupciones se usó el modo CHANGE, el cual establece que salte la interrupción cada vez que se cambie de estado (por flaco de bajada o subida) ya que el esp32 demostró un funcionamiento erróneo al intentar funcionar solo por flanco de subida o bajada, haciendo que estos dos estado y el CHANGE fueran exactamente iguales, el código se adaptó en consonancia, y para evitar futuros errores se estableció así por si más adelante se corrigiera el error y dichas interrupciones dejasen de funcionar correctamente.

```
void initSPIFFS(){                                  // Esta función lee el config,h guardado en la memoria no volátil
  if (!SPIFFS.begin()){
Serial.println("Cannot mount SPIFFS volume...");
  }

File file = SPIFFS.open("/config.json", "r");// Lee del archivo
  deserializeJson(doc, file);
  file.close();
  }
```

Figura 9.2 initSPIFFS()

Marcamos los 3 GPIO que controlaran el led de información como output por las mismas razones que el input anterior, un funcionamiento sin errores remarcables.

Seguidamente encontramos un bucle for que revisa el estado de cada "LOCKGPIO" es decir, lee si la cerradura conectada a la placa está abierta o cerrada para establecer el estado inicial de la misma, se ha observado que si no está conectada se marca abierto por defecto y al conectarse salta la interrupción y la marca como cerrada.

Activamos los eventos propios de las librerías Wi-Fi, y del servidor HTML, el cual solo se usa en caso de necesidad, este medio solo se configura en el setup y no hay ninguna función fuera de él, esto funciona correctamente aunque este al margen de los loops principales. Nos permite una conexión vía ip (ya que las conocemos de antemano) para que en caso de desconexión del servidor principal pueda funcionar con cierta normalidad.

Para terminar el setup encontramos una serie de instrucciones que comprueban si la versión actual del código es inferior a la subida al servidor de la empresa, en cuyo caso inicia la actualización, en caso de encontrarse la misma versión o una superior ignoraría el proceso.

Las funciones más relevantes serían las que controlan la conexión al Wi-Fi y al servidor AWS así como la función encargada de abrir las puertas, la de procesar los mensajes llegados del servidor y la de publicar mensajes en el mismo.

Las funciones que sirven para controlar el estado de la conexión inalámbrica serian conexiowifi() y las propias de los eventos de dicha librería, que se encargan de conectarse y en caso de desconexión o error en el mismo tratar de reconectarse a la misma.

```
void ConexionWifi(){
        WiFi.mode(WIFI_STA);
        WiFi.begin(doc["ssid"].as<const char *>(), doc["password"].as<const char *>());
         Serial.println(WiFi.localIP());
        delay(1000);
        if(WiFi.status() != WL_CONNECTED && MC == 0){
                esp_wifi_set_mac(WIFI_IF_STA, const_cast<uint8_t*>(mac));
        }
        if(WiFi.status() != WL_CONNECTED && MC == 1){
                esp_wifi_set_mac(WIFI_IF_STA, const_cast<uint8_t*>(mac0));
        }

}
```

Figura 9.3 ConexionWifi()

connectAWS() comprueba en primera instancia si está correctamente conectada a la red Wi-Fi (sería un gasto inútil de procesador de no estarlo) y en caso afirmativo empieza la conexión al servidor, comparando las credenciales y las políticas para establecer la misma. Toda la comunicación vía servidores AWS de AMAZON esta encriptada gracias a estas credenciales y políticas lo que permite una seguridad extremadamente alta a jaqueos externos de señal.

```
void connectAWS(){ // Nos conecta con amazon, si hay problemas revisar la informacion en el config o los certificados en el
        while (WiFi.status() != WL_CONNECTED) {
                ConexionWifi();
        }
                        // Configure WiFiClientSecure to use the AWS IoT device credentials
        net.setCACert(doc["aws_cert_ca"].as<const char *>());
        net.setCertificate(doc["aws_cert_crt"].as<const char *>());
        net.setPrivateKey(doc["aws_cert_private"].as<const char *>());
                        // Connect to the MQTT broker on the AWS endpoint we defined earlier
        client.begin(doc["aws_iot_endpoint"].as<const char *>(), 8883, net);
                        // Create a message handler
        client.onMessage(messageHandler);
        client.setKeepAlive(40);
        String cen = doc["centro_id"];
        String keebox_ide = doc["keebox_id"];
        String the_ip = WiFi.localIP().toString().c_str();
        String payload = "{\"centro_id\": \"" + cen + "\",\"keebox_id\": \"" + keebox_ide + "\",\"ip\":\"" + the_ip + "\"}";
        char envio[256];
        strcpy(envio, payload.c_str());
        client.setWill("keebox/off", envio);
        Serial.print("Connecting to AWS IOT");
        while (!client.connect(doc["thingname"])){
                Serial.print(".");
                delay(100);
        }
        if (!client.connected()){
                Serial.println("AWS IoT Timeout!");
                 return;
        }
                        // Subscribe to a topic
        client.subscribe(doc["aws_iot_suscribe_topic"].as<String>());
        client.subscribe(doc["aws_iot_suscribe_topic_global"].as<String>());
        publicaMensaje("conectado", "", 5, 0, dirIp(), version);
        Serial.println("AWS IoT Connected!");
        CambiaColor('a');
}
```

Figura 9.4 connectAWS()

Messagehandler() controla los mensajes que llegan desde el servidor, si no concidiera con ninguno se descartaría y devolvería un 404 al no haberse encontrado dicho comando, actualmente se puede resetear la placa, abrir una puerta, comprobar la IP o la

MAC de la placa, cambiar la MAC de la misma y comprobar el estado de las puertas conectadas.

```
void messageHandler(String &topic, String &payload){    // Controla las ordenes que le llegan a la placa
        int devuelta = 400;
        StaticJsonDocument<200> doc1;
        deserializeJson(doc1, payload);
        String message = doc1["message"];
        int puerta = doc1["puerta"];
        String dispositivo = doc1["dispositivo"];
        String id = doc1["id"];

        if (message == "ip"){
                publicaMensaje(dirIp(),id,0,0,"",version);
        }
        if (message == "mac"){
                publicaMensaje(WiFi.macAddress(),id,0,0,"",version);
        }
        if (message == "change_mac"){
                MC = 1;
                makeRandomMac(mac0);
                esp_wifi_set_mac(WIFI_IF_STA, const_cast<uint8_t*>(mac0));
        }

        if (message == "reset"){
                publicaMensaje("Reseteando Keebox", "", 0, 0, "", version);
                reseteaPlaca();
        }

        devuelta = Abre(message,puerta);
        if(devuelta >= 200 && devuelta < 400){
                int valord = devuelta -200;
                if(valord == 0) {
                        publicaMensaje("Puerta abierta ", id,1,puerta, dispositivo);
                        patata = 0;
                }
                else{
                        publicaMensaje("Puerta especial ", id,1,valord, dispositivo);
                        patata = 0;
                }
        }else if(devuelta == 400){
                publicaMensaje("Error abriendo puerta ", id,0,puerta, dispositivo);
                 patata = 0;
        }
        if(message == "estado_puertas"){                        //Envía el estado de las puertas para el formato
        kerong 1 es abierto o no conectado y 0 es cerrado.
                 String estado_puerta = "";
                int A;
                for(int t=0; t<=7; t++){
                        state[t] = digitalRead(lockGPIOs[t]);
                        A = t+1;
                         estado_puerta += "R"+ String(A) + ":" + String(state[t]) + ",";
                }
                 publicaMensaje(estado_puerta, id,6,puerta, dispositivo);
        }
}
```

Figura 9.5 messsageHandler()

La función publicamensaje() hace lo opuesto, cuando se la llama genera un texto con las variables introducidas y este aparece en AWS.

```
void publicaMensaje(String texto,String id,int tipo = 0,int puertaid
= 0,String dispositivo = "",String ver= ""){
                        // Publica un mensaje Json
            String s = creaMensaje(texto, id,
tipo,puertaid,dispositivo,ver);
            int n = s.length();
            char char_array[n + 1];
            strcpy(char_array, s.c_str());
            client.publish(doc["aws_iot_publish_topic"], char_array);
}
```

Figura 9.6 publicamensaje()

La última función relevante es la función abre() que es la llamada cuando se decide abrir una puerta a esta función se la puede llamar de dos formas o de la forma simple que se le pasa un número de relé y se abre la misma durante un segundo o un comando para abrir una puerta principal, esto solo se suele hacer con las placas de 4 relés pero por temas de compatibilidad y eficiencia de uso del material se añadió en este código también porque en caso de no ser necearía una placa extra para el control de la puerta principal una de trasteros puede encargarse de este control. Tras la apertura ya sea de 1 o 5 segundos, en el que la placa se congela debido al delay (esto siempre se trató de evitar dado a que usar delays en un proceso de este tipo es una mala técnica pero debido a ciertos errores de funcionamiento me vi forzado por parte de la empresa a implementarlo) haciendo que los mensajes siguientes se queden en cola y a la espera, se cierran todos los relés, si ya estaban cerrados no ocurren cambios, esto también se hace para asegurarse de que ningún elemento reciba corriente durante demasiado tiempo.

```
int Abre(String message,int rele1 = 0,long duracion1 = 900){ //Lee el mensaje y abre el relé en consecuencia

        duracion = duracion1;

        if (message == "abre"){

                rele = rele1;

        }else if(message == "abre_principal"){

                rele = 4;

                duracion = 4900;

                //digitalWrite(relayGPIOs[2], HIGH);

        }else if(message == "abre_garaje"){

                rele = 3;

                duracion = 4900;

        }else if(message == "abre_interna"){

                rele = 2;

                duracion = 4900;

        }else if(message == "abre_secundaria"){
```

10. INSTRUCCIONES PARA UNA INSTALACIÓN DE UN CENTRO DE TRASTEROS AUTOMATIZADO

10.1 PROCESO INICIAL

Una vez formalizado el contrato con la empresa comenzará el cambio de información vinculante.

Se valorará en el plano del centro el número de trasteros a automatizar y la disposición de estos a lo largo del centro, así como el tipo de instalación eléctrica de la que se dispondrá, si se colocaran bandejas eléctricas, todo deberá ir colocado en un falso techo o si todo se deberá ajustarse sobre los mismos trasteros o en los techos y paredes de la instalación.

Una vez conocidos estos datos se distribuirán sobre el papel los distintos elementos necesarios para el funcionamiento de la instalación, que vienen a ser:

- Número de PCBs necesarias.
- Router 4G.
- Repetidores de señal (estos solo serán necesarios si el centro es de un tamaño considerable o de si la distribución del mismo lo requiere).
- Fuentes de alimentación.
- Raspberry Pi

Se tratará de abarcar la mayor cantidad posible de puertas con las placas de 8 relés para una mayor eficiencia en el uso de recursos y se colocará en primera instancia placas adaptables para las puertas de acceso al recinto, ya sean peatonales, persianas de garaje o puertas internas que separan un recibidor del resto del recinto.

Para las puertas peatonales exteriores se recomienda cerraduras sin, por seguridad, esto es por si se diera el caso de que algún usuario accionase una de las puertas principales por error pasado el tiempo establecido estas quedasen cerradas y los centros seguro.

10.2 DISTRIBUCIÓN DEL MATERIAL

Sabiendo cual será la cantidad de material necesario podremos seguir con la instalación.

Para empezar se colocará un router 4G a la entrada del local, lo más alto posible y tratando de asegurar la mejor cobertura disponible, este router 4G empleado permite programar reinicios cada cierto tiempo, además de reinicios de emergencia a través de SMS. Este router controlará las placas principales de entrada cercanas y los controladores NFC instalados si los hubiese.

Conectado vía Ethernet se instalará una Raspberry cuya función será la de controlar el router y la alimentación principal de las placas si se requiriera un reinicio forzado. Para locales más amplios se conectaría un repetidor de señal conectado vía Ethernet al router 4G en las zonas más alejadas.

Normalmente y para reducir costes se colocará una sola fuente de alimentación de 12 V y 21 A o superior cuya alimentación (230 V) pasara a través de un relé conectado a la Raspberry y posteriormente conectará en serie todas las placas, para esto hará falta un cable de dos hilos de 1,5 de la longitud requerida para abarcar toda la distancia entre placas, en caso de que se utilicen fuentes individuales (12 V por 6 A) la línea que controla todas las fuentes pasará por la Raspberry, permitiendo así el mismo nivel de control.

10.3 INSTALACIÓN DEL MATERIAL

Una vez instalada la electrónica en los lugares aproximados en los que se diseñó, decimos aproximados por que se suele dar el caso de variaciones de la realidad con respecto a los planos, lugares en los que no se puede taladrar o no se instalaron los elementos de sujeción prometidos, se pasará a conectar los cables (que previamente habrán sido instalados por electricistas contratados por el dueño del centro siguiendo nuestras especificaciones) que conectan con las cerraduras, generalmente habrá sido dado un orden concreto de a que relé deberá conectarse cada puerta para una mayor eficiencia en la instalación, en caso de un error de conexión (confundir una puerta por otra) se podrá arreglar vía software desde la central, pero esto retrasará ligeramente el proceso.

Todo el material será proporcionado debidamente etiquetado y explicado para que los técnicos contratados puedan proceder con la instalación rápidamente y con los menores riesgos posibles.

11. INSTALACIONES REALIZADAS

Las instalaciones realizadas con esta tecnología, en cualquiera de sus versiones de desarrollo, serían:

1. Keeperville en elche.
2. Petits locals en Villanova y la Geltru.
3. Grupo Hortelano e hijos dos centros en Elche.
4. Cartagena Spain en Cartagena.
5. Trasters TuBox en Esplugues de Llobregat.
6. IlliceBox en elche.
7. El cuarto en Gavà.
8. AlmaZEN en Albacete.
9. Trasteros 365 en Castelldefels y Gavà.
10. TrasPrat en el Prat de Llobregat.

12. CONCLUSIONES Y TRABAJO FUTURO

12.1 CONCLUSIÓN

Para terminar es necesaria una recapitulación de todo lo anteriormente visto, todo el proceso de creación y mejora de esta tecnología, con los errores y los problemas comentados, fueron elaborados durante los 2 años trabajados en esta empresa, toda la información proporcionada ha sido gracias a todo lo aprendido y experimentado en este tiempo.

Podremos considerar un éxito el objetivo marcado al inicio de este proyecto puesto que se consiguió rediseñar la tecnología requerida y establecer unas pautas claras para su instalación, reparación (en caso de necesidad) y puesta en marcha para la empresa Keemetrix Systems S.L. consiguiendo que gracias a su sistema puntero en gestión de

alquileres y esta tecnología este llamando la atención rápidamente en este sector, consiguiendo nuevos contratos con el tiempo.

12.2 TRABAJO FUTURO

En cuanto al futuro de esta tecnología será emplearla en todas las instalaciones completamente automatizadas.

La mejora de esta tecnología puede basarse tanto en la mejora del acabado final como en la integración de la misma en carcasas personalizadas, hasta la fecha se usan cajas eléctricas herméticas fabricadas de material ABS de buena calidad, como en la integración de diversos sensores.

Los sensores más útiles a día de hoy para este propósito son los de temperatura y humedad puesto que en diversas instalaciones debido a la ubicación de los locales estos compartían comunidad con edificios de viviendas y de vez en cuando se pueden producir goteras (como fue el caso en 3 centros), si esto ocurre durante la instalación es fácilmente subsanable, puesto que se avisa a los peritos pertinentes y el fontanero correspondiente hace su labor antes de que cualquier equipo se pueda ver afectado, pero si en las placas de 4 relés (dado que tienen más pines disponibles) se añadiera esta clase de sensores se podría establecer una red de información que nos indicara si el estado del local es el idóneo o si por el contrario la humedad se disparase podríamos incluso acotar la zona donde se están produciendo estas fugas, ya que la colocación de la electrónica es conocida por la empresa.

Esto puede facilitar mucho la solución del problema antes de que a los dueños de trasteros les afecte esta entrada de agua y puedan sufrir pérdidas materiales en sus bienes almacenados, haciendo así que el dueño del local ni tenga que lidiar con los problemas consiguientes con los seguros. Haciendo que la decisión de implementarlos, aunque encarezca ligeramente el producto final y complicase el código necesario para la misma, sería altamente aceptada por el cliente y agradecida por los usuarios.

Otros futuros proyectos embarcados por la empresa a raíz del estudio de esta tecnología son el desarrollo de una llave cuyo único propósito es conectarse a la cerradura para poder saber si esta funciona correctamente sin arriesgar la placa electrónica y para conocer si los cables están en el lugar adecuado.

También se estuvo desarrollando un teclado con lector NFC para aquellas personas que no pudieran, o no estuvieran, acostumbradas a emplear el teléfono móvil pero quedó pospuesto debido a temas más apremiantes. Se progresará en ese tema en un futuro.

13. BIBLIOGRAFÍA

[1]https://docs.espressif.com/projects/arduino-esp32/en/latest/getting_started.html consultado en octubre de 2021

[2]https://randomnerdtutorials.com/esp32-pinout-reference-gpios/ consultado en octubre de 2021

[3]https://products.espressif.com/#/product-selector?language=en&names= consultado en noviembre de 2021

[3]https://www.luisllamas.es/encender-un-led-con-arduino/ consultado en octubre de 2021

[4]https://www.luisllamas.es/esp32-interrupciones-hardware/ consultado en octubre de 2021

[5]https://www.luisllamas.es/arduino-salida-rele/ consultado en octubre de 2021

[6]https://programarfacil.com/blog/arduino-blog/resistencia-pull-up-y-pull-down/ consultado en octubre de 2021

[7]https://programarfacil.com/esp8266/esp32/ consultado en octubre de 2021

[8]https://github.com/espressif consultado en octubre de 2021

[9] https://easyeda.com/es consultado en octubre de 2021

[10] https://code.visualstudio.com/ consultado en octubre de 2021

[11] https://aws.amazon.com/es/iot/ consultado en octubre de 2021

14. ANEXOS

ANEXO I

En este anexo encontraremos los enlaces a las hojas de datos de los componentes empleados en el proyecto.

[1] Espressif-Systems-ESP32-WROOM-32E: https: //www.lcsc.com/product-detail/WiFi-Modules_Espressif-Systems-ESP32-WROOM-32E-N8R2_C5361945.html

[2] Pulsador_BRIGHT-TSA063G50-250_C294565: https://www.lcsc.com/product-detail/Tactile-Switches_BRIGHT-TSA063G50-250_C294565.html

[3] Res100K_UNI-ROYAL-Uniroyal-Elec-25121WF1003T4E_C52842: https://www.lcsc.com/product-detail/Chip-Resistor-Surface-Mount_UNI-ROYAL-Uniroyal-Elec-25121WF1003T4E_C52842.html

[4] TE-Connectivity-640456-3_C8650: https://www.lcsc.com/product-detail/Wire-To-Board-Wire-To-Wire-Connector_TE-Connectivity-640456-3_C86503.html

[5] Shikues-MMBT2222A_C171740: https://www.lcsc.com/product-detail/Bipolar-Transistors-BJT_Shikues-MMBT2222A_C171740.html

[6] UNI-ROYAL-Uniroyal-Elec-25121WF1001T4E_C54315: https://www.lcsc.com/product-detail/Chip-Resistor-Surface-Mount_UNI-ROYAL-Uniroyal-Elec-25121WF1001T4E_C54315.html

[7] Ningbo-Songle-Relay-SRD-12VDC-SL-C_C30431: https://www.lcsc.com/product-detail/Power-Relays_Ningbo-Songle-Relay-SRD-12VDC-SL-C_C30431.html

[8] XFCN-PZ254V-11-02P_C492401: https://www.lcsc.com/product-detail/Pin-Headers_XFCN-PZ254V-11-02P_C492401.html

[9] Cixi-Kefa-Elec-KF2EDGR-5-08-5P_C441207: https://www.lcsc.com/product-detail/Pluggable-System-Terminal-Block_Cixi-Kefa-Elec-KF2EDGR-5-08-5P_C441207.html

[10] UNI-ROYAL-Uniroyal-Elec-25121WF1002T4E_C270956: https://www.lcsc.com/product-detail/Chip-Resistor-Surface-Mount_UNI-ROYAL-Uniroyal-Elec-25121WF1002T4E_C270956.html

[11] BOOMELE-Boom-Precision-Elec-DC-005-2-0_C16214: https://www.lcsc.com/product-detail/AC-DC-Power-Connectors_BOOMELE-Boom-Precision-Elec-DC-005-2-0_C16214.html

[12] DEGSON-DG125-5-0-02P-14-00A-H_C784572: https://www.lcsc.com/product-detail/Screw-terminal_DEGSON-DG125-5-0-02P-14-00A-H_C784572.html

[13] Xucheng-Elec-C3130_C3130: https://www.lcsc.com/product-detail/Fuse-Holders_Xucheng-Elec-C3130_C3130.html

[14] UMW-Youtai-Semiconductor-Co---Ltd--LM2596S-3-3_C347420: https://www.lcsc.com/product-detail/DC-DC-Converters_UMW-Youtai-Semiconductor-Co-Ltd-LM2596S-3-3_C347420.html

[15] Rubycon-16TZV680M10X10-5_C365678: https://www.lcsc.com/product-detail/Aluminum-Electrolytic-Capacitors-SMD_Rubycon-16TZV680M10X10-5_C365678.html

[16] KNSCHA-RVT220UF16V67RV0015_C2887273: https://www.lcsc.com/product-detail/Aluminum-Electrolytic-Capacitors-SMD_KNSCHA-RVT220UF16V67RV0015_C2887273.html

[17] MCC-Micro-Commercial-Components-SK44BL-TP_C151740: https://www.lcsc.com/product-detail/Schottky-Barrier-Diodes-SBD_MCC-Micro-Commercial-Components-SK44BL-TP_C151740.html

[18] Sunltech-Tech-SLS5D28S330MTT_C364126: https://www.lcsc.com/product-detail/Power-Inductors_Sunltech-Tech-SLS5D28S330MTT_C364126.html

[19] Everlight-Elec-15-21-GHC-XS1T1-2T_C131257: https://www.lcsc.com/product-detail/Light-Emitting-Diodes-LED_Everlight-Elec-15-21-GHC-XS1T1-2T_C131257.html

[20] UNI-ROYAL-Uniroyal-Elec-25121WJ0471T4E_C24919: https://www.lcsc.com/product-detail/Chip-Resistor-Surface-Mount_UNI-ROYAL-Uniroyal-Elec-25121WJ0471T4E_C24919.html

[21] LIZ-Elec-CR2512J10150G_C102539: https://www.lcsc.com/product-detail/Chip-Resistor-Surface-Mount_LIZ-Elec-CR2512J10150G_C102539.html

[22] Diodes-Incorporated-BZT52C3V3-7-F_C151564: https://www.lcsc.com/product-detail/Zener-Diodes_Diodes-Incorporated-BZT52C3V3-7-F_C151564.html

[23] CCTC-TCC0805X5R226M160FT_C380339: https://www.lcsc.com/product-detail/Multilayer-Ceramic-Capacitors-MLCC-SMD-SMT_CCTC-TCC0805X5R226M160FT_C380339.html

[24] MEIHUA-MHSC110RGBCT_C482558: https://www.lcsc.com/product-detail/Light-Emitting-Diodes-LED_MEIHUA-MHSC110RGBCT_C482558.html

ANEXO II

Políticas

En este anexo se encuentra el código perteneciente a las políticas creadas para AWS.

```
{"Version": "2012-10-17",
 "Statement": [{
        "Effect": "Allow",
        "Action": "iot:Connect",
        "Resource": "arn:aws:iot:eu-west-3:378507517308:client/NombreObjeto_ID del
centro_Nº"
        },
        {
        "Effect": "Allow",
        "Action": "iot:Subscribe",
        "Resource": [
                "arn:aws:iot:eu-west-3:378507517308:topicfilter/_____/ID del
centro/Nº/sub",
                "arn:aws:iot:eu-west-3:378507517308:topicfilter/_____/ID del centro/*/sub"
        ]
        },
        {
        "Effect": "Allow",
        "Action": "iot:Receive",
        "Resource": "arn:aws:iot:eu-west-3:378507517308:topic/_____/ID del
centro/Nº/sub"
        },
        {
        "Effect": "Allow",
        "Action": "iot:Publish",
        "Resource": "arn:aws:iot:eu-west-3:378507517308:topic/_____/pub"
        },
        {
         "Effect": "Allow",
        "Action": "iot:Publish",
        "Resource": "arn:aws:iot:eu-west-3:378507517308:topic/_____/off"
        }
 ]
}
```

ANEXOIII

Config.h

En este anexo se encuentra el primer código a grabar en los dispositivos, este se graba en la memoria no volátil, la cual no se borra con las actualizaciones posteriores, es el código que le dice a la PCB quien es.

```
{ "thingname": "Nombre_objeto",
        "aws_iot_publish_topic": "_____",
        "aws_iot_suscribe_topic": "_____/Nº/sub",
        "aws_iot_suscribe_topic_global": "_____/*/sub",
        "centro_id": "_____",
        "keebox_id": Nº,
        "tipo_puerta": _,
        "ssid": "Nombre_de_la_red",
        "password": "contraseña",
        "aws_iot_endpoint": "_____",
        "aws_cert_ca": "\n-----BEGIN CERTIFICATE-----\n_____\n-----END CERTIFICATE-----\n",
        "aws_cert_crt": "\n-----BEGIN CERTIFICATE-----\n_____\n-----END CERTIFICATE-----\n",
        "aws_cert_private": "\n-----BEGIN RSA PRIVATE KEY-----\n____\n-----END RSA PRIVATE KEY-----\n",
        "ip":
}
```

ANEXO IV

SRC (MAIN)

En este anexo se detalla el código Fuente de los dispositivos, el código principal que hace que todo funcione como debe.

```
#include <Arduino.h>                    // Librerias necesarias

#include "WiFi.h"
#include "ESPAsyncWebServer.h"
#include <WiFiClientSecure.h>
#include <MQTTClient.h>
#include "AsyncJson.h"
#include <ArduinoJson.h>
#include <esp_task_wdt.h>
#include <SPIFFS.h>
#include <pgmspace.h>
#include <esp32fota.h>
#include <esp_wifi.h>

#define NUM_RELAYS 8                    // Numero de Relés en la placa

DynamicJsonDocument doc(8192);          // json dinámico para la creación de los mensajes
const String version = "1.3.0";         // Versión interna del software
esp32FOTA esp32fota("keefota",version,false,true); // Cadena que compara el FOTA para las actualizaciones
const int relayGPIOs[NUM_RELAYS] = {16,17,18,19,13,25,26,27}; //GPIO al que se conecta cada rele
const int lockGPIOs[NUM_RELAYS] = {23,22,05,04,34,35,32,33};  //GPIO al que se conecta cada interrup
unsigned long tiempos[NUM_RELAYS] = {0,0,0,0,0,0,0,0}; //Tiempo que permanece abierto cada candado
int VarOPt[NUM_RELAYS] = {0,0,0,0,0,0,0,0};          //Variable aplicada a las interrupciones
int state[NUM_RELAYS] ={0,0,0,0,0,0,0,0};            //Para comprobar el estado de cada rele
int RelayState[NUM_RELAYS] ={0,0,0,0,0,0,0,0};       //Doble check
unsigned long control = 3600000;                     //3600000 Esto es una hora en milisegundos,
tiempo para la señal de alarma por la puerta abierta
const char* PARAM_INPUT_1 = "relay";
const char* PARAM_INPUT_2 = "state";
const char *PARAM_INPUT_3 = "message";
const int timeThreshold = 1000;                 //Tiempo asignado para evitar los rebotes en la interrupción
long startTime = 0;                             //Variable complementaria a la anterior
int ReleX[NUM_RELAYS];                          //Marca que rele hay que cerrar con millis()
int Prev[NUM_RELAYS] = {0,0,0,0,0,0,0,0};       //El estado anterior de cada candado
int ledrror = 0;
int rele;
long duracion = 1000;
long abreP[NUM_RELAYS] = {0,0,0,0,0,0,0,0 };
long timer[NUM_RELAYS] = {0,0,0,0,0,0,0,0 };
int intento_aws = 0;
int patata = 0;

uint8_t mac[6];                                 //Variable para la Mac  alternativa
uint8_t mac0[6];
int MC = 0;
#define WDT_TIMEOUT 60                  // Establece el número de segundos para reinicir por el wachdog
```

```
AsyncWebServer server(80);                    // Crear objeto AsyncWebServer en el puerto 80
WiFiClientSecure net = WiFiClientSecure();
MQTTClient client = MQTTClient(406);



void CambiaColor(char color){      //Función que controla la iluminación del led RGB INFO
        switch(color){
                case 'a':                       //Azul
                        digitalWrite(12,HIGH);
                        digitalWrite(15,LOW);
                        digitalWrite(14,LOW);
                break;
                case 'r':                       //Rojo
                         digitalWrite(12,LOW);
                         digitalWrite(15,HIGH);
                        digitalWrite(14,LOW);
                break;
                case 'v':                       //Verde
                        digitalWrite(12,LOW);
                        digitalWrite(15,LOW);
                        digitalWrite(14,HIGH);
                break;
                case 'm':                       //aMarillo
                        digitalWrite(12,LOW);
                        digitalWrite(15,HIGH);
                         digitalWrite(14,HIGH);
                break;
                case 'l':                       //Lila
                        digitalWrite(12,HIGH);
                        digitalWrite(15,HIGH);
                        digitalWrite(14,LOW);
                break;
                case 'c':                       //Cian
                        digitalWrite(12,HIGH);
                        digitalWrite(15,LOW);
                        digitalWrite(14,HIGH);
                 break;
                 case 'b':                      //Blanco
                        digitalWrite(12,HIGH);
                        digitalWrite(15,HIGH);
                         digitalWrite(14,HIGH);
                 break;
                case 'n':                       //Negro
                        digitalWrite(12,LOW);
                        digitalWrite(15,LOW);
                        digitalWrite(14,LOW);
                 break;
                default:
                break;
        }
}
```

```
void initSPIFFS(){                          // Esta función lee el config,h guardado en la memoria no volátil
        if (!SPIFFS.begin()){
                Serial.println("Cannot mount SPIFFS volume...");
        }

        File file = SPIFFS.open("/config.json", "r");// Lee del archivo
        deserializeJson(doc, file);
        file.close();
}
String creaMensaje(String texto, String id, int tipo = 0, int puertaid = 0, String dispositivo = "", String ver =
"") {                                       // Publica un mensaje Json
        StaticJsonDocument<200> doc1;
        doc1["centro_id"] = doc["centro_id"];
        doc1["keebox_id"] = doc["keebox_id"];
         doc1["cliente_id"] = id;
        doc1["tipo_id"] = tipo;
        doc1["kee_id"] = puertaid;
        doc1["response"] = texto;
        doc1["dispositivo"] = dispositivo;
        if (ver != ""){
                doc1["version"] = ver;
         }
        char jsonBuffer[512];
        serializeJson(doc1, jsonBuffer);
        return  jsonBuffer;                 // Imprime en el cliente
}

String dirIp(){                             //Devuelve la dirección IP del dispositivo
        return WiFi.localIP().toString().c_str();
 }

void publicaMensaje(String texto,String id,int tipo = 0,int puertaid = 0,String dispositivo = "",String ver= ""){
                                            // Publica un mensaje Json
        String s = creaMensaje(texto, id, tipo,puertaid,dispositivo,ver);
        int n = s.length();
        char char_array[n + 1];
        strcpy(char_array, s.c_str());
        client.publish(doc["aws_iot_publish_topic"], char_array);
}

void reseteaPlaca(){                        // Fuerza un reset por software
        CambiaColor('r');
        delay(2000);
        ESP.restart();
}

String estadoConexion(){            //Comprueba el estado de la conexión y devuelve un string con el mismo
        if(client.connected())
                return "conectado";
        else
                return "desconectado";
}

int Abre(String message,int rele1 = 0,long duracion1 = 900){ //Lee el mensaje y abre el relé en consecuencia
        duracion = duracion1;
        if (message == "abre"){
```

```
                rele = rele1;
        }else if(message == "abre_principal"){
                rele = 4;
                duracion = 4900;
                //digitalWrite(relayGPIOs[2], HIGH);
        }else if(message == "abre_garaje"){
                rele = 3;
                duracion = 4900;
        }else if(message == "abre_interna"){
                rele = 2;
                duracion = 4900;
        }else if(message == "abre_secundaria"){
                rele = 1;
                duracion = 4900;
        }else{return(404);}

        digitalWrite(relayGPIOs[rele-1], HIGH);
         patata = 1;
        delay(duracion);
        digitalWrite(relayGPIOs[0], LOW);           //Se cierran todo los relés disponibles por seguridad
        digitalWrite(relayGPIOs[1], LOW);
        digitalWrite(relayGPIOs[2], LOW);
        digitalWrite(relayGPIOs[3], LOW);
        digitalWrite(relayGPIOs[4], LOW);
        digitalWrite(relayGPIOs[5], LOW);
        digitalWrite(relayGPIOs[6], LOW);
        digitalWrite(relayGPIOs[7], LOW);

         VarOPt[rele-1] = 0;

        if(digitalRead(lockGPIOs[rele -1]) == HIGH && message == "abre"){
                tiempos[rele-1] = millis();
                Prev[rele-1] = 1;
                return 200;
        }else if(message != "abre"){
                return 200 + rele;
        }else{
                Prev[rele-1] = 2;
                return 400;
        }
}

void makeRandomMac(uint8_t mac[6]) {                    //Genera una dirección Mac al azar
        for (size_t i = 0; i < 6; ++i) {
                mac[i] = random(256);
        }
        mac[0] = mac[0] & ~0x01;
}

void messageHandler(String &topic, String &payload){    // Controla las ordenes que le llegan a la placa
        int devuelta = 400;
        StaticJsonDocument<200> doc1;
        deserializeJson(doc1, payload);
        String message = doc1["message"];
        int puerta = doc1["puerta"];
        String dispositivo = doc1["dispositivo"];
```

```
        String id = doc1["id"];

        if (message == "ip"){
                publicaMensaje(dirIp(),id,0,0,"",version);
        }
        if (message == "mac"){
                publicaMensaje(WiFi.macAddress(),id,0,0,"",version);
        }
        if (message == "change_mac"){
                MC = 1;
                makeRandomMac(mac0);
                esp_wifi_set_mac(WIFI_IF_STA, const_cast<uint8_t*>(mac0));
        }

        if (message == "reset"){
                publicaMensaje("Reseteando Keebox", "", 0, 0, "", version);
                reseteaPlaca();
        }

        devuelta = Abre(message,puerta);
        if(devuelta >= 200 && devuelta < 400){
                int valord = devuelta -200;
                if(valord == 0) {
                        publicaMensaje("Puerta abierta ", id,1,puerta, dispositivo);
                        patata = 0;
                }
                else{
                        publicaMensaje("Puerta especial ", id,1,valord, dispositivo);
                        patata = 0;
                }
        }else if(devuelta == 400){
                publicaMensaje("Error abriendo puerta ", id,0,puerta, dispositivo);
                 patata = 0;
        }
        if(message == "estado_puertas"){                        //Envía el estado de las puertas para el formato
                kerong 1 es abierto o no conectado y 0 es cerrado.
                 String estado_puerta = "";
                int A;
                for(int t=0; t<=7; t++){
                        state[t] = digitalRead(lockGPIOs[t]);
                        A = t+1;
                         estado_puerta += "R"+ String(A) + ":" + String(state[t]) + ",";
                }
                 publicaMensaje(estado_puerta, id,6,puerta, dispositivo);
        }
}

void publishMessage(){                          // Publica mensajes en amazon
        StaticJsonDocument<200> doc1;
        doc1["time"] = millis();
        char jsonBuffer[512];
        serializeJson(doc1, jsonBuffer); // print to client
        client.publish(doc["aws_iot_publish_topic"], jsonBuffer);
}
```

```
void ConexionWifi(){
	WiFi.mode(WIFI_STA);
	WiFi.begin(doc["ssid"].as<const char *>(), doc["password"].as<const char *>());
	 Serial.println(WiFi.localIP());
	delay(1000);
	if(WiFi.status() != WL_CONNECTED && MC == 0){
		esp_wifi_set_mac(WIFI_IF_STA, const_cast<uint8_t*>(mac));
	}
	if(WiFi.status() != WL_CONNECTED && MC == 1){
		esp_wifi_set_mac(WIFI_IF_STA, const_cast<uint8_t*>(mac0));
	}
	/*if(doc["ip"] != nullptr) {                                    //Esto para ip fija, no es recomendable su
uso ya que ralentiza la conexión y genera cuellos de botella
		int laip[4];
		copyArray(doc["ip"], laip);
		IPAddress gateway = WiFi.gatewayIP();
		IPAddress local_IP(laip[0], laip[1], laip[2], laip[3]);
		IPAddress subnet = WiFi.subnetMask();
		 IPAddress primaryDNS = WiFi.dnsIP(0);   // optional
		IPAddress secondaryDNS = WiFi.dnsIP(1); // optional
		Serial.println(WiFi.localIP());
		if (!WiFi.config(local_IP, gateway, subnet, primaryDNS, secondaryDNS)){
			Serial.println("STA Failed to configure");
		}
	}*/
}

void WiFiStationConnected(WiFiEvent_t event, WiFiEventInfo_t info) {
	Serial.println("Connected to AP successfully!");
	CambiaColor('m');
	MC = 0;
}

void WiFiGotIP(WiFiEvent_t event, WiFiEventInfo_t info){
	Serial.println("WiFi conectada");
	 Serial.println("Dirección IP: ");
	Serial.println(WiFi.localIP());
	makeRandomMac(mac);
}

void WiFiStationDisconnected(WiFiEvent_t event, WiFiEventInfo_t info){
	Serial.println("Disconnected from WiFi access point");
	 Serial.print("WiFi lost connection. Reason: ");
	 Serial.println(info.wifi_sta_disconnected.reason);
	CambiaColor('r');
	Serial.println("Trying to Reconnect");
	ConexionWifi();
}

void connectAWS(){ // Nos conecta con amazon, si hay problemas revisar la informacion en el config o los
certificados en el
	while (WiFi.status() != WL_CONNECTED) {
		ConexionWifi();
	}
			// Configure WiFiClientSecure to use the AWS IoT device credentials
	net.setCACert(doc["aws_cert_ca"].as<const char *>());
```

```
	net.setCertificate(doc["aws_cert_crt"].as<const char *>());
	net.setPrivateKey(doc["aws_cert_private"].as<const char *>());
			// Connect to the MQTT broker on the AWS endpoint we defined earlier
	client.begin(doc["aws_iot_endpoint"].as<const char *>(), 8883, net);
			// Create a message handler
	client.onMessage(messageHandler);
	client.setKeepAlive(40);
	String cen = doc["centro_id"];
	String keebox_ide = doc["keebox_id"];
	String the_ip = WiFi.localIP().toString().c_str();
	String payload = "{\"centro_id\": \"" + cen + "\",\"keebox_id\": \"" + keebox_ide + "\",\"ip\":\"" +
the_ip + "\"}";
	char envio[256];
	strcpy(envio, payload.c_str());
	client.setWill("keebox/off", envio);
	Serial.print("Connecting to AWS IOT");
	while (!client.connect(doc["thingname"])){
		Serial.print(".");
		delay(100);
	}
	if (!client.connected()){
		Serial.println("AWS IoT Timeout!");
		 return;
	}
			// Subscribe to a topic
	client.subscribe(doc["aws_iot_suscribe_topic"].as<String>());
	client.subscribe(doc["aws_iot_suscribe_topic_global"].as<String>());
	publicaMensaje("conectado", "", 5, 0, dirIp(), version);
	Serial.println("AWS IoT Connected!");
	CambiaColor('a');
}

void IRAM_ATTR ISR_0() {                                    //Esta y las otras 7 funciones establecen que
deben hacer cuando salta una interrupcion fisica desde los candados
	if (millis() - startTime > timeThreshold){
		delayMicroseconds(2000);
		state[0] = digitalRead(lockGPIOs[0]);
		if (Prev[0] == 1){
			VarOPt[0] = 2;
			abreP[0] = millis();
			 //Serial.println("Puerta 1 Abierta");
		}else if(Prev[0] == 2){
			VarOPt[0] = 1;
			abreP[0] = millis();
			//Serial.println("Puerta 1 Cerrada");
		}else{
			 VarOPt[0] = 0;
			//Serial.println("Entra en isr pero no pasa");
		}
		startTime = millis();
	}
}

void IRAM_ATTR ISR_1() {
	if (millis() - startTime > timeThreshold){
		delayMicroseconds(2000);
```

```
                state[1] = digitalRead(lockGPIOs[1]);
                if (Prev[1] == 1){
                        VarOPt[1] = 2;
                        abreP[1] = millis();
                }else if(Prev[1] == 2){
                        VarOPt[1] = 1;
                        abreP[1] = millis();
                }else{
                        VarOPt[1] = 0;
                }
                startTime = millis();
  }
}

void IRAM_ATTR ISR_2() {
        if (millis() - startTime > timeThreshold){
                delayMicroseconds(2000);
                state[2] = digitalRead(lockGPIOs[2]);
                if (Prev[2] == 1){
                        VarOPt[2] = 2;
                        abreP[2] = millis();
                }else if(Prev[2] == 2){
                        VarOPt[2] = 1;
                        abreP[2] = millis();
                }else{
                        VarOPt[2] = 0;
                }
                startTime = millis();
        }
}

void IRAM_ATTR ISR_3() {
        if (millis() - startTime > timeThreshold){
                delayMicroseconds(2000);
                state[3] = digitalRead(lockGPIOs[3]);
                if (Prev[3] == 1){
                        VarOPt[3] = 2;
                        abreP[3] = millis();
                }else if(Prev[3] == 2){
                        VarOPt[3] = 1;
                        abreP[3] = millis();
                }else{
                        VarOPt[3] = 0;
                }
                startTime = millis();
        }
}

void IRAM_ATTR ISR_4() {
        if (millis() - startTime > timeThreshold){
                delayMicroseconds(2000);
                state[4] = digitalRead(lockGPIOs[4]);
                if (Prev[4] == 1){
                        VarOPt[4] = 2;
                        abreP[4] = millis();
                }else if(Prev[4] == 2){
```

```
                    VarOPt[4] = 1;
                    abreP[4] = millis();
                }else{
                    VarOPt[4] = 0;
                }
                startTime = millis();
        }
}

void IRAM_ATTR ISR_5() {
        if (millis() - startTime > timeThreshold){
                delayMicroseconds(2000);
                state[5] = digitalRead(lockGPIOs[5]);
                if (Prev[5] == 1){
                    VarOPt[5] = 2;
                    abreP[5] = millis();
                }else if(Prev[5] == 2){
                    VarOPt[5] = 1;
                    abreP[5] = millis();
                }else{
                    VarOPt[5] = 0;
                }
                startTime = millis();
         }
}

void IRAM_ATTR ISR_6() {
        if (millis() - startTime > timeThreshold){
                delayMicroseconds(2000);
                state[6] = digitalRead(lockGPIOs[6]);
                if (Prev[6] == 1){
                    VarOPt[6] = 2;
                    abreP[6] = millis();
                }else if(Prev[6] == 2){
                    VarOPt[6] = 1;
                    abreP[6] = millis();
                }else{
                    VarOPt[6] = 0;
                }
                startTime = millis();
        }
}

void IRAM_ATTR ISR_7() {
        if (millis() - startTime > timeThreshold){
                delayMicroseconds(2000);
                state[7] = digitalRead(lockGPIOs[7]);
                if (Prev[7] == 1){
                    VarOPt[7] = 2;
                     abreP[7] = millis();
                }else if(Prev[7] == 2){
                    VarOPt[7] = 1;
                    abreP[7] = millis();
                }else{
                    VarOPt[7] = 0;
                }
```

```
                startTime = millis();
        }
}

void loop2 (void *pvParameters){  // Loop del Segundo nucleo, aqui se procesa la informacion de las interrupciones para que no entre en conflicto con la conexión
        for(;;){
                for(int t=0; t<=7; t++){ //Comprueba si ha saltado la interrupción y publica un mensaje
                        if(patata == 0){
                                if(timer[t] > 0 && millis() - timer[t] > 1000){
                                        if(digitalRead(lockGPIOs[t]) == HIGH){
                                                tiempos[t] = millis();
                                                Prev[t] = 2;
                                                abreP[t] = 0;
                                                ReleX[t] = 1;
                                                timer[t]=0;
                                        }else{timer[t]=0;}
                                }
                                if(abreP[t] > 0 && (millis() - abreP[t]) > 500){
                                        if(digitalRead(lockGPIOs[t]) == LOW){
                                                Prev[t] = 1;
                                                abreP[t] = 0;
                                        }else if(digitalRead(lockGPIOs[t]) == HIGH){
                                                Prev[t] = 2;
                                                abreP[t] = 0;
                                                ReleX[t] = 1;
                                        }
                                }
                                if(VarOPt[t] > 0 && abreP[t] == 0){
                                        if(Prev[t] == 2 && VarOPt[t] == 2){
                                                publicaMensaje("Puerta abierta","",1,t+1,"");
                                                VarOPt[t] = 0;
                                                tiempos[t] = millis();
                                        }else if(Prev[t] == 1 && ReleX[t] == Prev[t] && VarOPt[t] == 1){
                                                publicaMensaje("Puerta cerrada","",2,t+1,"");
                                                VarOPt[t] = 0;
                                                ReleX[t] = 0;
                                                tiempos[t] = 0;
                                                timer[t] = millis();
                                        }
                                }
                        }
                        if(tiempos[t] > 0){  //Comprueba el estado de los candados por si alguno se hubiera quedado abierto demasiado tiempo
                                if(millis() - tiempos[t] >= control ){
                                        if(digitalRead(lockGPIOs[t]) == LOW){
                                                tiempos[t] = 0;
                                        }else{
                                                publicaMensaje("Aviso tiempo","",401,t+1,"","");
                                                tiempos[t] = millis();
                                        }
                                }
                        }
                }
        }
 }
```

```
void setup(){
	Serial.begin(115200); // Serial port for debugging purposes

	xTaskCreate(loop2, "mi_tarea", 10000, NULL, 1, NULL);

	initSPIFFS();
	String centro_id = doc["centro_id"];
	esp32fota.checkURL = "https://___._________.com/fota?id="+ centro_id;  // Donde mira la placa
por si se tiene que actualizar

	esp_task_wdt_init(WDT_TIMEOUT, true); // enable panic so ESP32 restarts
	esp_task_wdt_add(NULL);               // add current thread to WDT watch

	pinMode(32, INPUT);
	pinMode(33, INPUT);
	pinMode(34, INPUT);
	pinMode(35, INPUT);

	attachInterrupt(lockGPIOs[0], ISR_0, CHANGE);    //Modo de funcionamiento de las interrupciones
	attachInterrupt(lockGPIOs[1], ISR_1, CHANGE);
	attachInterrupt(lockGPIOs[2], ISR_2, CHANGE);
	attachInterrupt(lockGPIOs[3], ISR_3, CHANGE);
	attachInterrupt(lockGPIOs[4], ISR_4, CHANGE);
	attachInterrupt(lockGPIOs[5], ISR_5, CHANGE);
	attachInterrupt(lockGPIOs[6], ISR_6, CHANGE);
	attachInterrupt(lockGPIOs[7], ISR_7, CHANGE);

	Serial.println("MAC Adress:");
	Serial.println(WiFi.macAddress());

	 makeRandomMac(mac);

	pinMode(12,OUTPUT);                              //Led para control errores Blue
	pinMode(15,OUTPUT);                              //Led para control errores Red
	pinMode(14,OUTPUT);                              //Led para control errores Green

//Establece todos los relés en LOW cuando el programa empieza si esta en NO el relé esta apagado
	for (int i = 1; i <= NUM_RELAYS; i++)// Establece los estados de cada puerta al iniciarse la placa
		pinMode(relayGPIOs[i - 1], OUTPUT);
		digitalWrite(relayGPIOs[i - 1], LOW);
		if (digitalRead(lockGPIOs[i - 1]) == LOW){
			Prev[i - 1] = 1;
			ReleX[i-1] = 0;
		}
		else{
			Prev[i - 1] = 2;
			 ReleX[i-1] = 1;
		}
	}

	WiFi.onEvent(WiFiStationConnected,  ARDUINO_EVENT_WIFI_STA_CONNECTED);
	WiFi.onEvent(WiFiGotIP, ARDUINO_EVENT_WIFI_STA_GOT_IP);
	WiFi.onEvent(WiFiStationDisconnected, ARDUINO_EVENT_WIFI_STA_DISCONNECTED);

	ConexionWifi();
```

```
	server.on("/ip", HTTP_GET, [](AsyncWebServerRequest *request) {
		request->send(200, "application/json; charset=UTF-8", creaMensaje(dirIp(), "", 1));
	});

	server.on("/desconecta", HTTP_GET, [](AsyncWebServerRequest *request) {
		 intento_aws = 10;
		client.disconnect();
		request->send(200, "application/json; charset=UTF-8", creaMensaje("desconectado de amazon", "", 1));
		CambiaColor('l');

	});

	server.on("/reset", HTTP_GET, [](AsyncWebServerRequest *request) {
		request->send(200, "application/json; charset=UTF-8", creaMensaje("Reseteando Keebox", "", 1));
		reseteaPlaca();
	});

	server.on("/estado", HTTP_GET, [](AsyncWebServerRequest *request) {
		request->send(200, "application/json; charset=UTF-8", creaMensaje(estadoConexion(), "", 1));
	});
  // Send a GET request to <ESP_IP>/update?relay=<inputMessage>&state=<inputMessage2>
	http://192.168.1.106/llama?relay=1&state=1
	server.on("/llama", HTTP_GET, [](AsyncWebServerRequest *request) { // Para hacer llamadas desde web en lugar de la app
		String inputMessage;
		String inputParam;
		String inputMessage2;
		String inputParam2;
		String inputMessage3;
		String inputParam3;
		int devuelta = 400;
		// GET input1 value on <ESP_IP>/update?relay=<inputMessage>
		if (request->hasParam(PARAM_INPUT_1) & request->hasParam(PARAM_INPUT_2) & request->hasParam(PARAM_INPUT_3)) {
			inputMessage = request->getParam(PARAM_INPUT_1)->value();
			inputParam = PARAM_INPUT_1;
			inputMessage2 = request->getParam(PARAM_INPUT_2)->value();
			inputParam2 = PARAM_INPUT_2;
			inputMessage3 = request->getParam(PARAM_INPUT_3)->value();
			inputParam3 = PARAM_INPUT_3;
			devuelta = Abre(inputMessage3,inputMessage.toInt());

		}else{
			inputMessage = "No message sent";
			inputParam = "none";
		}
		if(devuelta == 200)
			request->send(200, "application/json; charset=UTF-8", creaMensaje("puerta abierta", "interno", 1));
		else
			request->send(400, "application/json; charset=UTF-8", creaMensaje("error", "interno", 1));
```

```
        });

        server.begin();
        connectAWS();
        bool updatedNeeded = esp32fota.execHTTPcheck(); // Comprueba si debe actualizarse o no
         if (updatedNeeded) {
                esp32fota.execOTA();
        }

} // end setup

void loop(void) {           //Main principal, se ejecuta infinitamente comprobando las conexiones pertinentes.
        client.loop();
        if(!client.connected()){
                CambiaColor('m');
                if(intento_aws < 10){
                        intento_aws++;
                        connectAWS();
                        //Serial.println(intento_aws);
                }

        }

        if (WiFi.status() != WL_CONNECTED){             //Si está desconectado enciende el led rojo
                CambiaColor('r');
        }

         esp_task_wdt_reset();                  //Si la placa no pasa por aquí en el tiempo se reinicia

}
```