UNIVERSITAT POLITÈCNICA DE VALÈNCIA

CAMPUS D'ALCOI

Proyecto de la instalación domótica de una vivienda unifamiliar con raspberry pi situada en el término municipal de Onteniente (Valencia)

MEMORIA PRESENTADA POR:

Sergio Bodí Pastor

GRADO DE *[INGENIERÍA ELÉCTRICA]*

Convocatoria de defensa: *[septiembre 2020]*

Resumen

Este proyecto de fin de grado (TFG) del grado en ingeniería eléctrica trata sobre la automatización de una casa domótica mediante el protocolo MQTT, utilizando una Raspberry Pi como broker MQTT y una serie de microcontroladores llamados ESP32 distribuidos por la casa que actuarían como clientes del anteriormente mencionado protocolo.

En la Raspberry Pi se ejecuta en segundo plano el bróker MQTT Mosquitto usado para la comunicación con los diferentes microcontroladores donde se adquiere las señales de los diferentes sensores conectados a estos y Node-RED para enlazar y programar las acciones.

Palabras clave

Las palabras clave son las siguientes: MQTT, ESP32, Raspberry Pi, Node-RED, IoT, Microcontrolador.

Abstract

This thesis of the bachelor degree of electrical engineering is about the home automation based in the MQTT protocol, using a Raspberry Pi as a broker MQTT and a set of microcontrollers called ESP32, distributed around the house, as clients of the previously called MQTT protocol.

In the Raspberry Pi is executed on the background the MQTT broker Mosquitto used for the communication with the microcontrollers, where the microcontrollers get the signals of the diferents sensors connected to them and Node-RED to link and program the actions.

Key words

The key words are the following ones: MQTT, ESP32, Raspberry Pi, Node-RED, IoT, Microcontroller.

Contenido

1. Introducción

1.1 Antecedentes y justificación del proyecto

El internet de las cosas (Internet of Things o IOT) es un concepto que se refiere a una interconexión digital de objetos cotidianos con internet. Es, en definitiva, la conexión de internet más con objetos que con personas.
Constituye un cambio radical en la calidad de vida de las personas en la sociedad, ofrece una gran cantidad de nuevas oportunidades de acceso a datos, servicios específicos en la educación, seguridad, asistencia sanitaria y en el transporte, entre otros campos.
El término internet de las cosas se usa con una denotación de conexión avanzada de dispositivos, sistemas y servicios que van más allá del tradicional M2M (máquina a máquina) y abarca una amplia variedad de protocolos, dominios y aplicaciones.

La capacidad de conectar dispositivos embebidos con capacidades limitadas de CPU, memoria y energía significa que IoT puede tener aplicaciones en casi cualquier área. Estos sistemas podrían encargarse de recolectar información en diferentes entornos: desde ecosistemas naturales hasta edificios y fábricas, por lo que podrían utilizarse para monitoreo ambiental y planeamiento urbanístico.

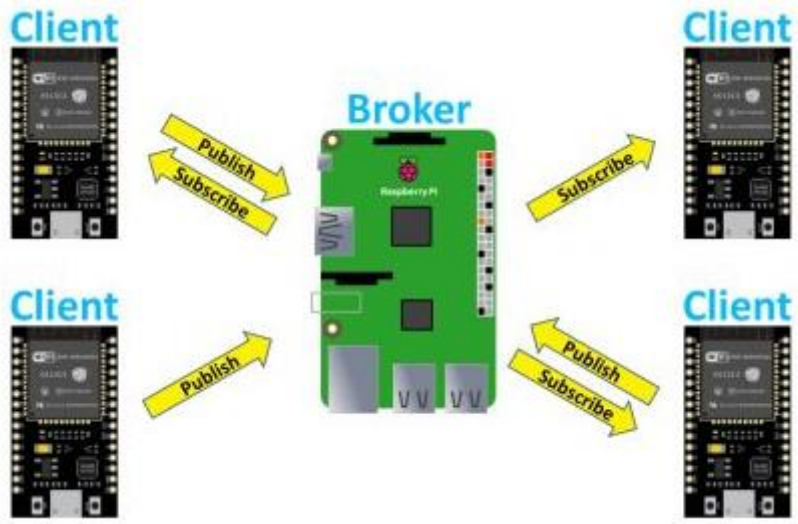
El sistema tiene una arquitectura orientada a eventos, construida de abajo a arriba (basada en el contexto de procesos y operaciones, en tiempo real) y tendrá en considera cualquier nivel adicional. Por lo tanto, el modelo orientado a eventos y el enfoque funcional coexistirán con nuevos modelos capaces de tratar excepciones y evolución insólita de procesos.

https://es.wikipedia.org/wiki/Internet_de_las_cosas

Durante el desarrollo de la tecnología IoT has surgido una serie de protocolos de comunicación, pero en este proyecto nos vamos a centran solamente en una, MQTT (Message Queue Telemetry Transport).

1.2 Objetivos del proyecto

El propósito del presente proyecto es el desarrollo de una instalación domótica básica.
Se pretende conectar una serie de sensores y dispositivos a un microcontrolador llamado ESP32, que tiene WiFi integrado.
Este microcontrolador envía la información de los sensores al bróker (Raspberry Pi) mediante el protocolo MQTT estableciendo una comunicación M2M (machine-to-machine).
En la Raspberry Pi mediante Node-RED que es una herramienta de desarrollo basada en flujo para programación visual, se automatizan las acciones dependiendo de la información proporcionada por los sensores y mediante la comunicación MQTT se comunica con los microcontroladores para que estos actúen sobre sus salidas, a las que hay conectado diferentes circuitos para poder actuar sobre otras cargas como iluminación y los motores de las persianas.



2. Aspectos previos

2.1 MQTT

2.1.1 Introducción a MQTT

Para dispositivos de Internet de las Cosas (IoT) es necesaria la conexión a Internet. La conexión a Internet permite que los dispositivos trabajen entre ellos y con servicios backend.
El protocolo de red subyacente de internet es TCP/IP.
MQTT (Message Queue Telemetry Transport), que está construido sobre la pila de TCP/IP, se ha convertido en el estándar para las comunicaciones de IoT.

Originariamente, MQTT fue inventado y desarrollado por IBM a finales de los 90.
Su aplicación original era conectar sensores de los oleoductos con satélites. Tal como sugiere su nombre, es un protocolo de mensajería que soporta la comunicación asíncrona entre las partes. Un protocolo de mensajería asíncrona disocia al emisor y al receptor de los mensajes tanto en espacio como en tiempo, y, por lo tanto, es escalable en entornos de red no confiables. A pesar de su nombre, no tiene nada que ver con las colas de mensajería, y, en cambio, utiliza un modelo de publicación suscripción. A finales del 2014, se convirtió oficialmente en un estándar abierto de OASIS, y se soporta en lenguajes de programación populares mediante la utilización de varias implementaciones de código abierto.

2.1.2 Por qué MQTT

MQTT es un protocolo de red liviano y flexible que logra el equilibrio adecuado para los desarrolladores de IoT:

- El protocolo liviano le permite implementarse en hardware de dispositivos altamente limitados y en redes con ancho de banda de alta latencia/limitado.
- Su flexibilidad hace que pueda soportar varios escenarios de aplicaciones para dispositivos y servicios de IoT.

La mayor parte de los desarrolladores ya están familiarizados con los servicios web de HTTP. Así, ¿por qué no hacer que los dispositivos de IoT se conecten con servicios web? El dispositivo podría enviar sus datos como una solicitud HTTP y recibir actualizaciones del sistema como la respuesta de HTTP. Este patrón de solicitud y respuesta tiene algunas limitaciones graves:

- HTTP es un protocolo sincronizado. El cliente espera a que el servidor responda. Los navegadores web tienen este requisito, que trae consigo el costo de una mala escalabilidad. En el mundo de IoT, el gran número de dispositivos y muy probablemente una red no confiable, o de alta latencia, han hecho que la comunicación sincronizada sea problemática. Un protocolo de mensajería asíncrona es mucho más adecuado para las aplicaciones de IoT. Los sensores pueden enviar lecturas y dejar que la red descubra la ruta y el momento óptimos para la entrega de servicios de destino.
- El HTTP tiene una dirección. El cliente debe iniciar una conexión. En una aplicación de IoT los dispositivos y los sensores son normalmente los clientes, lo que significa que no pueden recibir comandos de la red de forma pasiva.
- HTTP es un protocolo 1-1. El cliente hace una solicitud y el servidor responde. Transmitir un mensaje a todos los dispositivos de la red, lo que es un caso de uso habitual en las aplicaciones IoT, es algo difícil y caro.
- HTTP es un protocolo pesado con muchas cabeceras y reglas. No es adecuado para redes congestionadas.

Por las razones anteriores, la mayor parte de los sistemas escalables de alto rendimiento utilizan un bus de mensajería asíncrona, en vez de servicios web, para intercambiar datos internamente. De hecho, el protocolo de mensajería que se utiliza más habitualmente en los sistemas de middleware empresarial se llama AMQP (Advanced Message Queuing Protocol). Sin embargo, en entornos de alto rendimiento, el poder de computación y la latencia de la red no son generalmente una preocupación. AMQP está diseñado para la confiabilidad y la interoperabilidad en aplicaciones empresariales. Tiene un amplio conjunto de funciones, pero no es adecuado para las aplicaciones de IoT con recursos limitados.

Además de AMQP, hay otros protocolos de mensajería populares. Por ejemplo, el XMPP (Extensible Messaging and Presence Protocol) es un protocolo de mensajería instantánea (IM) peer-to-peer. Tiene muchas funciones que soportan casos de uso de IM, como los adjuntos de presencia y de medios de comunicación. Comparado con MQTT, requiere muchos más recursos tanto en el dispositivo como en la red.

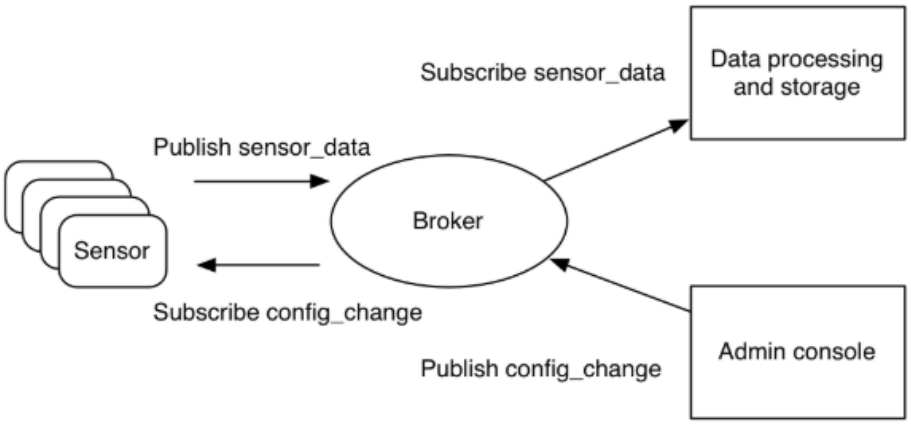
Así que, ¿qué es lo que hace que MQTT sea tan liviano y flexible? Una de las principales funciones del protocolo MQTT es su modelo de publicación y suscripción. Como con todos los protocolos de mensajería, éste desacopla los datos del consumidor y del publicador.

El protocolo MQTT define los tipos de entidades en la red: un intermediario de mensajes y un número de clientes. El intermediario es un servidor que recibe todos los mensajes de los clientes y luego los redirige a clientes de destinos relevantes. Un cliente es cualquier cosa que pueda interactuar con el intermediario para enviar o recibir mensajes. Un cliente puede ser un sensor de IoT en el campo o una aplicación del centro de datos que procesa datos IoT.

1. El cliente se conecta con el intermediario. Se puede suscribir a cualquier “tema” de mensajes de intermediario. Esta conexión puede ser una conexión TCP/IP simple o una conexión TLS cifrada para mensajes confidenciales.
2. El cliente publica el mensaje, sobre un tema, enviando el mensaje y el tema al intermediario.
3. Después, el intermediario redirige el mensaje a todos los clientes que están suscritos a ese “tema”.

Ya que os mensajes MQTT están organizados por temas, el desarrollador de aplicaciones tiene la flexibilidad de especificar que ciertos clientes solo puedan interactuar con determinados mensajes. Por ejemplo, los sensores publicarán sus lecturas sobre el tema “sensor_data” y se suscribirá al tema “config-change”. Las aplicaciones de procesamiento de datos que guardan datos del sensor en una base de datos backend se suscribirán al tema “sensor_data”.
Una aplicación de consola de administración puede recibir los comandos del administrador del sistema para ajustar las configuraciones de los sensores, como la sensibilidad y la frecuencia de la muestra, y publicar esos cambios en el tema “config_change”

El modelo de publicación y suscripción de MQTT para los sensores de IoT



Al mismo tiempo, MQTT es liviano. Tiene una cabecera simple para especificar el tipo de mensaje, un tema basado en texto y, a continuación, una carga útil binaria y arbitraria. La aplicación puede utilizar cualquier formato de datos para la carga útil, como: JSON, XML, cifrado binario o Base64, siempre que los clientes destino puedan analizar la carga útil.

https://developer.ibm.com/es/articles/iot-mqtt-why-good-for-iot/

2.1.3 Cliente MQTT

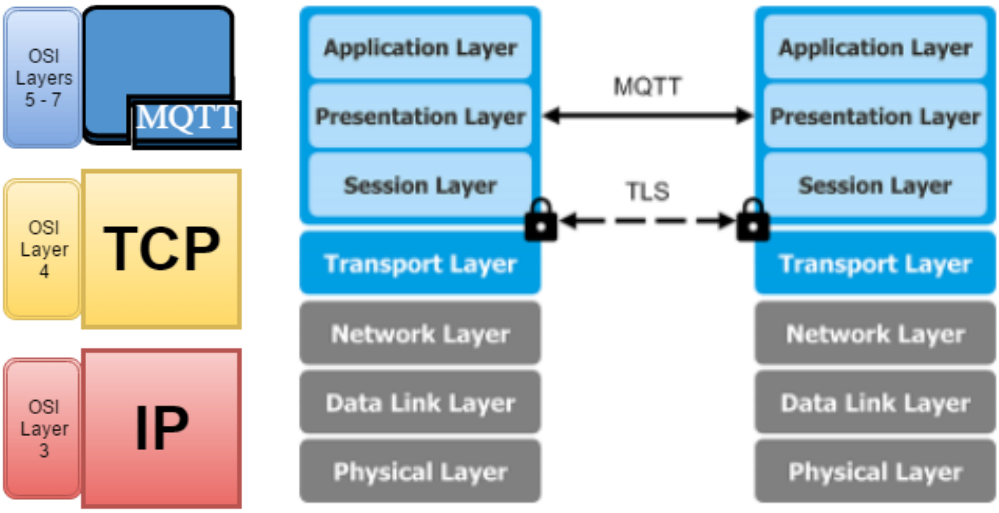
Un cliente MQTT puede ser cualquier dispositivo que se conecte a un bróker MQTT a través de internet, desde uno con recursos muy limitados como un microcontrolador hasta un ordenador con prestaciones más que suficientes.
La principal ventaja de los clientes MQTT es que es muy sencillo implementarlos, permitiendo su uso en dispositivos con pocos recursos como es el caso del microprocesador que usamos en el proyecto, el ESP32.
Además, las librerías de clientes MQTT tienen una amplia variedad de lenguajes para su uso (C++, Java, Arduino, etc.), lo que permite que dicho protocolo sea muy flexible.

2.1.4 Bróker MQTT

El bróker es el elemento principal del modelo de publicación y suscripción, al cual se conectan los clientes para enviar y recibir los mensajes. El bróker recibe los mensajes, los filtra en función del "topic", y los envía a los clientes que estén suscritos a dicho "topic".
También se encarga de autentificar a los clientes que tratan de conectarse a él.

2.1.5 Establecimiento de la conexión

El protocolo MQTT se localiza en las capas superiores de OSI, y normalmente se apoya en TCP/IP. Esto significa que los participantes de una aplicación MQTT deben tener una pila TCP/IP.



Como MQTT depende del protocolo de transporte TCP. Por defecto, las conexiones TCP no usan una comunicación encriptada. Para encriptar toda la comunicación MQTT, varios brokers MQTT permiten el uso de TLS.

Para iniciar la comunicación, primero, el cliente se conecta con el intermediario enviando un mensaje CONNECT. El mensaje CONNECT pide establecer una conexión desde el cliente hacia el intermediario. El mensaje CONNECT tiene los siguientes parámetros de contenido.
Parámetros del mensaje CONNECT

- cleanSession. Este identificador especifica si la conexión es persistente o no. Una sesión persistente almacena todos los mensajes de la suscripción y los que potencialmente se han perdido (dependiendo de QoS) en el intermediario. Nombre de usuario, la autentificación del intermediario y las credenciales de autorización.
- lastWillTopic. Cuando una conexión se cae de forma inesperada, el intermediario publica automáticamente un mensaje de "última voluntad" en un tema.
- lastWillQos. El mensaje de "última voluntad" en QoS.
- lasWillMessage. El propio mensaje de "última voluntad".
- keepAlive. Este es el intervalo de tiempo para que el cliente compruebe la disponibilidad del intermediario para mantener la conexión viva.

El cliente recibirá un mensaje CONNACK del intermediario. El mensaje CONNACK tiene los siguientes parámetros de contenido.

Descripción del parámetro:

- sessionPresent. Esto indica si la conexión ya tiene una sesión persistente. Esto significa que la conexión ya tiene temas a los que se ha suscrito, y que recibirá la entrega de los mensajes que faltan.
- returnCode0. Indica éxito. Otros valores identifican la causa de la falla.

Después de que se establece una conexión, el cliente puede enviar al intermediario uno o más mensajes SUBSCRIBE para indicar que recibirá mensajes del intermediario para determinados temas. Esos mensajes pueden tener una o varias repeticiones en los siguientes parámetros.
Parámetros del mensaje SUBSCRIBE

- qos. El indicador qos (calidad de servicio, o QoS) indica la consistencia con la que los mensajes de este tema se tienen que entregar a los clientes.

  -Valor 0: No confiable, el mensaje se entrega como mucho una vez, si el cliente no está disponible en ese momento se perderá el mensaje.
  -Valor 1: el mensaje se debería entregar al menos una vez.
  -Valor 2: el mensaje se debería entregar exactamente una vez.

- topic. Un tema al que suscribirse. Un tema puede tener varios niveles separados por el carácter de barra diagonal. Por ejemplo: "dw/demo" y "ibm/bluemix/mqtt" son temas válidos.

Después de que un cliente se haya suscrito correctamente a un tema, el intermediario devuelve un mensaje SUBACK con uno o más parámetros “returnCode”.
Parámetros del mensaje SUBACK

- returnCode. En el comando SUBSCRIBE existe un código de retorno para cada uno de los temas. Los valores de retorno son los siguientes.

  -Valores 0 – 2: éxito con el nivel de QoS correspondiente.
  -Valor 128: falla

Correspondiente al mensaje SUBSCRIBE, también puede dejar la suscripción (UNSUBSCRIBE) de un tema o de varios.
Parámetros del mensaje UNSUBSCRIBE

- topic. Este parámetro se puede repetir para varios temas.

El cliente puede enviar mensajes PUBLISH al intermediario. El mensaje contiene un tema y una carga útil de datos. Después, el intermediario redirige el mensaje a todos los clientes que están suscritos a ese tema.

Parámetros del mensaje PUBLISH

- topicName. El tema en que se publica el mensaje.
- qos. La calidad del nivel de servicio de la entrega del mensaje.
- retainFlag. Este indicador indica si el intermediario retendrá el mensaje como el último mensaje conocido para este tema.
- payload. Los datos reales que hay en el mensaje. Puede ser una cadena de texto o un boleano.

2.1.6 Topic

El bróker MQTT utiliza los temas o “topics” para decidir qué cliente recibe qué mensaje. También existen los SYS-topics, que son temas especiales que revelan información sobre el bróker.
En MQTT un tema se refiere a la cadena de texto UTF-8 que el bróker usa para filtrar los mensajes para cada cliente conectado. El tema consiste en uno o más niveles. Cada nivel está separado por una barra inclinada (separador de niveles).

home/cocina/temperatura

<u>comodines</u>

Cuando un cliente se suscribe a un tema, este se puede suscribir a ese tema exactamente del mensaje publicado o se pueden usar comodines para suscribir a múltiples temas simultáneamente.

Hay dos tipos diferentes de comodines: un nivel o multinivel.

1. Un nivel: +

   home/+/temperatura

   esto significa que suscripción anterior podría coincidir con, por ejemplo:

   home/cocina/temperatura
   home/comedor/temperatura

2. Multinivel: #

   home/habitacion1/#

   esto significa que la suscripción anterior coincide con los todos los temas de la habitación1 como, por ejemplo:

   home/habitacion1/Luz
   home/habitacion1/Pul
   home/habitacion1/temperatura

   entre otros.

2.2 Mosquitto

Eclipse Mosquitto es un broker de mensajes open source (licencia EPL/EDL) que implementa el protocolo MQTT versiones 5.0, 3.1.1 y 3.1. Mosquitto es ligero y adecuado para todos los dispositivos, desde SBC (Single Board Computer) como la Raspberry pi hasta servidores.

El proyecto mosquitto también proporciona una librería en C para implementar clientes MQTT.

3. Software

3.1 Raspberry Pi OS

Raspberry Pi OS (anteriormente llamado Raspbian) es una distribución del sistema operativo GNU/Linux basado en Debian, y por lo tanto libre para la SBC Raspberry Pi, orientado a la enseñanza de informática.

Destaca el menú "raspi-config" que permite configurar el sistema operativo sin tener que modificar archivos de configuración manualmente. Entre sus funciones, permite expandir la partición root para que ocupe toda la tarjeta de memoria, configurar el teclado, aplicar overclocks, etc.

https://es.wikipedia.org/wiki/Raspberry_Pi_OS

En cuanto a las opciones de descarga se encuentran:

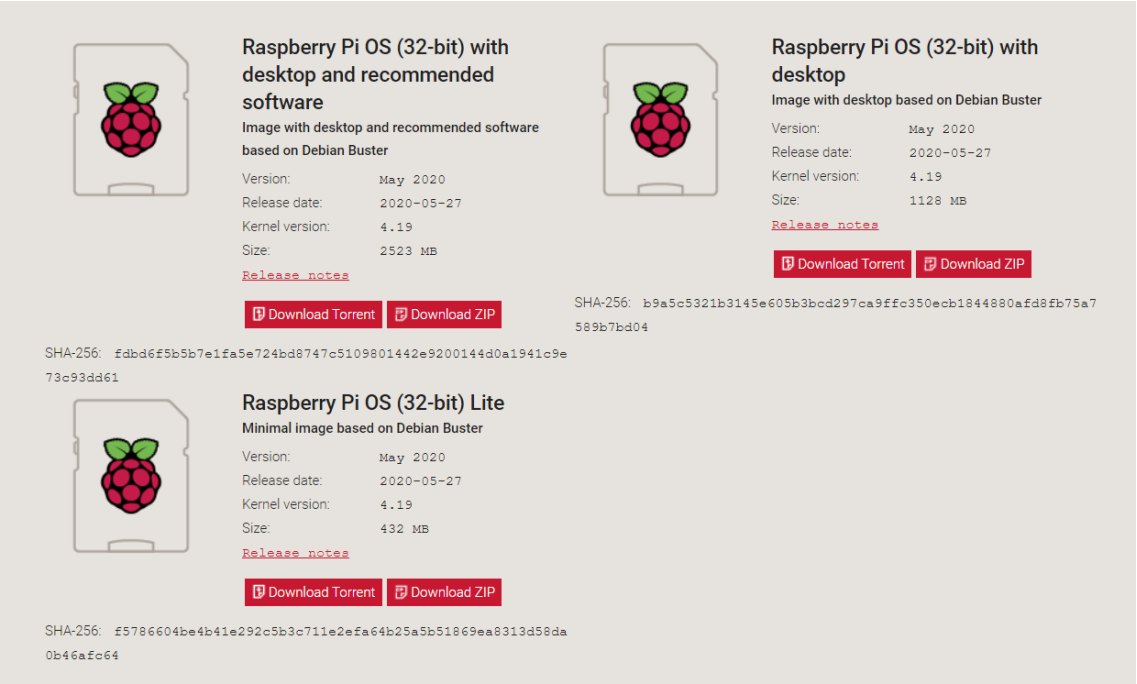
Raspberry Pi OS (32 bits) Lite → en esta versión no disponemos de entorno gráfico y no hay software instalado a parte del propio sistema operativo.
Esta opción no sirve para este proyecto debido a que se usa Node-RED que es un entorno visual.

Raspberry Pi OS (32 bits) con escritorio → en esta versión es exactamente la misma que en la versión Lite, es decir, no tiene software adicional instalado a parte del propio sistema operativo, pero sí que tiene un entorno gráfico.

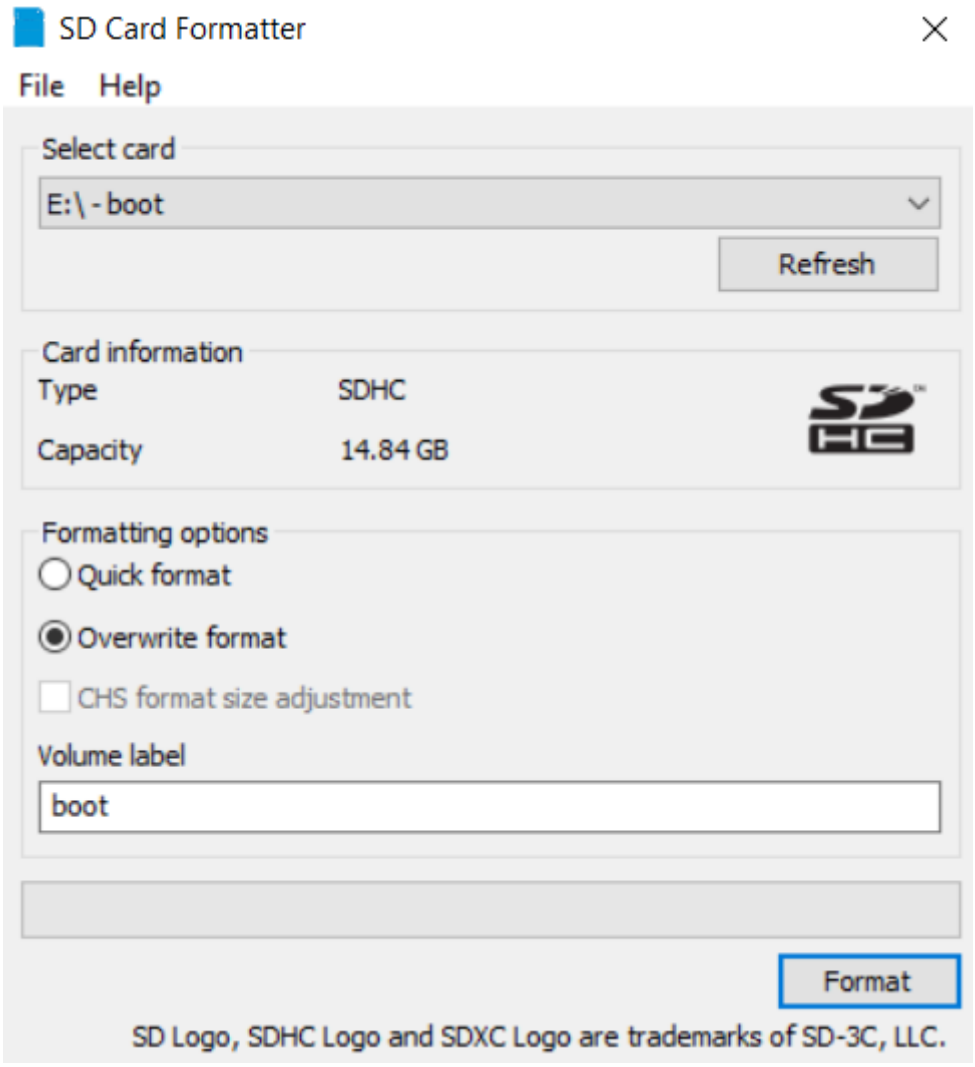
Raspberry Pi OS (32 bits) con escritorio y software recomendado → Esta versión es como la anterior, pero tiene instalado software a parte del sistema operativo, este es el que uso en el proyecto debido a que es más sencillo instalar programas como Node-RED o el bróker MQTT Mosquitto porque ja tiene instalado el software necesario para que estos programas funcionen correctamente.

El procedimiento para instalar es sistema operativo es:

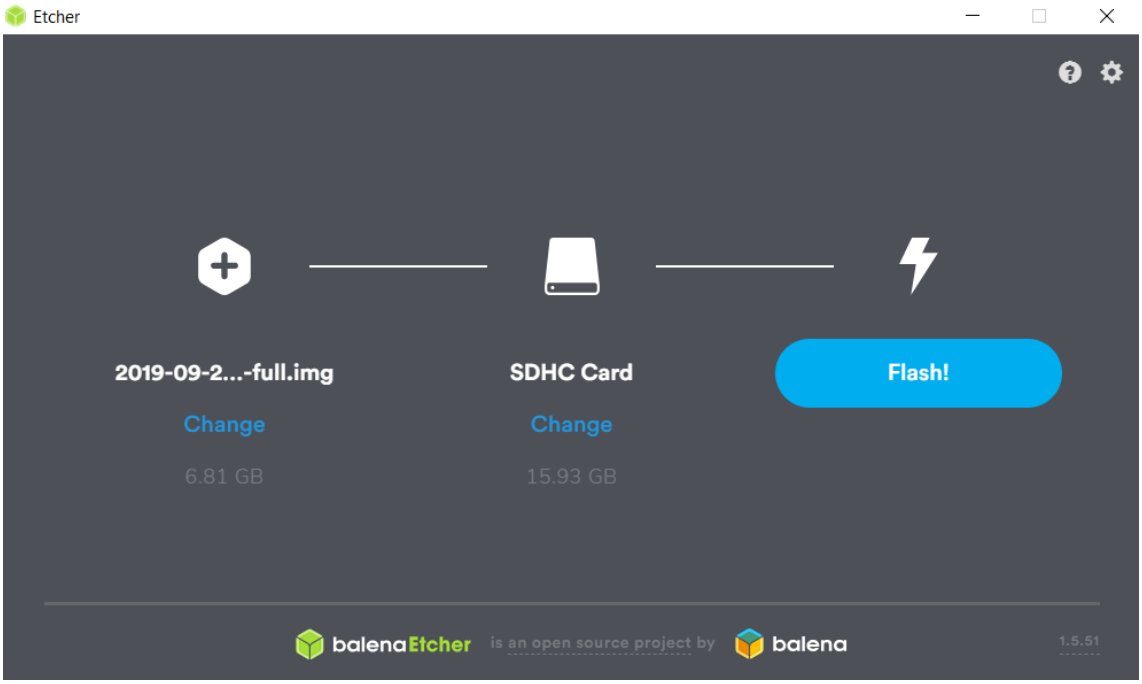
1. Descargar la imagen de la página oficial https://www.raspberrypi.org/downloads/raspberry-pi-os/



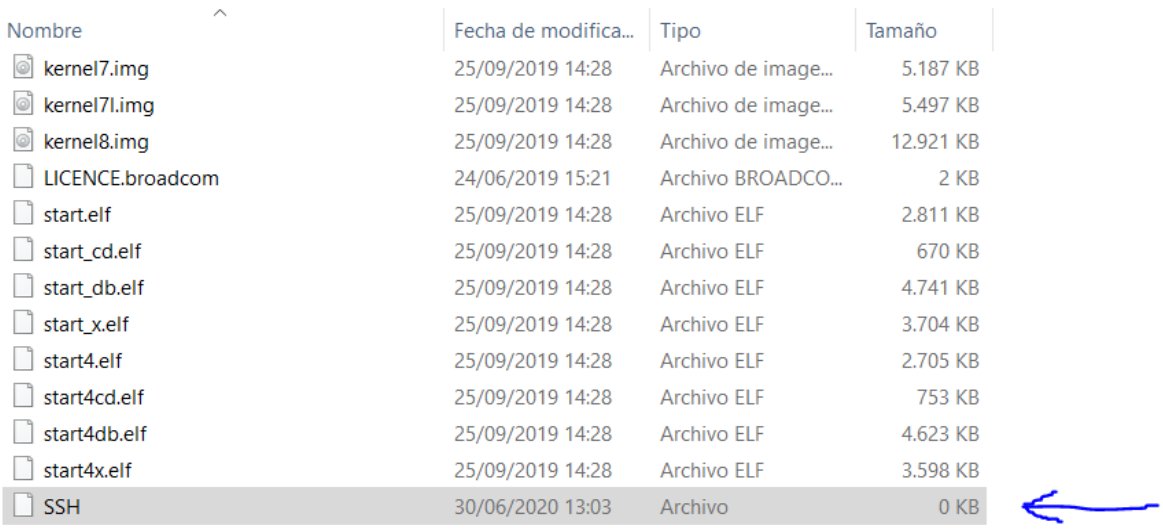
2. Formatear la tarjeta SD con un programa específico como podría ser https://www.sdcard.org/downloads/formatter/



3. Para grabar la imagen en la tarjeta SD se necesita un programa como balenaEtcher https://www.balena.io/etcher/



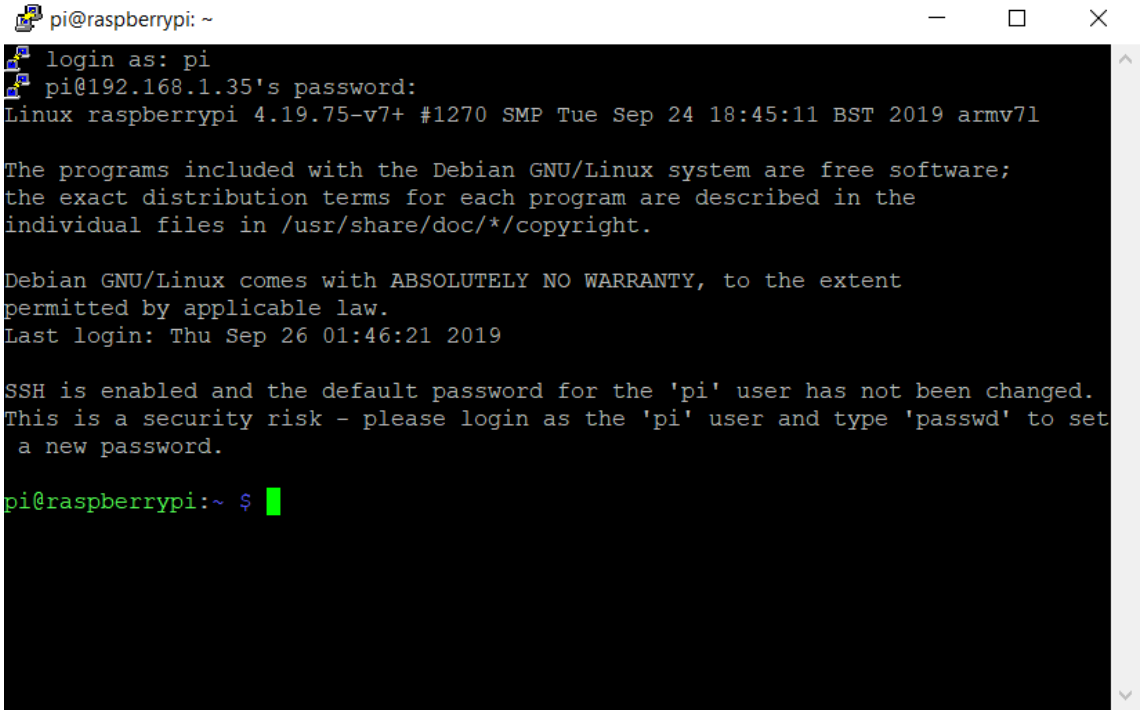
4. Crear un archivo sin extensión con el nombre “SSH” para tener acceso remoto.



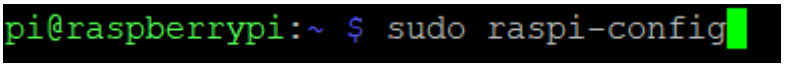
Nombre	Fecha de modifica...	Tipo	Tamaño
kernel7.img	25/09/2019 14:28	Archivo de image...	5.187 KB
kernel7l.img	25/09/2019 14:28	Archivo de image...	5.497 KB
kernel8.img	25/09/2019 14:28	Archivo de image...	12.921 KB
LICENCE.broadcom	24/06/2019 15:21	Archivo BROADCO...	2 KB
start.elf	25/09/2019 14:28	Archivo ELF	2.811 KB
start_cd.elf	25/09/2019 14:28	Archivo ELF	670 KB
start_db.elf	25/09/2019 14:28	Archivo ELF	4.741 KB
start_x.elf	25/09/2019 14:28	Archivo ELF	3.704 KB
start4.elf	25/09/2019 14:28	Archivo ELF	2.705 KB
start4cd.elf	25/09/2019 14:28	Archivo ELF	753 KB
start4db.elf	25/09/2019 14:28	Archivo ELF	4.623 KB
start4x.elf	25/09/2019 14:28	Archivo ELF	3.598 KB
SSH	30/06/2020 13:03	Archivo	0 KB

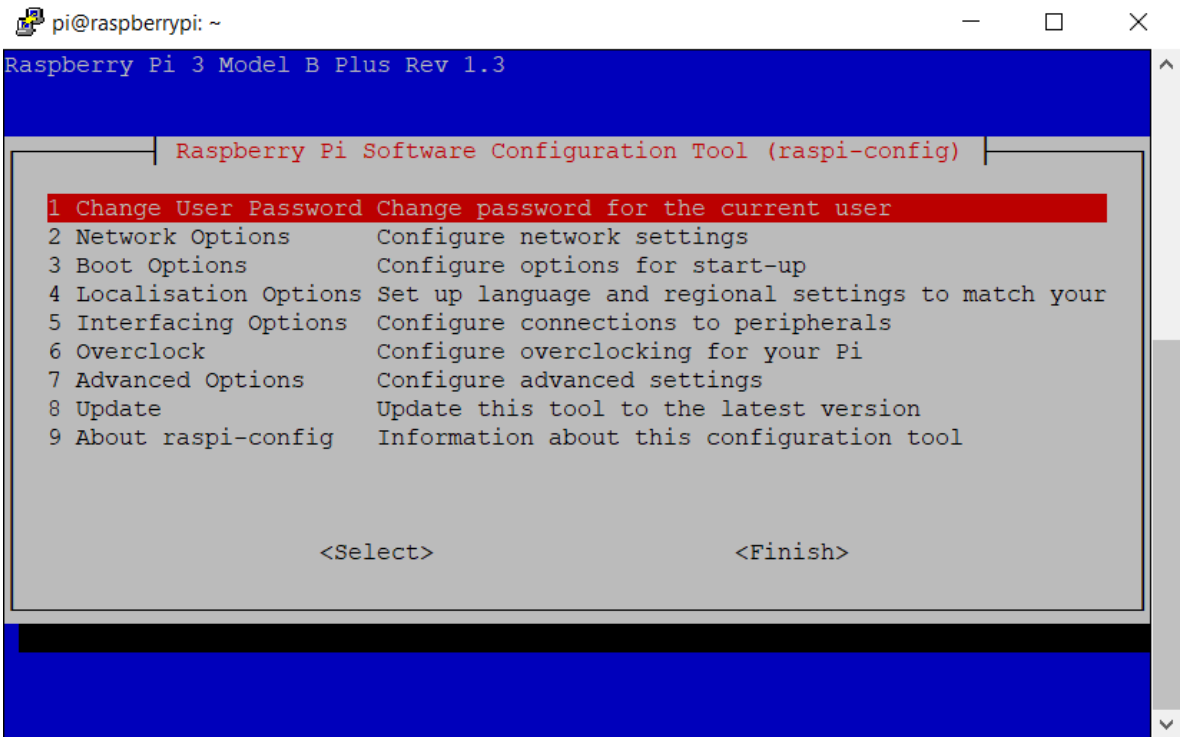
5. Expulsar la tarjeta e introducirla a la Raspberry Pi.

6. Acceder de manera remota con PuTTY



7. Realizar la configuración inicial de la Raspberry Pi





8. Actualizar por si hay modificaciones en el sistema operativo posteriores a la descarga.

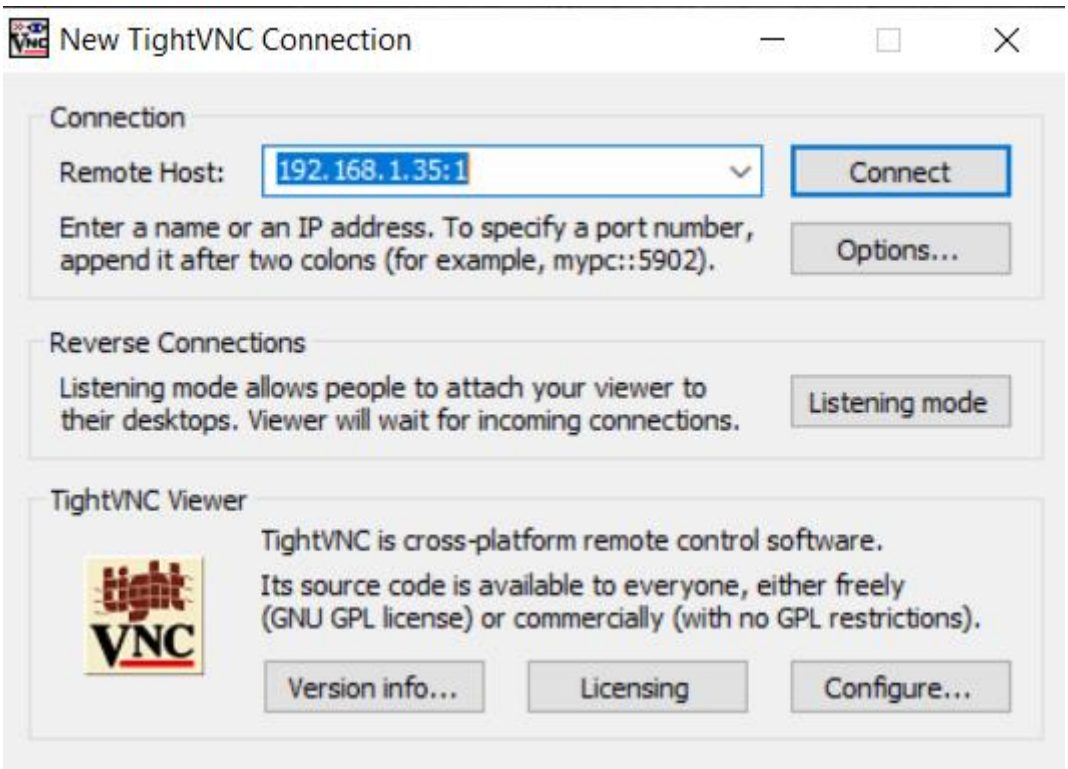
```
pi@raspberrypi:~ $ sudo apt-get update
```

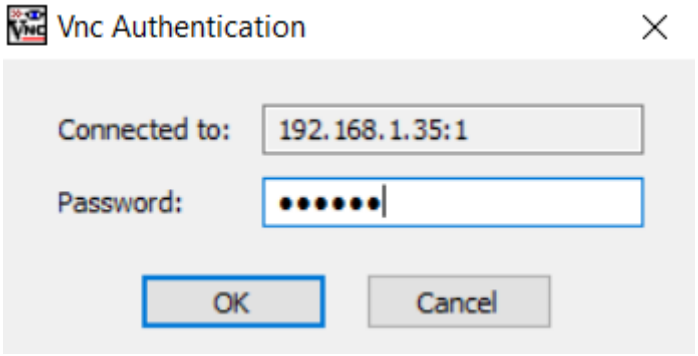
```
pi@raspberrypi:~ $ sudo apt-get upgrade
```

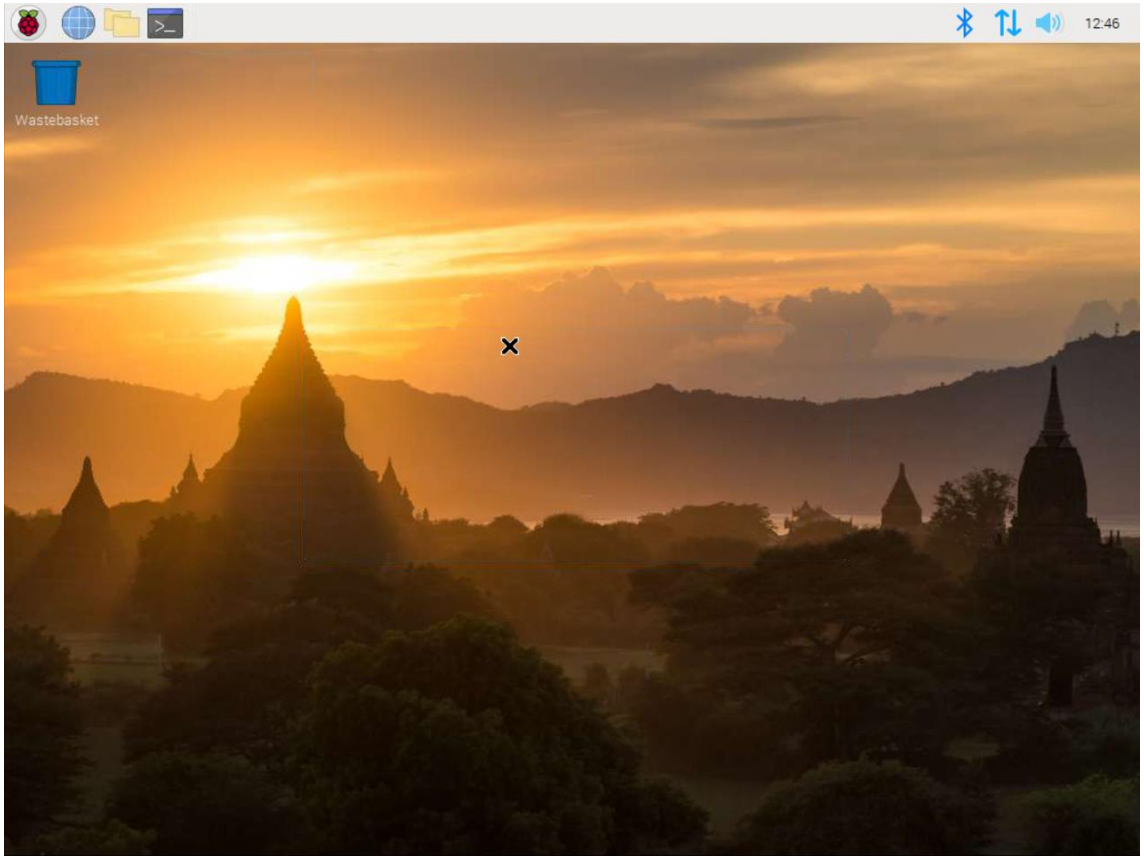
9. Instalar y ejecutar tightvncserver para poder acceder al escritorio de Raspbian de forma remota.

```
pi@raspberrypi:~ $ sudo apt-get install tightvncserver
```

```
pi@raspberrypi:~ $ tightvncserver

You will require a password to access your desktops.

Password:
Verify:
Would you like to enter a view-only password (y/n)? n

New 'X' desktop is raspberrypi:1

Creating default startup script /home/pi/.vnc/xstartup
Starting applications specified in /home/pi/.vnc/xstartup
Log file is /home/pi/.vnc/raspberrypi:1.log
```




Vnc Authentication
Connected to:
192.168.1.35:1
Password:
OK
Cancel


12:46
Wastebasket

3.2 Mosquitto

3.2.1 Instalación de mosquitto

Se introducen los siguientes comandos para instalar el bróker mosquitto.

```
pi@raspberrypi:~ $ sudo apt-get install mosquitto
```

```
pi@raspberrypi:~ $ wget http://repo.mosquitto.org/debian/mosquitto-repo.gpg.key
```

```
pi@raspberrypi:~ $ sudo apt-key add mosquitto-repo.gpg.key
```

```
pi@raspberrypi:~ $ cd /etc/apt/sources.list.d/
pi@raspberrypi:/etc/apt/sources.list.d $ sudo wget http://repo.mosquitto.org/debian/mosquitto-buster.list
```

A continuación, se introduce el siguiente comando para instalar mosquitto como cliente.

```
root@raspberrypi:~# apt-get install mosquitto
```

Opciones
-c
--config-file
Carga la configuración desde un archivo. Si no se da, los valores son los proporcionados por defecto
-d
--daemon
Mosquitto funciona en segundo plano como Daemon (tipo especial de proceso informático no interactivo, es decir, que se ejecuta en segundo plano en vez de ser controlado directamente por el usuario). Todo el resto de comportamiento se mantiene igual.
-p
--port
Escucha en el puerto especificado en vez de el puerto por defecto 1883. Esto funciona adicionalmente a la configuración del puerto establecido en el archivo config. Puede ser especificado múltiples veces para múltiples servicios en diferentes puertos.
-v
--verbose
Esto es equivalente a la configuración log_type a all en el archivo de configuración.

Configuración
El bróker puede ser configurado usando la configuración descrita en mosquitto.conf(5) y esto es el punto principal de información para Mosquitto. Los archivos requeridos para el soporte SSL/TLS están descritos en mosquitto-tls(7).

Estado bróker
Los clientes pueden encontrar información sobre el bróker suscribiéndose a los topics en la jerarquía $SYS como sigue. Los topics marcados como estáticos son solo enviados una vez por cliente suscrito. Todos los otros topics son actualizados cada sys_interval segundos.Si sys_interval es 0, entonces las actualizaciones no son enviadas.

$SYS/broker/bytes/received
El número total de bytes recibidos desde que el bróker inició.

$SYS/broker/bytes/sent
El número total de bytes enviados desde que el bróker inició.

$SYS/broker/clients/connected
El número actual de clientes conectados.

$SYS/broker/clients/expired
El número de clientes persistentes desconectados que hayan caducado y eliminados a través de la opción de persistent_client_expiration.

$SYS/broker/clients/disconnected
El número total de clientes persistentes (con la sesión limpiar desactivada) que se hayan registrado en el bróker pero que estén actualmente desconectados.

$SYS/broker/clients/máximum
El número máximo de clientes que han sido conectados al bróker al mismo tiempo.

$SYS/broker/clients/total
El número total de clientes activos e inactivos conectados y registrados actualmente en el bróker.

$SYS/broker/connection/#
Cuando los puentes son configurados para/desde el bróker, Es común proporcionar un topic de estado que indica el estado de la conexión. Esto es proporcionado en $SYS/broker/connection/ por defecto. Si el valor del topic es 1 la conexión se mantiene activa, si es 0 entonces no está activa.

$SYS/broker/heap/current size
El tamaño actual de la pila de memoria en uso por Mosquitto.

$SYS/broker/heap/máximum size
El mayor tamaño de la pila usado por Mosquitto.

$SYS/broker/load/connections/+
La media móvil del número de paquetes CONNECT recibidos por el bróker en diferentes intervalos de tiempo. El final “+” de la jerarquía puede ser 1min, 5 min o 15 min. El valor retornado representa el número de conexiones recibidas en 1 minuto, haciendo la media sobre 1, 5 o 15 minutos.

$SYS/broker/load/bytes/received/+
La media móvil del número de bytes recibidos por el bróker en diferentes intervalos de tiempo. El final “+” de la jerarquía puede ser 1min, 5 min o 15 min. El valor retornado representa el número de conexiones recibidas en 1 minuto, haciendo la media sobre 1, 5 o 15 minutos.

$SYS/broker/load/bytes/sent/+
La media móvil del número de bytes enviados por el bróker en diferentes intervalos de tiempo. El final “+” de la jerarquía puede ser 1min, 5 min o 15 min. El valor retornado representa el número de conexiones recibidas en 1 minuto, haciendo la media sobre 1, 5 o 15 minutos.

$SYS/broker/load/messages/received/+
La media móvil del número de todos los tipos de mensajes recibidos por el bróker en diferentes intervalos de tiempo. El final “+” de la jerarquía puede ser 1min, 5 min o 15 min. El valor retornado representa el número de conexiones recibidas en 1 minuto, haciendo la media sobre 1, 5 o 15 minutos.

$SYS/broker/load/messages/sent/+
La media móvil del número de todos los tipos de mensajes enviados por el bróker en diferentes intervalos de tiempo. El final “+” de la jerarquía puede ser 1min, 5 min o 15 min. El valor retornado representa el número de conexiones recibidas en 1 minuto, haciendo la media sobre 1, 5 o 15 minutos.

$SYS/broker/load/publish/dropped/+
La media móvil del número de mensajes publicados perdidos por el bróker en diferentes intervalos de tiempo. El final “+” de la jerarquía puede ser 1min, 5 min o 15 min. El valor retornado representa el número de conexiones recibidas en 1 minuto, haciendo la media sobre 1, 5 o 15 minutos.

$SYS/broker/load/publish/received/+
La media móvil del número de mensajes publicados recibidos por el bróker en diferentes intervalos de tiempo. El final “+” de la jerarquía puede ser 1min, 5 min o 15 min. El valor retornado representa el número de conexiones recibidas en 1 minuto, haciendo la media sobre 1, 5 o 15 minutos.

$SYS/broker/load/publish/sent/+
La media móvil del número de mensajes publicados enviados por el bróker en diferentes intervalos de tiempo. El final “+” de la jerarquía puede ser 1min, 5 min o 15 min. El valor retornado representa el número de conexiones recibidas en 1 minuto, haciendo la media sobre 1, 5 o 15 minutos.

$SYS/broker/load/sockets/+
La media móvil del número de conexiones abiertas por el bróker en diferentes intervalos de tiempo. El final “+” de la jerarquía puede ser 1min, 5 min o 15 min. El valor retornado representa el número de conexiones recibidas en 1 minuto, haciendo la media sobre 1, 5 o 15 minutos.

$SYS/broker/messages/inflight
El número de mensajes con QoS>0 que están esperando “acknowledgments”

$SYS/broker/messages/received
El número total de mensajes de cualquier tipo recibidos desde el inicio del bróker.

$SYS/broker/messages/sent
El número total de mensajes de cualquier tipo enviados desde el inicio del bróker.

$SYS/broker/publish/messages/dropped
El número total de mensajes publicados que han sido perdidos debido a los límites de inflight/queuing. Ver las opciones max_inflight_messages y max_queued_messages.

$SYS/broker/publish/messages/received
El número total de mensajes publicados recibidos desde el incio del bróker.

$SYS/broker/publish/messages/sent
El número total de mensajes publicados enviados desde el inicio del bróker.

$SYS/broker/retained messages/count
El número total de mensajes activos en el bróker.

$SYS/broker/store/messages/count
El número actual de mensajes retenidos en el almacenamiento de mensajes. Esto incluye mensajes retenidos y mensajes en la cola de cliente duraderos.

$SYS/broker/store/messages/bytes
El número de bytes actuales retenidos en la carga de mensajes en el almacenamiento. Esto incluye mensajes retenidos y mensajes en la cola de clientes duraderos.

$SYS/broker/subscriptions/count
El número total de suscripciones activas en el bróker.

$SYS/broker/versión
La versión del broker. Estatico.

Puentes
Múltiples brokers pueden ser conectados juntos con una funcionalidad de puente. Esto puede ser útil cuando es deseable compartir información entre ubicaciones, pero donde no toda la información necesita ser compartida. Un ejemplo podría ser donde un número de usuarios están utilizando un bróker para ayudar a registrar el uso de potencia. El uso de potencia podría ser compartido a través de puentear todos los brokers de usuario a un bróker común, permitiendo que el uso de potencia de todos los usuarios pudiera ser recopilada y comparada. La otra información se mantendría local en cada bróker.

Señales
SIGHUP
Al recibir la señal SIGHUP, mosquitto intentará volver a cargar los datos del archivo de configuración, suponiendo que se proporcionó el argumento -c cuando se inició mosquitto. No todos los parámetros de configuración se pueden volver a cargar sin reiniciar.

SIGUSR1
Al recibir la señal SIGUSR1, mosquitto escribirá la base de datos de persistencia en el disco. Esta señal solo actúa si la persistencia está activada.

SIGUSR2
La señal SIGUSR2 hace que mosquitto imprima el árbol de suscripción actual, junto con información sobre dónde existen los mensajes retenidos. Esto es solo una función de prueba y puede eliminarse en cualquier momento.

Archivos

/etc/mosquitto/mosquitto.conf
Archivo de configuración.

/var/lib/mosquitto/mosquitto.db
Ubicación de almacenamiento de los mensajes persistentes si persist está activado.

/etc/hosts.allow
/etc/hosts.deny
Control de acceso de usuarios via tcp-wrappers.

3.2.2 Estructura de topics

Raspberry pi (Broker MQTT)

- ESP32 (Entrada) → home/entrada
  - Sensor de presencia (pres0.0) → home/entrada/presencia0
  - Pulsador (pul0.0) → home/entrada/pulsador0
  - Punto de luz (lamp0.0) → home/entrada/lampara0
  - Timbre (timbre0.0) → home/entrada/timbre
  - Pulsador (pul1.0) → home/entrada/pulsador1
  - Pulsador (pul1.1) → home/entrada/pulsador2
  - Sensor de presencia (pres1.0) → home/entrada/presencia1
  - Punto de luz (lamp1.0) → home/entrada/lampara1
  - Punto de luz (lamp1.1) → home/entrada/lampara2

- ESP32 (Habitación 1) → home/hab1
  - Pulsador (pul2.0) → home/hab1/pulsador0
  - Pulsador (pul2.1) → home/hab1/pulsador1
  - Pulsador (pul2.2) → home/hab1/pulsador2
  - Pulsador (pul2.3) → home/hab1/pulsador3
  - Sensor de temperatura (temp2.0) → home/hab1/temperatura
  - Punto de luz (lamp2.0) → home/hab1/lampara0
  - Calefacción (res2.0) → home/hab1/calefaccion
- ESP32(Baño1) → home/baño1
  - Pulsador (pul3.0) → home/baño/pulsador0
  - Pulsador (pul3.1) → home/baño/pulsador1
  - Pulsador (pul3.2) → home/baño/pulsador2
  - Pulsador (pul3.3) → home/baño/pulsador3
  - Punto de luz (lamp3.0) → home/baño/lampara0
  - Punto de luz (lamp3.1) → home/baño/lampara1
  - Punto de luz (lamp3.2) → home/baño/lampara2
  - Sensor de temperatura (temp3.0) → home/baño/temperatura
  - Calefacción (res3.0) → home/baño/calefaccion

- ESP32 (Habitación 2) → home/hab2
  - Pulsador (pul4.0) → home/hab2/pulsador
  - Pulsador (pul4.1) → home/hab2/pulsador
  - Pulsador (pul4.2) → home/hab2/pulsador
  - Pulsador (pul4.3) → home/hab2/pulsador
  - Sensor de temperatura (temp4.0) → home/hab2/temperatura
  - Punto de luz (lamp4.0) → home/hab2/lampara0
  - Calefacción (res4.0) → home/hab2/calefaccion

ESP32(Salón/Comedor) → home/comedor

- Pulsador (pul5.0) → home/entrada/ pulsador0
- Pulsador (pul5.1) → home/comedor/pulsador1
- Pulsador (pul5.2) → home/comedor/pulsador2
- Pulsador (pulperup5.0) → home/comedor/pulsadorUp0
- Pulsador (pulperdown5.0) → home/comedor/pulsadorDown0
- Pulsador (pulperup5.1) → home/comedor/pulsadorUp1
- Pulsador (pulperdown5.1) → home/comedor/pulsadorDown1
- Pulsador (pulperup5.2) → home/comedor/pulsadorUp2
- Pulsador (pulperdown5.2) → home/comedor/pulsadorDown2
- Punto de luz (lamp5.0) → home/comedor/lampara0
- Punto de luz (lamp5.1) → home/comedor/lampara1
- Sensor de temperatura (temp5.0) → home/comedor/temperatura1
- Sensor de temperatura (temp5.1) → home/comedor/temperatura2
- Calefacción (res5.0) → home/comedor/calefaccion0
- Calefacción (res5.1) → home/comedor/calefaccion1
- Relé (perup0) → home/comedor/persianaUp0
- Relé (perdown0) → home/comedor/persianaDown0
- Relé (perup1) → home/comedor/persianaUp1
- Relé (perdown1) → home/comedor/persianaDown1
- Relé (perup2) → home/comedor/persianaUp2
- Relé (perdown2) → home/comedor/persianaDown2

ESP32(Patio) → home/patio

- Pulsador (pul6.0) → home/patio/pulsador0
- Pulsador (pul6.1) → home/patio/pulsador1
- Pulsador (pul6.2) → home/patio/pulsador2
- Pulsador (pul6.3) → home/patio/pulsador3
- Punto de luz (lamp6.0) → home/patio/lampara0
- Punto de luz (lamp6.1) → home/patio/lampara1

ESP32(Baño2) →home/baño2

- Pulsador (pul7.0) → home/baño2/pulsador0
- Punto de luz (lamp7.0) → home/baño2/lampara0

ESP32(Cocina) → home/cocina

- Pulsador (pul8.0) → home/comedor pulsador0
- Pulsador (pul8.1) → home/cocina/pulsador1
- Pulsador (pul8.2) → home/cocina/pulsador2
- Punto de luz (lamp8.0) → home/cocina/lampara0
- Punto de luz (lamp8.1) → home/cocina/lampara1
- Sensor de temperatura (temp8.0) → home/cocina/temperatura
- Calefacción (res8.0) → home/cocina/calefaccion
- Sensor de gas (gas8.0) → home/cocina/alarmaGas
- Zumbador (zum8.0) → home/cocina/zumbador

3.3 Node-RED

Node-RED es una herramienta de programación visual. Muestra visualmente las relaciones y funciones, y permite al usuario programar sin tener que escribir en un lenguaje de programación. Node-RED es un editor de flujo basado en el navegador donde se puede añadir o eliminar nodos y conectarlos entre sí con el fin de hacer que se comuniquen entre ellos.
En Node-RED, cada nodo es uno de los siguientes tipos: un nodo de inyección o un nodo de función. Los nodos de inyección producen un mensaje sin necesidad de entrada y lanzan una entrada y realizan algún trabajo con él. Con una gran cantidad de estos nodos para elegir, Node-RED hace que el conectar los dispositivos de hardware, API's y servicios en línea sea más fácil que nunca.
Para instalar Node-RED tenemos que tener instalado en Node.js (que está incluido cuando instalas Node-RED) y npm que no está incluido en este paquete.
NPM es un sistema de gestión de paquetes por defecto para Node.js, un entorno de ejecución para JavaScript.
Existen dos opciones:

Local → empieza a funcionar cuando introducimos la instrucción *node-red* en el terminal y finaliza cuando pulsamos Ctrl-C.

Como servicio → Esto significa que puede funcionar en segundo plano y activarse automáticamente cuando el dispositivo arranca.

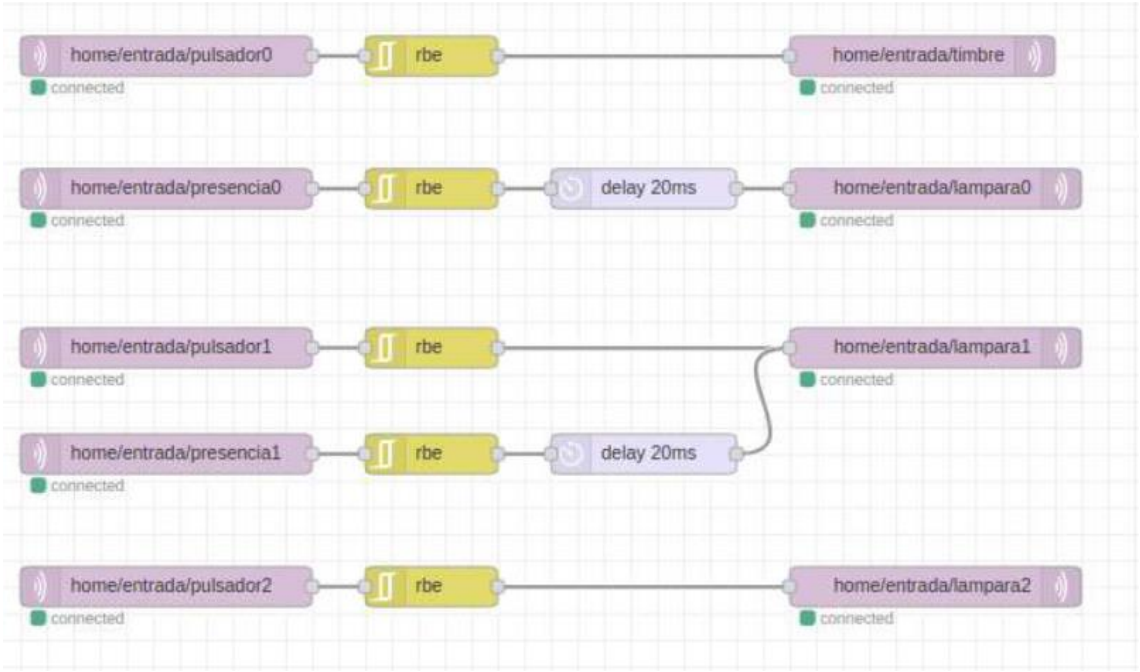
En este caso usaríamos Node-RED como servicio.
Para hacer esto debemos introducir el siguiente comando:

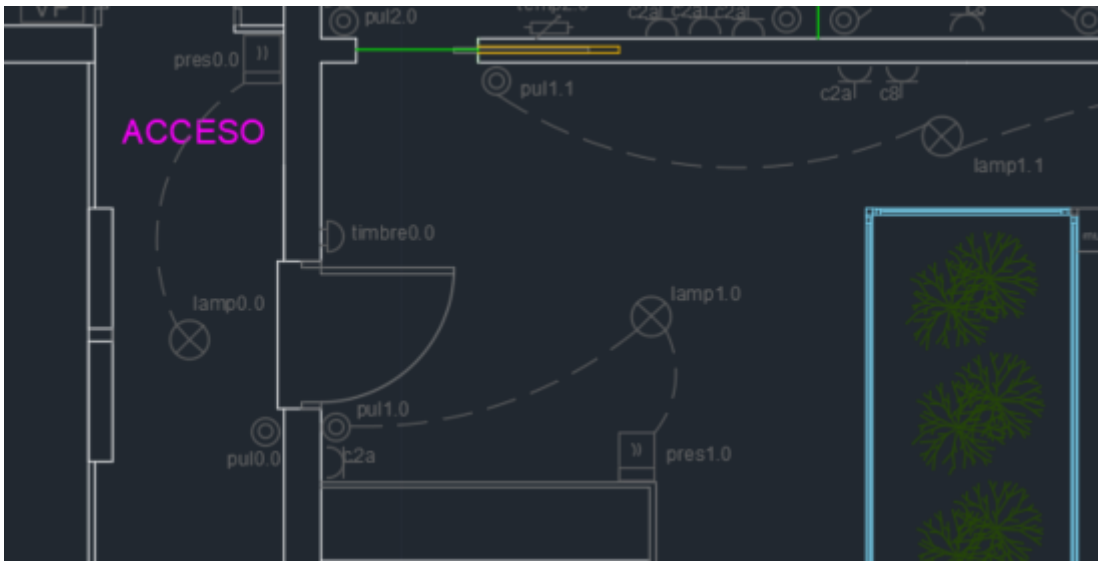
sudo systemctl enable nodered.service

Para desactivar este servicio se introduce:

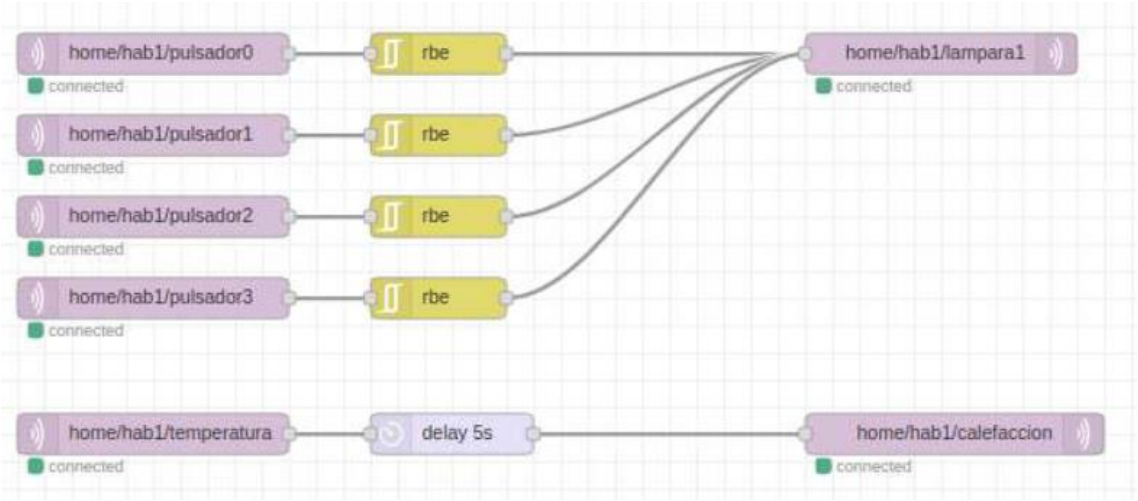
sudo systemctl disable nodered.service

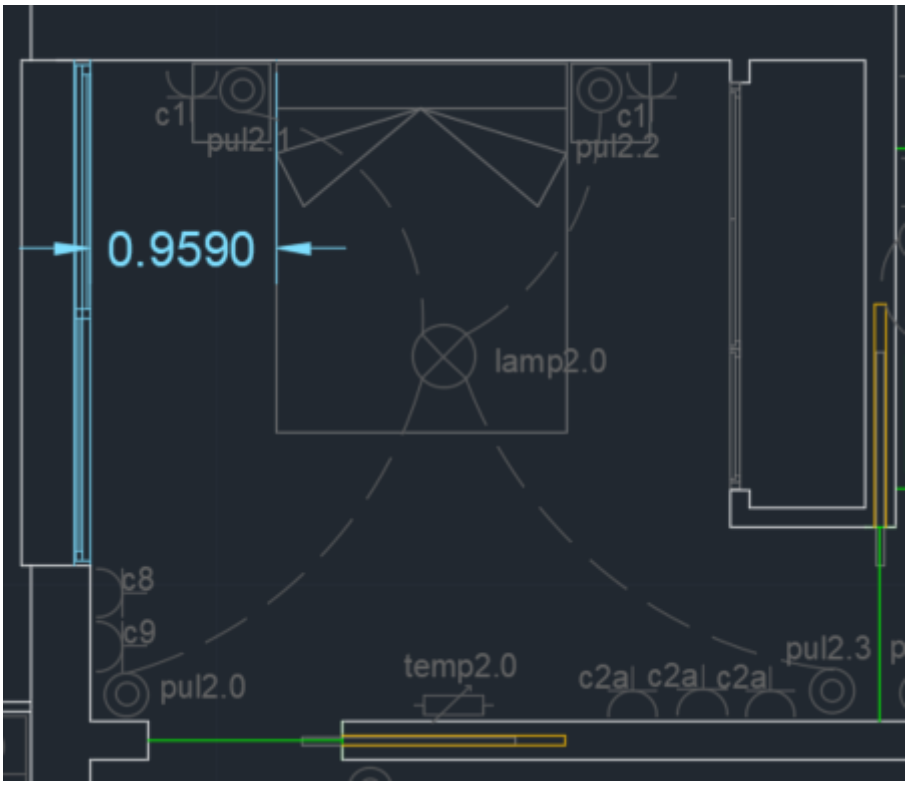
3.3.1 Programación Node-RED Entrada/Recibidor



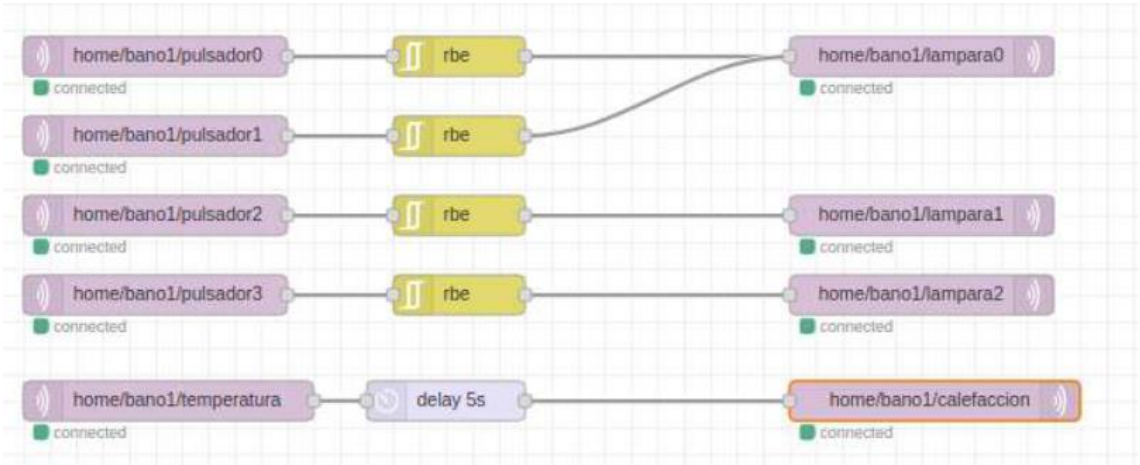


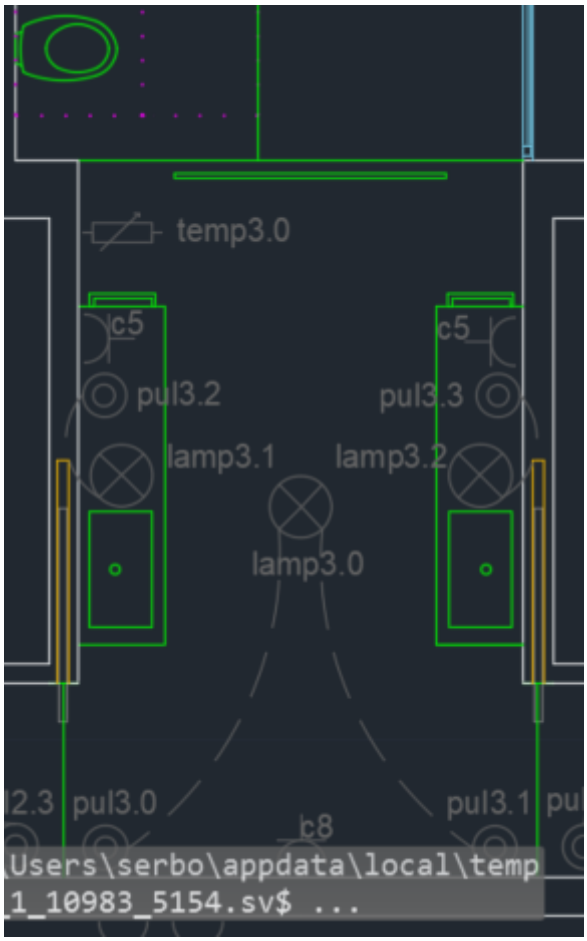
3.3.2 Programación Node-RED Habitación 1



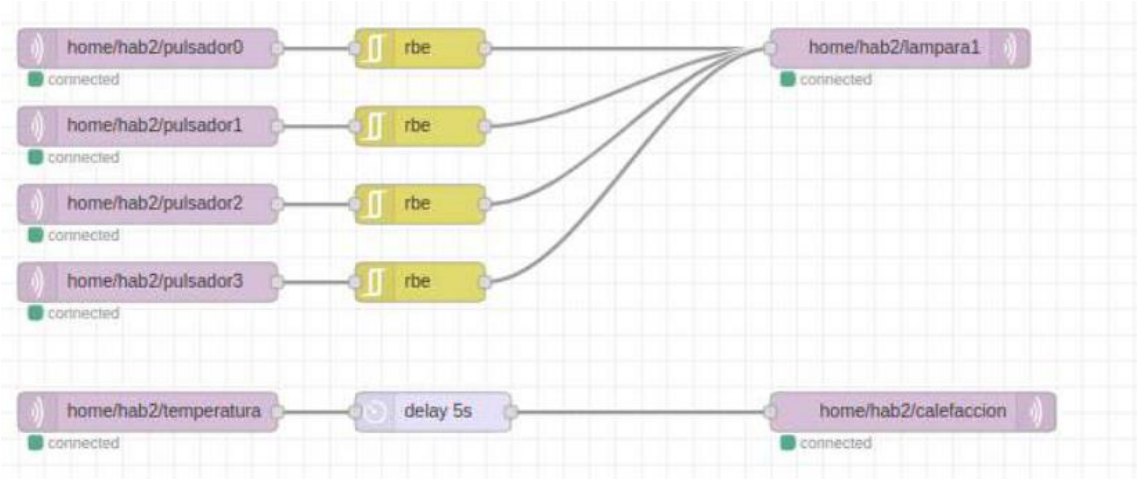


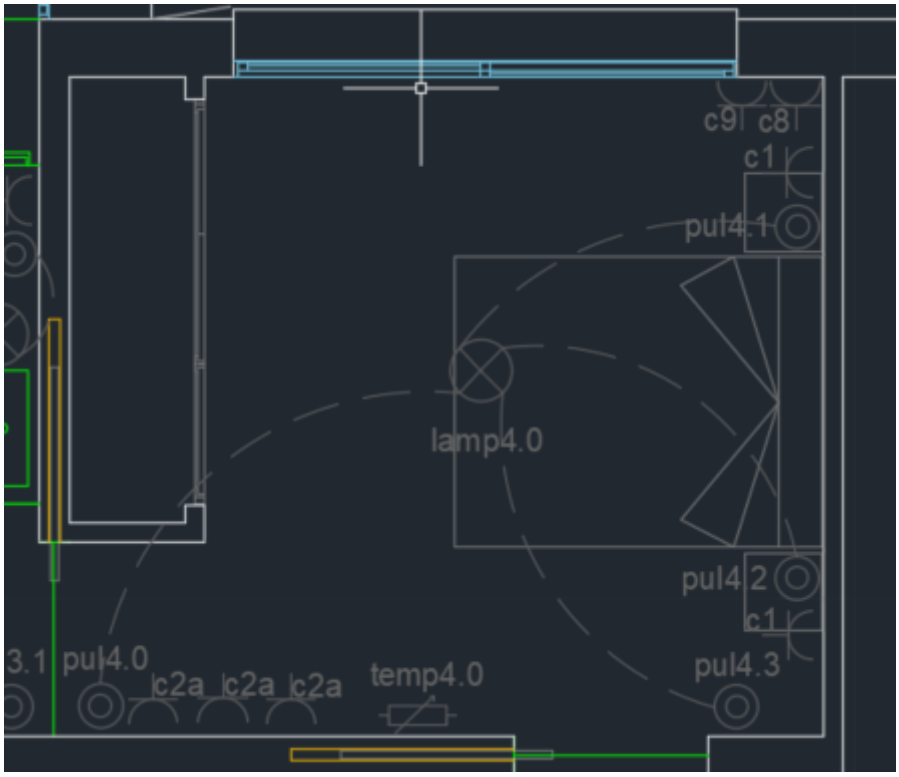
3.3.3 Programación Node-RED Baño1



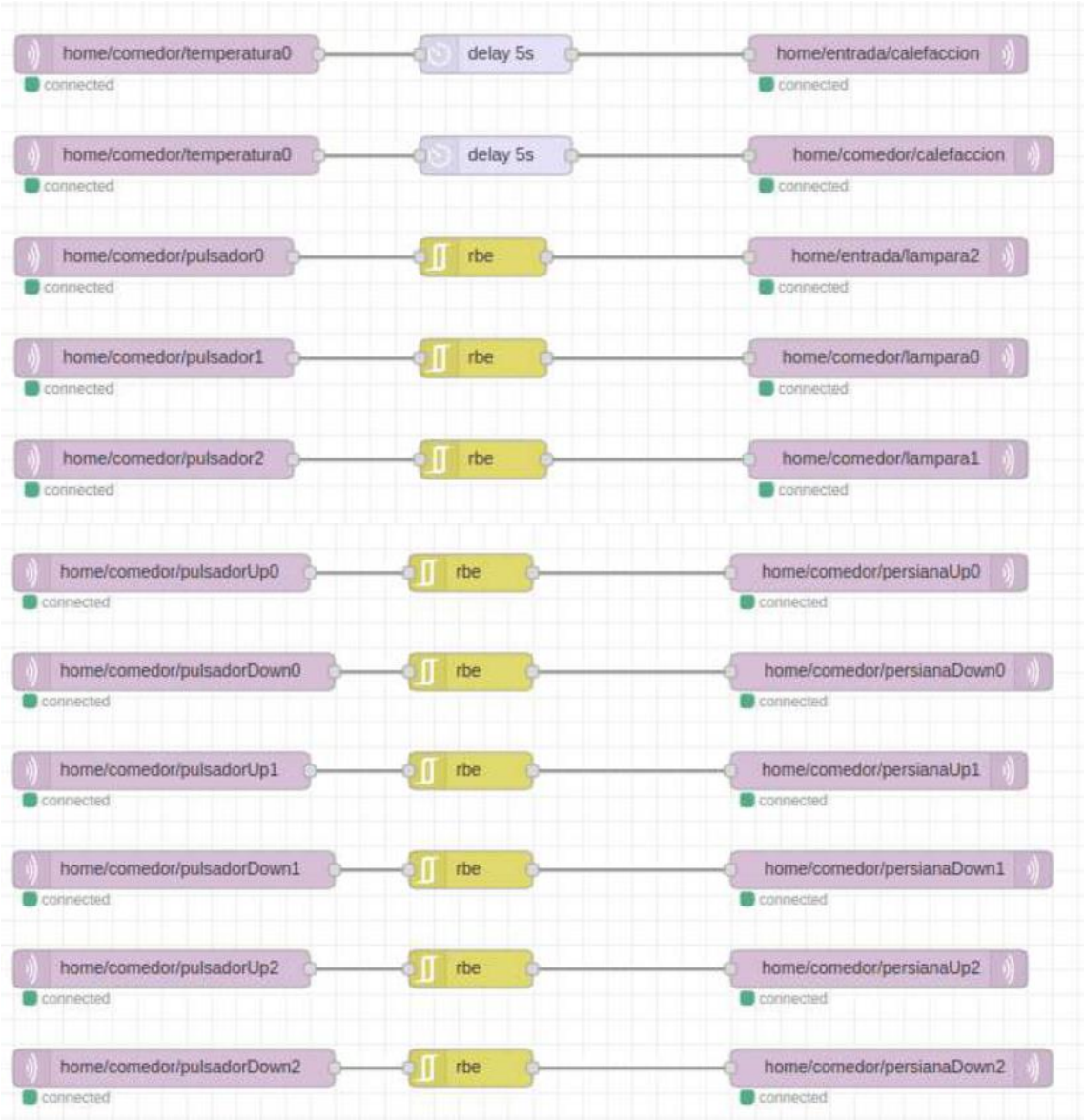


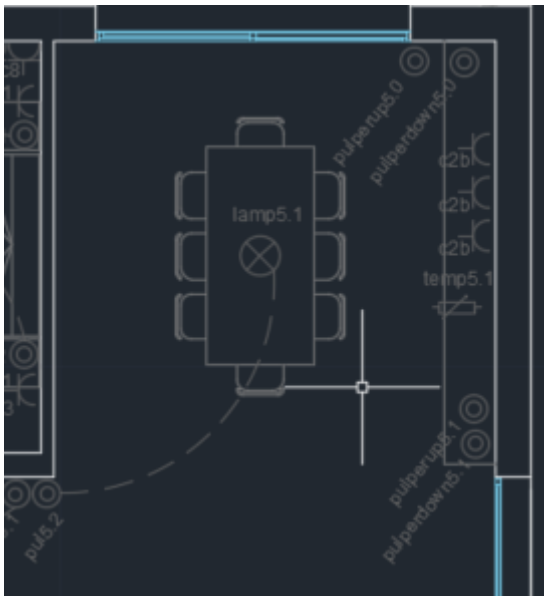
3.3.4 Programación Node-RED Habitación 2

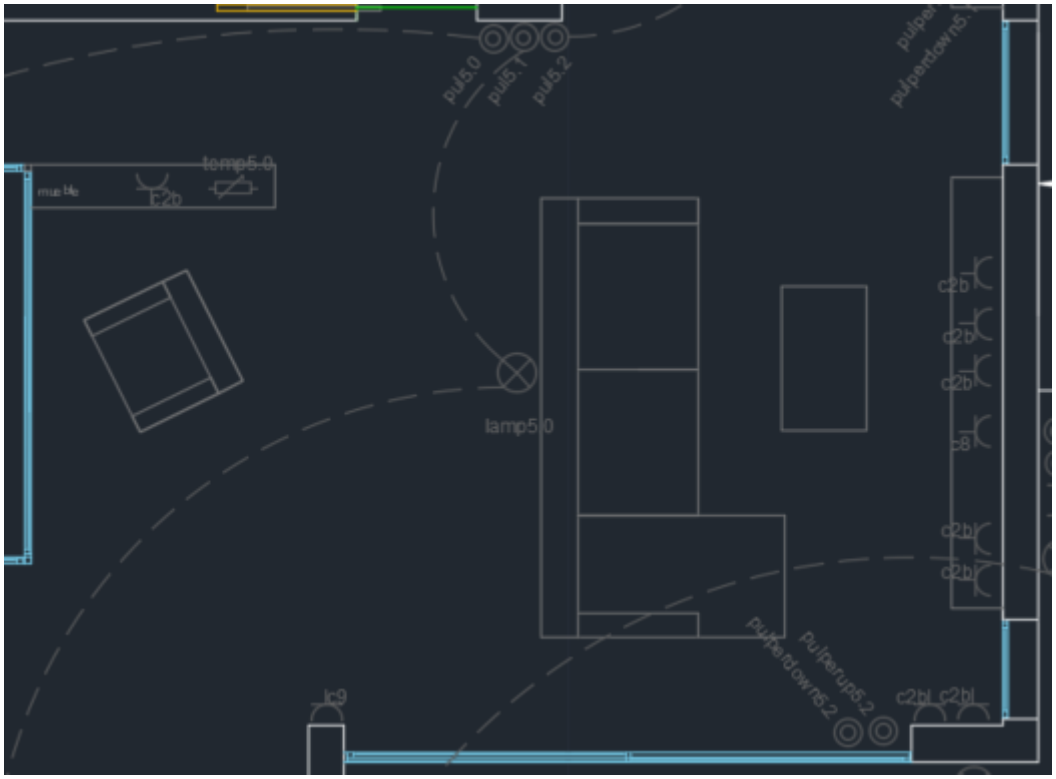




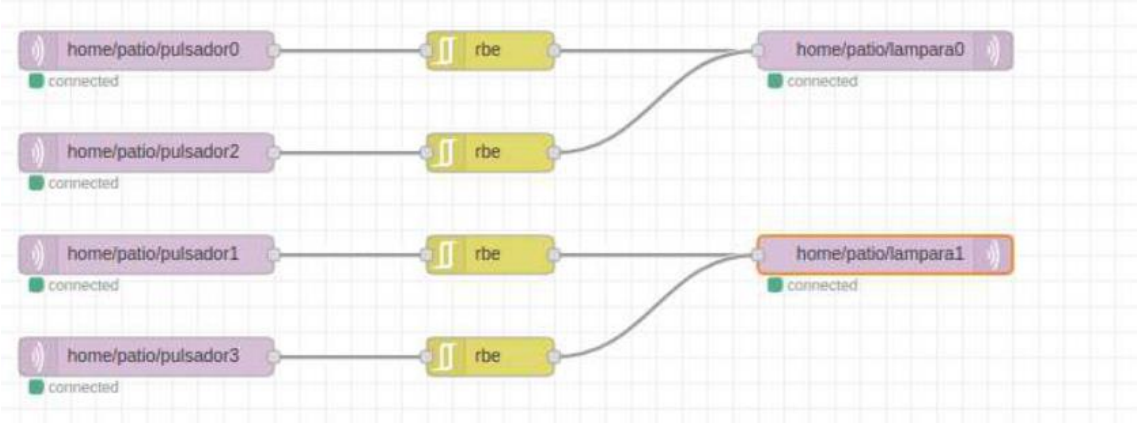
3.3.5 Programación Node-RED Comedor

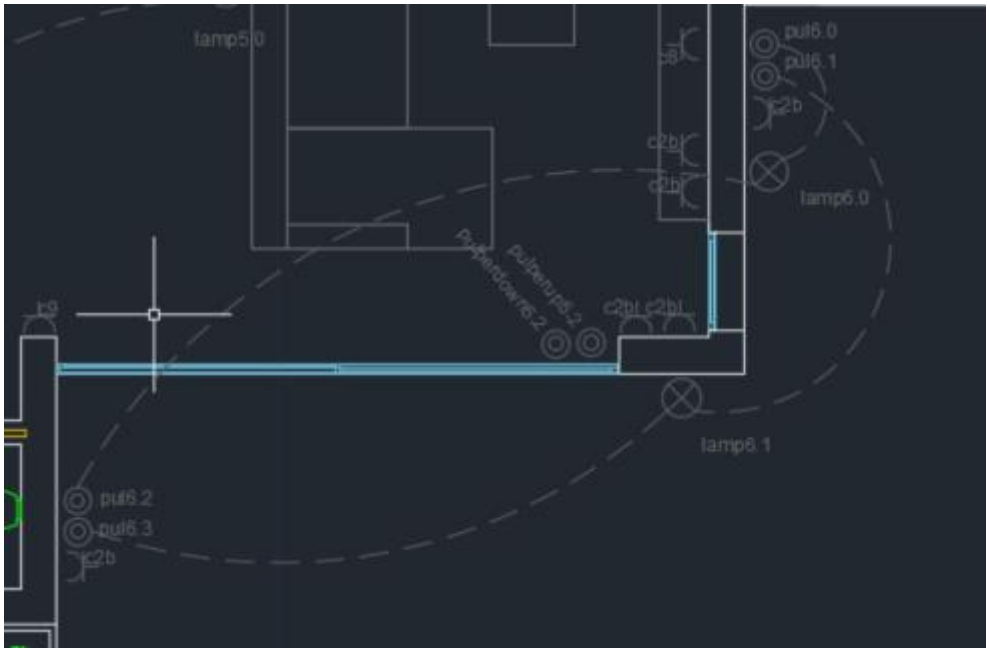


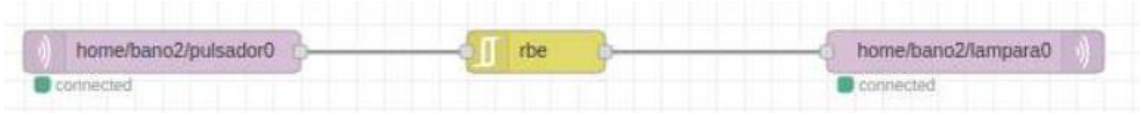

lamp5.1
temp5.1
c2b
c2b
c2b
puj5.2

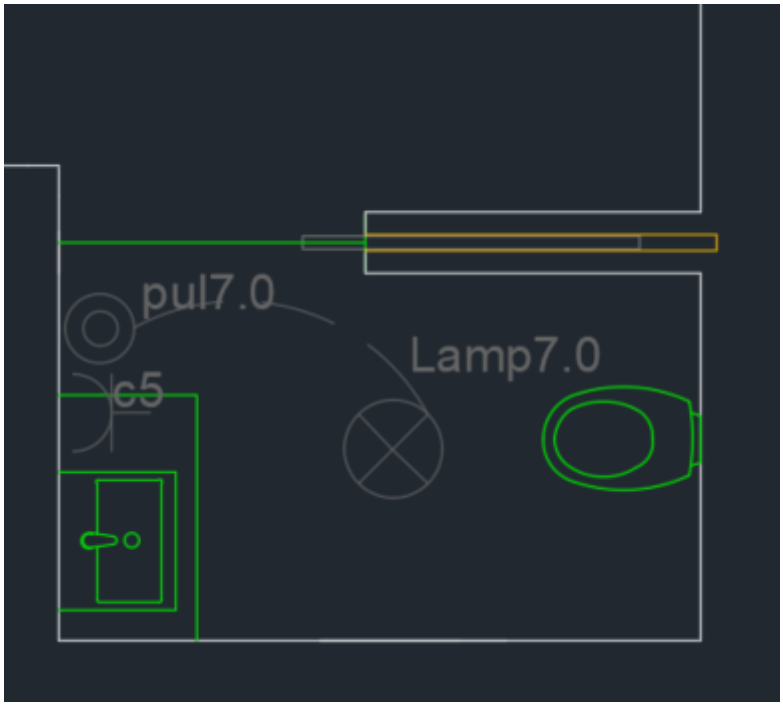

puj5.0
puj5.1
puj5.2
temp5.0
mueble
F2b
lamp5.0
c2b
c2b
c2b
c8
c2b
c2b
lc9
c2bl
c2bl

3.3.6 Programación Node-RED Patio y Baño2

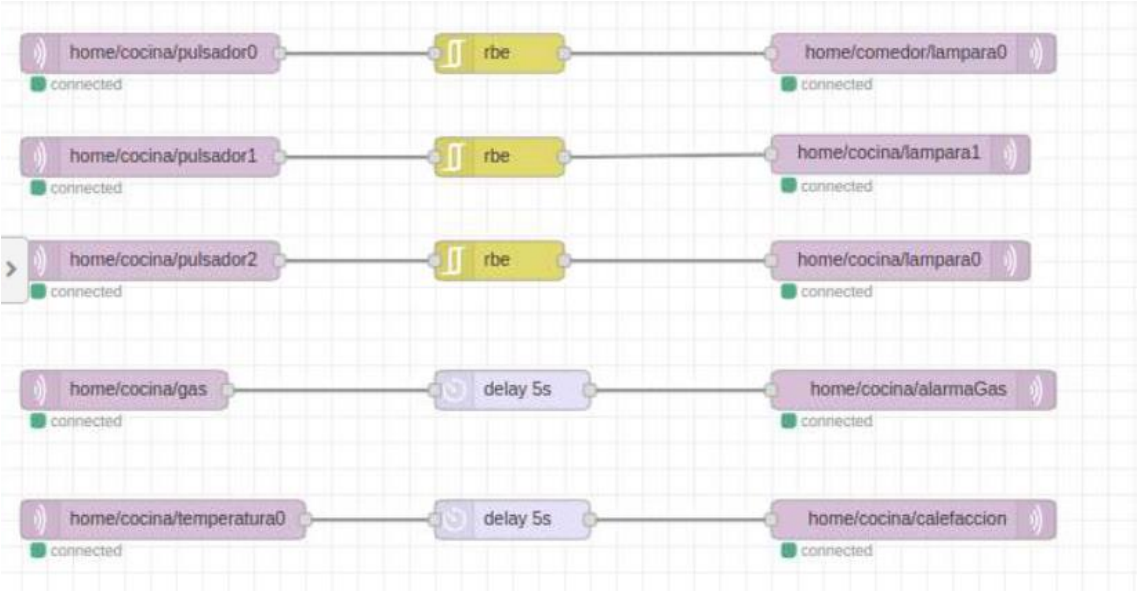


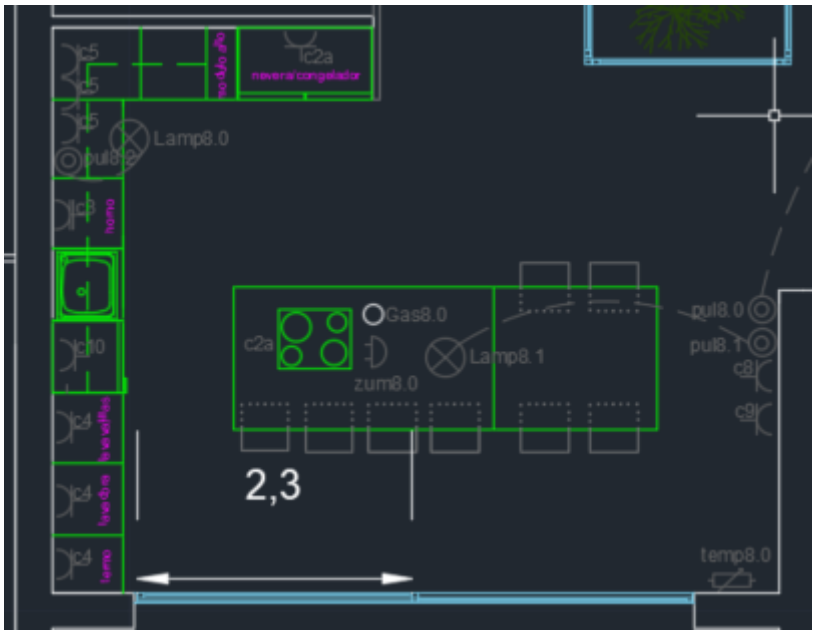



home/bano2/pulsador0
connected
rbe
home/bano2/lampara0
connected


pul7.0
Lamp7.0
c5

3.3.6 Programación Node-RED Cocina





3.4 ESP32

3.4.1 Programación ESP32 Entrada/Recibidor

```
#include <WiFi.h>
#include <PubSubClient.h>

const char* ssid = "Nombre de la red wifi";
//nombre de la red wifi de la vivienda
const char* password = "contraseña de la red wifi";
//contraseña de la red wifi anteriormente mencionada
const char* mqtt_server = "192.168.1.39";
//dirección ip del broker mqtt

const int pres00 = 34;
//GPIO al que está conectado el sensor de presencia
pres0.0
const int pres10 = 35;
//GPIO al que está conectado el sensor de presencia
pres1.0

const int pul00 = 32;
//GPIO al que está conectado el pulsador
pul0.0
const int pul10 = 33;
//GPIO al que está conectado el pulsador
pul1.0
const int pul11 = 25;
//GPIO al que está conectado el pulsador pul1.1

const int lamp00 = 21;
//GPIO al que está conectado la lámpara
lamp0.0
const int lamp10 = 19;
//GPIO al que está conectado la lámpara
lamp1.0
const int lamp11 = 18;
//GPIO al que está conectado la lámpara
lamp1.1
const int timbre00 = 17;
//GPIO al que está conectado el timbre
timbre0.0
const int kEnt = 16;
//GPIO al que está conectado el relé
kEnt

int contador1 = 0;
//Esta variable se utiliza para cambiar de estado a cada pulsación del
pulsador pul1.0
int contador2 = 0;
//Esta variable se utiliza para cambiar de estado a cada pulsación del
pulsador pul1.1
int presencia00 = 0;
//Esta variable contiene el valor de la entrada digital a la que está
conectado el sensor de presencia
pres0.0
int presencia10 = 0;
//Esta variable contiene el valor de la entrada digital a la que está
conectado el sensor de presencia
pres1.0
int pulsador00 = 0;
```

```
//Esta variable contiene el valor de la entrada digital a la que está
conectado el pulsador
pul0.0
int pulsador10 = 0;
//Esta variable contiene el valor de la entrada digital a la que está
conectado el pulsador
pul1.0
int pulsador11 = 0;
//Esta variable contiene el valor de la entrada digital a la que está
conectado el pulsador
pul1.1

//El siguiente grupo de variables se utiliza para primero convertir el
meensaje que entra en una cadena y segundo
//convertir esta cadena en una variable float
String str;
char str0;
char str1;
char str2;
char str3;
char str4;
byte mat[10];
float temperatura;

WiFiClient espClient;
//Crea un cliente que se puede conectar a una determinada dirección IP
definida en
client.connect()

PubSubClient client(espClient);
//Crea un cliente para publicar y suscribirse a través de un
determinado cliente WiFi

char msg[50];
//Se declara la variable que contiene el mensaje que se envía por
MQTT
long lastMsg = 0;

//Esta función establece conexión con la red WiFi de la vivienda
void setup_wifi()
{

  delay(10);
  // We start by connecting to a WiFi network
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  randomSeed(micros());

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
```

```
}

//Esta función es la que se encarga de recibir e interpretar los
mensajes MQTT de los topics suscritos
void callback(char* topic, byte* payload, unsigned int length)
{
  //Cuando un mensaje tiene el topic "home/entrada/timbre" este entra
en el bucle if
  if
(strcmp(topic,"home/entrada/timbre")==0){
    //comprueba si el segundo caracter es 'N' porque el mensaje que se
recibe es "ON" o "OFF"
    if ((char)payload[1] == 'N'){
      //por tanto cuando el mensaje de entrada es "ON" se activa una
salida digital conectada al
timbre
    digitalWrite(timbre00,
HIGH);
   }else{
    //en el caso de que el mensaje de entrada sea "OFF" se desactiva
la salida digital anteriormente mencionada
    digitalWrite(timbre00,
LOW);
   }
  }
  //En este caso ocurre del mismo modo que el bucle if anterior
  if
(strcmp(topic,"home/entrada/lampara0")==0){
   if ((char)payload[1] == 'N'){
    digitalWrite(lamp00, HIGH);
   }else{
    digitalWrite(lamp00, LOW);
   }
  }
  if (strcmp(topic,"home/entrada/lampara1")==0){
   if ((char)payload[1] == 'N'){
    digitalWrite(lamp10, HIGH);
   }else{
    digitalWrite(lamp10, LOW);
   }
  }
  if (strcmp(topic,"home/entrada/lampara2")==0){
   if ((char)payload[1] == 'N'){
    digitalWrite(lamp11, HIGH);
   }else{
    digitalWrite(lamp11, LOW);
   }
  }
  //En este topic recibimos la temperatura, por lo que necesitamos
transformar el mensaje
  //en algo que pueda interpretar el microcontrolador
  if
(strcmp(topic,"home/entrada/calefaccion")==0){
   str =
"";
   for (int i = 0;i < length; i++){
    mat[i] = payload[i];
   }
   str0 = mat[0];
   str1 = mat[1];
   str2 = mat[2];
```

```
    str3 = mat[3];
    str4 = mat[4];
    //concatenamos los caracteres que creemos convenientes
    str = String(str + str0 + str1 + str2 + str3 + str4);
    //y convertimos esa variable String en un float
    temperatura = str.toFloat();
    //Dependiendo de la temperatura recibida se activará o desactivará
la calefacción
    if (temperatura<15){
     digitalWrite(kEnt, HIGH);
    }if (temperatura>20){
     digitalWrite(kEnt, LOW);
    }
   }
}

//Esta función se utiliza para que el microcontrolador se mantenga
siempre conectado al servidor MQTT
void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Create a random client ID
    String clientId = "ESP8266Client-";
    clientId += String(random(0xffff), HEX);
    // Attempt to connect
    if (client.connect(clientId.c_str())) {
      Serial.println("connected");
      //Una vez conectado, envia un mensaje


      //Nos suscribimos a los topics de entrada
      client.subscribe("home/entrada/timbre");
      client.subscribe("home/entrada/lampara0");
      client.subscribe("home/entrada/lampara1");
      client.subscribe("home/entrada/lampara2");
      client.subscribe("home/entrada/calefaccion");

    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}

void setup() {
  //Definimos que GPIOs son salidas
  pinMode(lamp00, OUTPUT);
  pinMode(lamp10, OUTPUT);
  pinMode(lamp11, OUTPUT);
  pinMode(timbre00, OUTPUT);
  pinMode(kEnt, OUTPUT);

  //Definimos que GPIOs son entradas
  pinMode(pres00, INPUT);
  pinMode(pres10, INPUT);
  pinMode(pul00, INPUT);
  pinMode(pul10, INPUT);
```

```
  pinMode(pul11, INPUT);

  //Definimos la velocidad de transmissión del puerto serie
  Serial.begin(115200);
  //Llamamos a la función que conecta con la red WiFi de la vivienda
  setup_wifi();
  //Definimos el puerto que tiene abierto el servidor para la
comunicación MQTT
  client.setServer(mqtt_server, 1883);
  //Definimos la función que recibe los mensajes de entrada
  client.setCallback(callback);
}

void loop() {

  if (!client.connected()) {
    reconnect();
  }
  client.loop();

  //Leemos el estado del GPIO
  presencia00 = digitalRead(pres00);
  presencia10 = digitalRead(pres10);
  pulsador00 = digitalRead(pul00);
  pulsador10 = digitalRead(pul10);
  pulsador11 = digitalRead(pul11);


  //Debido a que no queremos enviar un mensaje cada vez que el loop()
se ejecute
  //establecemos un tiempo para la lectura
  long now = millis();
  if (now - lastMsg > 1000) {
    lastMsg = now;
    if (presencia00 == HIGH){
      //Definimos el mensaje MQTT
      snprintf(msg, 50, "ON");
      Serial.print("Publish message: ");
      Serial.println(msg);
      //Enviamos el mensaje MQTT a determinado topic
      client.publish("home/entrada/presencia0", msg);
    }else{
      snprintf(msg, 50, "OFF");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/entrada/presencia0", msg);
    }
    if (presencia00 == HIGH){
      snprintf(msg, 50, "ON");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/entrada/presencia1", msg);
   }else{
      snprintf(msg, 50, "OFF");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/entrada/presencia1", msg);
   }
   if (presencia00 == HIGH){
      snprintf(msg, 50, "ON");
      Serial.print("Publish message: ");
```

```
      Serial.println(msg);
      client.publish("home/entrada/presencia1", msg);
  }else{
      snprintf(msg, 50, "OFF");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/entrada/presencia1", msg);
  }
  if (pulsador00 == HIGH){
     snprintf (msg, 50, "ON");
     Serial.print("Publish message: ");
     Serial.println(msg);
     client.publish("home/entrada/pulsador0", msg);
   }else{
     snprintf (msg, 50, "OFF");
     Serial.print("Publish message: ");
     Serial.println(msg);
     client.publish("home/entrada/pulsador0", msg);
   }
 }
   //En este caso cada vez que se pulsa el pulsador cambia de estado
   if (pulsador10 == HIGH){
    contador1++;
    delay(200);
    if (contador1 % 2 == 0){
      snprintf (msg, 50, "ON");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/entrada/pulsador1", msg);
    }if (contador1 % 2 == 1){
      snprintf (msg, 50, "OFF");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/entrada/pulsador1", msg);
    }
   }
   if (pulsador11 == HIGH){
    contador2++;
    delay(200);
    if (contador % 2 == 0){
      snprintf (msg, 50, "ON");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/entrada/pulsador2", msg);
      }if (contador2 % 2 == 1){
      snprintf (msg3, 50, "OFF");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/entrada/pulsador2", msg);
      }
   }
}
```

3.4.2 Programación ESP32 Habitación1

```
#include <WiFi.h>
#include <PubSubClient.h>

const char* ssid = "Nombre de la red wifi";
//nombre de la red wifi de la vivienda
const char* password = "contraseña de la red wifi";
//contraseña de la red wifi anteriormente mencionada
const char* mqtt_server = "192.168.1.39";
//dirección ip del broker mqtt

const int pul20 = 34;
//GPIO al que está conectado el pulsador pul2.0
const int pul21 = 35;
//GPIO al que está conectado el pulsador pul2.1
const int pul22 = 32;
//GPIO al que está conectado el pulsador pul2.2
const int pul23 = 33;
//GPIO al que está conectado el pulsador pul2.3
const int temp20 = 25;
//GPIO al que está conectado al sensor de temperatura temp2.0

const int lamp20 = 21;
//GPIO al que está conectado la lámpara lamp2.0
const int k1 = 19;
//GPIO al que está conectado el relé K1

int contador1 = 0;
//Esta variable se utiliza para cambiar de estado a cada pulsación de
los pulsadores
int pulsador20 = 0;
//Esta variable contiene el valor de la entrada digital a la que está
conectado el pulsador pul2.0
int pulsador21 = 0;
//Esta variable contiene el valor de la entrada digital a la que está
conectado el pulsador pul2.1
int pulsador22 = 0;
//Esta variable contiene el valor de la entrada digital a la que está
conectado el pulsador pul2.2
int pulsador23 = 0;
//Esta variable contiene el valor de la entrada digital a la que está
conectado el pulsador pul2.3
int temp20val = 0;
//Esta variable contiene el valor de la entrada analógica del sensor
de temperatura

//El siguiente grupo de variables se utiliza para primero convertir el
meensaje que entra en una cadena y segundo
//convertir esta cadena en una variable float
String str;
char str0;
char str1;
char str2;
char str3;
char str4;
byte mat[10];
float temperatura;

//Las siguientes variables se utilizan para leer la temperatura del
sensor
```

```
float temp;
float temperaturaMedida;

WiFiClient espClient;
//Crea un cliente que se puede conectar a una determinada dirección IP
definida en
client.connect()

PubSubClient client(espClient);
//Crea un cliente para publicar y suscribirse a través de un
determinado cliente WiFi

char msg[50];
//Se declara la variable que contiene el mensaje que se envía por
MQTT
long lastMsg = 0;

//Esta función establece conexión con la red WiFi de la vivienda
void setup_wifi()
{

  delay(10);
  // We start by connecting to a WiFi network
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  randomSeed(micros());

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

//Esta función es la que se encarga de recibir e interpretar los
mensajes MQTT de los topics suscritos
void callback(char* topic, byte* payload, unsigned int length)
{
  if (strcmp(topic,"home/hab1/calefaccion")==0){
       str = "";
       for (int i = 0; i < length; i++){
         mat[i] = payload[i];
       }
       str0 = mat[0];
       str1 = mat[1];
       str2 = mat[2];
       str3 = mat[3];
       str4 = mat[4];
       str = String(str + str0 + str1 + str2 + str3 + str4);
       temperatura = str.toFloat();
       if (temperatura<15){
          digitalWrite(k1, HIGH);
       }if(temperatura>20){
```

```
        digitalWrite(k1, LOW);
      }
    }
  if (strcmp(topic, "home/hab1/lampara1")==0){
    if ((char)payload[1] == 'N'){
    digitalWrite(lamp20, HIGH);
   }else{
    digitalWrite(lamp20, LOW);
   }
  }
}
//Esta función se utiliza para que el microcontrolador se mantenga
siempre conectado al servidor MQTT
void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Create a random client ID
    String clientId = "ESP8266Client-";
    clientId += String(random(0xffff), HEX);
    // Attempt to connect
    if (client.connect(clientId.c_str())) {
      Serial.println("connected");
      //Una vez conectado, envia un mensaje


      //Nos suscribimos a los topics de entrada
      client.subscribe("home/hab1/lampara1");
      client.subscribe("home/hab1/calefaccion");

    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}

void setup() {
  //Definimos que GPIOs son salidas
  pinMode(lamp20, OUTPUT);
  pinMode(k1, OUTPUT);

  //Definimos que GPIOs son entradas
  pinMode(pul20, INPUT);
  pinMode(pul21, INPUT);
  pinMode(pul22, INPUT);
  pinMode(pul23, INPUT);

  //Definimos la velocidad de transmissión del puerto serie
  Serial.begin(115200);
  //Llamamos a la función que conecta con la red WiFi de la vivienda
  setup_wifi();
  //Definimos el puerto que tiene abierto el servidor para la
comunicación MQTT
  client.setServer(mqtt_server, 1883);
  //Definimos la función que recibe los mensajes de entrada
  client.setCallback(callback);
}
```

```
void loop(){

  if (!client.connected()) {
    reconnect();
  }
  client.loop();

  //Leemos el estado del GPIO
  pulsador20 = digitalRead(pul20);
  pulsador21 = digitalRead(pul21);
  pulsador22 = digitalRead(pul22);
  pulsador23 = digitalRead(pul23);

  //Leemos el valor del sensor de temperatura

  temp20val = analogRead(temp20);
  temp = map(temp20, 0, 4095, 0, 5000);
  temperaturaMedida = temp/100;

  //Debido a que no queremos enviar un mensaje cada vez que el loop()
se ejecute
  //establecemos un tiempo para la lectura
  long now = millis();
  if (now - lastMsg > 4000) {
    lastMsg = now;
    snprintf (msg, 50, "%f", temperaturaMedida);
    Serial.print("Publish message: ");
    Serial.println(msg);
    client.publish("home/hab1/temperatura", msg);
  }
  if (pulsador20 == HIGH){
    contador1++;
    delay(200);
    if (contador1 % 2 == 0){
      snprintf (msg, 50, "ON");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/hab1/pulsador0", msg);
    }if (contador1 % 2 == 1){
      snprintf (msg, 50, "OFF");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/hab1/pulsador0", msg);
    }
  }
  if (pulsador21 == HIGH){
    contador1++;
    delay(200);
    if (contador1 % 2 == 0){
      snprintf (msg, 50, "ON");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/hab1/pulsador1", msg);
    }if (contador1 % 2 == 1){
      snprintf (msg, 50, "OFF");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/hab1/pulsador1", msg);
    }
  }
```

```
  if (pulsador22 == HIGH){
    contador1++;
    delay(200);
    if (contador1 % 2 == 0){
      snprintf (msg, 50, "ON");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/hab1/pulsador2", msg);
    }if (contador1 % 2 == 1){
      snprintf (msg, 50, "OFF");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/hab1/pulsador2", msg);
    }
  }
  if (pulsador23 == HIGH){
    contador1++;
    delay(200);
    if (contador1 % 2 == 0){
      snprintf (msg, 50, "ON");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/hab1/pulsador3", msg);
    }if (contador1 % 2 == 1){
      snprintf (msg, 50, "OFF");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/hab1/pulsador3", msg);
    }
  }
}
```

3.4.3 Programación ESP32 Baño1

```
#include <WiFi.h>
#include <PubSubClient.h>

const char* ssid = "Nombre de la red wifi";
//nombre de la red wifi de la vivienda
const char* password = "contraseña de la red wifi";
//contraseña de la red wifi anteriormente mencionada
const char* mqtt_server = "192.168.1.39";
//dirección ip del broker mqtt

const int pul30 = 34;
//GPIO al que está conectado el pulsador pul2.0
const int pul31 = 35;
//GPIO al que está conectado el pulsador pul2.1
const int pul32 = 32;
//GPIO al que está conectado el pulsador pul2.2
const int pul33 = 33;
//GPIO al que está conectado el pulsador pul2.3
const int temp30 = 25;
//GPIO al que está conectado al sensor de temperatura

const int lamp30 = 21;
//GPIO al que está conectado la lámpara lamp2.0
const int lamp31 = 19;
//GPIO al que está conectado la lámpara lamp2.0
const int lamp32 = 18;
//GPIO al que está conectado la lámpara lamp2.0
const int k2 = 17;
//GPIO al que está conectado el relé K1

int contador1 = 0;
//Esta variable se utiliza para cambiar de estado a cada pulsación de
los pulsadores
int contador2 = 0;
//Esta variable se utiliza para cambiar de estado a cada pulsación de
los pulsadores
int contador3 = 0;
//Esta variable se utiliza para cambiar de estado a cada pulsación de
los pulsadores
int pulsador30 = 0;
//Esta variable contiene el valor de la entrada digital a la que está
conectado el pulsador pul2.0
int pulsador31 = 0;
//Esta variable contiene el valor de la entrada digital a la que está
conectado el pulsador pul2.1
int pulsador32 = 0;
//Esta variable contiene el valor de la entrada digital a la que está
conectado el pulsador pul2.2
int pulsador33 = 0;
//Esta variable contiene el valor de la entrada digital a la que está
conectado el pulsador pul2.3
int temp30val = 0;
//Esta variable contiene el valor de la entrada analógica a la que
está conectado el sensor de temperatura temp4.0

//El siguiente grupo de variables se utiliza para primero convertir el
meensaje que entra en una cadena y segundo
//convertir esta cadena en una variable float
String str;
```

```
char str0;
char str1;
char str2;
char str3;
char str4;
byte mat[10];
float temperatura;

//Las siguientes variables se utilizan para leer la temperatura del
sensor
float temp;
float temperaturaMedida;

WiFiClient espClient;
//Crea un cliente que se puede conectar a una determinada dirección IP
definida en
client.connect()

PubSubClient client(espClient);
//Crea un cliente para publicar y suscribirse a través de un
determinado cliente WiFi

char msg[50];
//Se declara la variable que contiene el mensaje que se envía por
MQTT
long lastMsg = 0;

//Esta función establece conexión con la red WiFi de la vivienda
void setup_wifi()
{

  delay(10);
  // We start by connecting to a WiFi network
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  randomSeed(micros());

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

//Esta función es la que se encarga de recibir e interpretar los
mensajes MQTT de los topics suscritos
void callback(char* topic, byte* payload, unsigned int length)
{
  if (strcmp(topic,"home/bano/calefaccion")==0){
      str = "";
      for (int i = 0; i < length; i++){
        mat[i] = payload[i];
      }
```

```
      str0 = mat[0];
      str1 = mat[1];
      str2 = mat[2];
      str3 = mat[3];
      str4 = mat[4];
      str = String(str + str0 + str1 + str2 + str3 + str4);
      temperatura = str.toFloat();
      if (temperatura<15){
         digitalWrite(k2, HIGH);
      }if(temperatura>20){
        digitalWrite(k2, LOW);
      }
    }
  if (strcmp(topic, "home/bano/lampara0")==0){
    if ((char)payload[1] == 'N'){
    digitalWrite(lamp30, HIGH);
   }else{
    digitalWrite(lamp30, LOW);
   }
  }
  if (strcmp(topic, "home/bano/lampara1")==0){
    if ((char)payload[1] == 'N'){
    digitalWrite(lamp31, HIGH);
   }else{
    digitalWrite(lamp31, LOW);
   }
  }
  if (strcmp(topic, "home/bano/lampara2")==0){
    if ((char)payload[1] == 'N'){
    digitalWrite(lamp32, HIGH);
   }else{
    digitalWrite(lamp32, LOW);
   }
  }
}

//Esta función se utiliza para que el microcontrolador se mantenga
siempre conectado al servidor MQTT
void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Create a random client ID
    String clientId = "ESP8266Client-";
    clientId += String(random(0xffff), HEX);
    // Attempt to connect
    if (client.connect(clientId.c_str())) {
      Serial.println("connected");
      //Una vez conectado, envia un mensaje


      //Nos suscribimos a los topics de entrada
      client.subscribe("home/bano1/lampara0");
      client.subscribe("home/bano1/lampara1");
      client.subscribe("home/bano1/lampara2");
      client.subscribe("home/bano1/calefaccion");

    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
```

```
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}

void setup() {
  //Definimos que GPIOs son salidas
  pinMode(lamp30, OUTPUT);
  pinMode(lamp31, OUTPUT);
  pinMode(lamp32, OUTPUT);
  pinMode(k2, OUTPUT);

  //Definimos que GPIOs son entradas
  pinMode(pul30, INPUT);
  pinMode(pul31, INPUT);
  pinMode(pul32, INPUT);
  pinMode(pul33, INPUT);

  //Definimos la velocidad de transmissión del puerto serie
  Serial.begin(115200);
  //Llamamos a la función que conecta con la red WiFi de la vivienda
  setup_wifi();
  //Definimos el puerto que tiene abierto el servidor para la
comunicación MQTT
  client.setServer(mqtt_server, 1883);
  //Definimos la función que recibe los mensajes de entrada
  client.setCallback(callback);
}

void loop(){

  if (!client.connected()) {
    reconnect();
  }
  client.loop();

  //Leemos el estado del GPIO
  pulsador30 = digitalRead(pul30);
  pulsador31 = digitalRead(pul31);
  pulsador32 = digitalRead(pul32);
  pulsador33 = digitalRead(pul33);

  //Leemos el valor del sensor de temperatura
  temp30val = analogRead(temp30);
  temp = map(temp30, 0, 4095, 0, 5000);
  temperaturaMedida = temp/100;


//Debido a que no queremos enviar un mensaje cada vez que el loop() se
ejecute
  //establecemos un tiempo para la lectura
  long now = millis();
  if (now - lastMsg > 4000) {
    lastMsg = now;
    snprintf (msg, 50, "%f", temperaturaMedida);
    Serial.print("Publish message: ");
    Serial.println(msg);
    client.publish("home/bano1/temperatura", msg);
  }
  if (pulsador30 == HIGH){
```

```
    contador1++;
    delay(200);
    if (contador1 % 2 == 0){
      snprintf (msg, 50, "ON");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/bano1/pulsador0", msg);
    }if (contador1 % 2 == 1){
      snprintf (msg, 50, "OFF");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/bano1/pulsador0", msg);
    }
  }
  if (pulsador31 == HIGH){
    contador1++;
    delay(200);
    if (contador1 % 2 == 0){
      snprintf (msg, 50, "ON");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/bano1/pulsador1", msg);
    }if (contador1 % 2 == 1){
      snprintf (msg, 50, "OFF");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/bano1/pulsador1", msg);
    }
  }
  if (pulsador32 == HIGH){
    contador2++;
    delay(200);
    if (contador2 % 2 == 0){
      snprintf (msg, 50, "ON");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/bano1/pulsador2", msg);
    }if (contador2 % 2 == 1){
      snprintf (msg, 50, "OFF");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/bano1/pulsador2", msg);
    }
  }
  if (pulsador33 == HIGH){
    contador2++;
    delay(200);
    if (contador2 % 2 == 0){
      snprintf (msg, 50, "ON");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/bano1/pulsador3", msg);
    }if (contador2 % 2 == 1){
      snprintf (msg, 50, "OFF");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/bano1/pulsador3", msg);
    }
  }
}
```

3.4.4 Programación ESP32 Habitación2

```
#include <WiFi.h>
#include <PubSubClient.h>

const char* ssid = "Nombre de la red wifi";
//nombre de la red wifi de la vivienda
const char* password = "contraseña de la red wifi";
//contraseña de la red wifi anteriormente mencionada
const char* mqtt_server = "192.168.1.39";
//dirección ip del broker mqtt

const int pul40 = 34;
//GPIO al que está conectado el pulsador pul2.0
const int pul41 = 35;
//GPIO al que está conectado el pulsador pul2.1
const int pul42 = 32;
//GPIO al que está conectado el pulsador pul2.2
const int pul43 = 33;
//GPIO al que está conectado el pulsador pul2.3
const int temp40 = 25;
//GPIO al que está conectado al sensor de temperatura

const int lamp40 = 21;
//GPIO al que está conectado la lámpara lamp2.0
const int k3 = 19;
//GPIO al que está conectado el relé K1

int contador1 = 0;
//Esta variable se utiliza para cambiar de estado a cada pulsación de
los pulsadores
int pulsador40 = 0;
//Esta variable contiene el valor de la entrada digital a la que está
conectado el pulsador pul2.0
int pulsador41 = 0;
//Esta variable contiene el valor de la entrada digital a la que está
conectado el pulsador pul2.1
int pulsador42 = 0;
//Esta variable contiene el valor de la entrada digital a la que está
conectado el pulsador pul2.2
int pulsador43 = 0;
//Esta variable contiene el valor de la entrada digital a la que está
conectado el pulsador pul2.3
int temp40val = 0;
//Esta variable contiene el valor de la entrada analógica a la que
está conectado el sensor de temperatura temp4.0

//El siguiente grupo de variables se utiliza para primero convertir el
meensaje que entra en una cadena y segundo
//convertir esta cadena en una variable float
String str;
char str0;
char str1;
char str2;
char str3;
char str4;
byte mat[10];
float temperatura;

//Las siguientes variables se utilizan para leer la temperatura del
sensor
```

```
float temp;
float temperaturaMedida;

WiFiClient espClient;
//Crea un cliente que se puede conectar a una determinada dirección IP
definida en
client.connect()

PubSubClient client(espClient);
//Crea un cliente para publicar y suscribirse a través de un
determinado cliente WiFi

char msg[50];
//Se declara la variable que contiene el mensaje que se envía por
MQTT
long lastMsg = 0;

//Esta función establece conexión con la red WiFi de la vivienda
void setup_wifi()
{

  delay(10);
  // We start by connecting to a WiFi network
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  randomSeed(micros());

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

//Esta función es la que se encarga de recibir e interpretar los
mensajes MQTT de los topics suscritos
void callback(char* topic, byte* payload, unsigned int length)
{
  if (strcmp(topic,"home/hab2/calefaccion")==0){
       str = "";
       for (int i = 0; i < length; i++){
         mat[i] = payload[i];
       }
       str0 = mat[0];
       str1 = mat[1];
       str2 = mat[2];
       str3 = mat[3];
       str4 = mat[4];
       str = String(str + str0 + str1 + str2 + str3 + str4);
       temperatura = str.toFloat();
       if (temperatura<15){
          digitalWrite(K3, HIGH);
       }if(temperatura>20){
```

```
        digitalWrite(K3, LOW);
      }
    }
  if (strcmp(topic, "home/hab2/lampara1")==0){
    if ((char)payload[1] == 'N'){
    digitalWrite(lamp40, HIGH);
   }else{
    digitalWrite(lamp40, LOW);
   }
  }
}
//Esta función se utiliza para que el microcontrolador se mantenga
siempre conectado al servidor MQTT
void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Create a random client ID
    String clientId = "ESP8266Client-";
    clientId += String(random(0xffff), HEX);
    // Attempt to connect
    if (client.connect(clientId.c_str())) {
      Serial.println("connected");
      //Una vez conectado, envia un mensaje


      //Nos suscribimos a los topics de entrada
      client.subscribe("home/hab2/lampara1");
      client.subscribe("home/hab2/calefaccion");

    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}

void setup() {
  //Definimos que GPIOs son salidas
  pinMode(lamp40, OUTPUT);
  pinMode(k3, OUTPUT);

  //Definimos que GPIOs son entradas
  pinMode(pul40, INPUT);
  pinMode(pul41, INPUT);
  pinMode(pul42, INPUT);
  pinMode(pul43, INPUT);

  //Definimos la velocidad de transmissión del puerto serie
  Serial.begin(115200);
  //Llamamos a la función que conecta con la red WiFi de la vivienda
  setup_wifi();
  //Definimos el puerto que tiene abierto el servidor para la
comunicación MQTT
  client.setServer(mqtt_server, 1883);
  //Definimos la función que recibe los mensajes de entrada
  client.setCallback(callback);
}
```

```
void loop(){

  if (!client.connected()) {
    reconnect();
  }
  client.loop();

  //Leemos el estado del GPIO
  pulsador40 = digitalRead(pul40);
  pulsador41 = digitalRead(pul41);
  pulsador42 = digitalRead(pul42);
  pulsador43 = digitalRead(pul43);

  //Leemos el valor del sensor de temperatura
  temp40val = analogRead(temp40);
  temp = map(temp40, 0, 4095, 0, 5000);
  temperaturaMedida = temp/100;

  //Debido a que no queremos enviar un mensaje cada vez que el loop()
se ejecute
  //establecemos un tiempo para la lectura
  long now = millis();
  if (now - lastMsg > 4000) {
    lastMsg = now;
    snprintf (msg, 50, "%f", temperaturaMedida);
    Serial.print("Publish message: ");
    Serial.println(msg);
    client.publish("home/hab2/temperatura", msg);
  }
  if (pulsador40 == HIGH){
    contador1++;
    delay(200);
    if (contador1 % 2 == 0){
      snprintf (msg, 50, "ON");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/hab2/pulsador0", msg);
    }if (contador1 % 2 == 1){
      snprintf (msg, 50, "OFF");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/hab2/pulsador0", msg);
    }
  }
  if (pulsador41 == HIGH){
    contador1++;
    delay(200);
    if (contador1 % 2 == 0){
      snprintf (msg, 50, "ON");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/hab2/pulsador1", msg);
    }if (contador1 % 2 == 1){
      snprintf (msg, 50, "OFF");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/hab2/pulsador1", msg);
    }
  }
  if (pulsador42 == HIGH){
```

```
    contador1++;
    delay(200);
    if (contador1 % 2 == 0){
      snprintf (msg, 50, "ON");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/hab2/pulsador2", msg);
    }if (contador1 % 2 == 1){
      snprintf (msg, 50, "OFF");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/hab2/pulsador2", msg);
    }
  }
  if (pulsador43 == HIGH){
    contador1++;
    delay(200);
    if (contador1 % 2 == 0){
      snprintf (msg, 50, "ON");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/hab2/pulsador3", msg);
    }if (contador1 % 2 == 1){
      snprintf (msg, 50, "OFF");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/hab2/pulsador3", msg);
    }
  }
}
```

3.4.5 Programación ESP32 Comedor

```
#include <WiFi.h>
#include <PubSubClient.h>

const char* ssid = "Nombre de la red wifi";
//nombre de la red wifi de la vivienda
const char* password = "contraseña de la red wifi";
//contraseña de la red wifi anteriormente mencionada
const char* mqtt_server = "192.168.1.39";
//dirección ip del broker mqtt

const int temp50 = 16;
//GPIO al que está conectado al sensor de temperatura
const int temp51 = 17;
//GPIO al que está conectado al sensor de temperatura
const int pul50 = 35;
//GPIO al que está conectado el pulsador pul5.0
const int pul51 = 34;
//GPIO al que está conectado el pulsador pul5.1
const int pul52 = 33;
//GPIO al que está conectado el pulsador pul5.2
const int pulPerUp50 = 32;
//GPIO al que está conectado el pulsador pulPerUp5.0
const int pulPerDown50 = 27;
//GPIO al que está conectado el pulsador pulPerDown5.0
const int pulPerUp51 = 26;
//GPIO al que está conectado el pulsador pulPerUp5.1
const int pulPerDown51 = 25;
//GPIO al que está conectado el pulsador pulPerDown5.1
const int pulPerUp52 = 23;
//GPIO al que está conectado el pulsador pulPerUp5.2
const int pulPerDown52 = 22;
//GPIO al que está conectado el pulsador pulPerDown5.2
const int pul60 = 21;
//GPIO al que está conectado el pulsador pul6.0
const int pul61 = 19;
//GPIO al que está conectado el pulsador pul6.1
const int pul62 = 18;
//GPIO al que está conectado el pulsador pul6.2
const int pul63 = 4;
//GPIO al que está conectado el pulsador pul6.3

int pulsador50 = 0;
//Esta variable contiene el valor de la entrada digital a la que está
conectado el pulsador pul5.0
int pulsador51 = 0;
//Esta variable contiene el valor de la entrada digital a la que está
conectado el pulsador pul5.1
int pulsador52 = 0;
//Esta variable contiene el valor de la entrada digital a la que está
conectado el pulsador pul5.2
int pulsador60 = 0;
//Esta variable contiene el valor de la entrada digital a la que está
conectado el pulsador pul6.0
int pulsador61 = 0;
//Esta variable contiene el valor de la entrada digital a la que está
conectado el pulsador pul6.1
int pulsador62 = 0;
//Esta variable contiene el valor de la entrada digital a la que está
conectado el pulsador pul6.2
```

```
int pulsador63 = 0;
//Esta variable contiene el valor de la entrada digital a la que está
conectado el pulsador pul6.3
int pulsadorPerUp50 = 0;
//Esta variable contiene el valor de la entrada digital a la que está
conectado el pulsador pulPerUp50
int pulsadorPerDown50 = 0;
//Esta variable contiene el valor de la entrada digital a la que está
conectado el pulsador pulPerDown50
int pulsadorPerUp51 = 0;
//Esta variable contiene el valor de la entrada digital a la que está
conectado el pulsador pulPerUp51
int pulsadorPerDown51 = 0;
//Esta variable contiene el valor de la entrada digital a la que está
conectado el pulsador pulPerDown51
int pulsadorPerUp52 = 0;
//Esta variable contiene el valor de la entrada digital a la que está
conectado el pulsador pulPerUp52
int pulsadorPerDown52 = 0;
//Esta variable contiene el valor de la entrada digital a la que está
conectado el pulsador pulPerDown52
int contador1 = 0;
//Esta variable es usada para conocer el estado del pulsador pul5.0
int contador2 = 0;
//Esta variable es usada para conocer el estado del pulsador pul5.1
int contador3 = 0;
//Esta variable es usada para conocer el estado del pulsador pul5.2
int contador4 = 0;
//Esta variable es usada para conocer el estado del pulsador pul6.0
int contador5 = 0;
//Esta variable es usada para conocer el estado del pulsador pul6.1
int contador6 = 0;
//Esta variable es usada para conocer el estado del pulsador pul6.2
int contador7 = 0;
//Esta variable es usada para conocer el estado del pulsador pul6.3


const int lamp50 = 14;
//GPIO al que está conectado la lámpara lamp5.0
const int lamp51 = 15;
//GPIO al que está conectado la lámpara lamp5.1

//Propiedades de las salidas PWM
const int freq = 5000;
const int dimChannel1 = 0;
const int dimChannel2 = 1;
const int resolution = 8;

//Variables usadas para la lectura y conversión de la señal del sensor
de temperatura temp5.0
float temp_50;
float tempVal_50;
float temperaturaMedida50;

//Variables usadas para la lectura y conversión de la señal del sensor
de temperatura temp5.0
float temp_51;
float tempVal_51;
float temperaturaMedida51;

//Variables usadas para la lectura del mensaje que se recibe por MQTT
```

```
String str50;
char str50_0;
char str50_1;
char str50_2;
char str50_3;
char str50_4;
byte mat50[10];
float temperatura50;

//Variables usadas para la lectura del mensaje que se recibe por MQTT
String str51;
char str51_0;
char str51_1;
char str51_2;
char str51_3;
char str51_4;
byte mat51[10];
float temperatura51;

WiFiClient espClient;
//Crea un cliente que se puede conectar a una determinada dirección IP
definida en
client.connect()

PubSubClient client(espClient);
//Crea un cliente para publicar y suscribirse a través de un
determinado cliente WiFi

char msg[50];
//Se declara la variable que contiene el mensaje que se envía por
MQTT
long lastMsg = 0;

//Esta función establece conexión con la red WiFi de la vivienda
void setup_wifi()
{

  delay(10);
  // We start by connecting to a WiFi network
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  randomSeed(micros());

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

//Esta función es la que se encarga de recibir e interpretar los
mensajes MQTT de los topics suscritos
```

```
void callback(char* topic, byte* payload, unsigned int length)
{
  if (strcmp(topic,"home/comedor/lampara0")==0){
    String str;
    byte mat[2];
    for (int i = 0; i < length; i++){
      mat[i] = payload[i];
    }
    char str50_0 = mat50[0];
    char str50_1 = mat50[1];
    char str50_2 = mat50[2];
    str50 = String(str50 + str50_0 + str50_1 + str50_2);
    int porcentaje = str50.toInt();
    switch (porcentaje){
      case 0:
        ledcWrite(dimChannel1, 0);
      break;
      case 25:
        ledcWrite(dimChannel1,63.75);
      break;
      case 50:
        ledcWrite(dimChannel1, 127.5);
      break;
      case 75:
        ledcWrite(dimChannel1, 192.25);
      break;
      case 10:
        ledcWrite(dimChannel1, 255);
      break;
    }
  }
  if (strcmp(topic,"home/comedor/lampara1")==0){
    String str;
    byte mat[2];
    for (int i = 0; i < length; i++){
      mat[i] = payload[i];
    }
    char str51_0 = mat51[0];
    char str51_1 = mat51[1];
    char str51_2 = mat51[2];
    str = String(str51 + str51_0 + str51_1 + str51_2);
    int porcentaje = str51.toInt();
    switch (porcentaje){
      case 0:
        ledcWrite(dimChannel2, 0);
      break;
      case 25:
        ledcWrite(dimChannel2,63.75);
      break;
      case 50:
        ledcWrite(dimChannel2, 127.5);
      break;
      case 75:
        ledcWrite(dimChannel2, 192.25);
      break;
      case 10:
        ledcWrite(dimChannel2, 255);
      break;
    }
  }
}
```

```
//Esta función se utiliza para que el microcontrolador se mantenga
siempre conectado al servidor MQTT
void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Create a random client ID
    String clientId = "ESP8266Client-";
    clientId += String(random(0xffff), HEX);
    // Attempt to connect
    if (client.connect(clientId.c_str())) {
      Serial.println("connected");
      //Una vez conectado, envia un mensaje


      //Nos suscribimos a los topics de entrada
      client.subscribe("home/comedor/lampara1");
      client.subscribe("home/comedor/lampara1");

    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}

void setup() {
  //Definimos que GPIOs son salidas PWM
  ledcSetup(dimChannel1, freq, resolution);
  ledcAttachPin(lamp50, dimChannel1);
  ledcSetup(dimChannel2, freq, resolution);
  ledcAttachPin(lamp51, dimChannel2);


  //Definimos que GPIOs son entradas
  pinMode(pul50, INPUT);
  pinMode(pul51, INPUT);
  pinMode(pul52, INPUT);
  pinMode(pul60, INPUT);
  pinMode(pul61, INPUT);
  pinMode(pul62, INPUT);
  pinMode(pul63, INPUT);
  pinMode(pulPerUp50, INPUT);
  pinMode(pulPerUp51, INPUT);
  pinMode(pulPerUp52, INPUT);
  pinMode(pulPerDown50, INPUT);
  pinMode(pulPerDown51, INPUT);
  pinMode(pulPerDown52, INPUT);

  //Definimos la velocidad de transmisión del puerto serie
  Serial.begin(115200);
  //Llamamos a la función que conecta con la red WiFi de la vivienda
  setup_wifi();
  //Definimos el puerto que tiene abierto el servidor para la
comunicación MQTT
  client.setServer(mqtt_server, 1883);
  //Definimos la función que recibe los mensajes de entrada
```

```
  client.setCallback(callback);
}

void loop(){
  if (!client.connected()){
    reconnect();
  }
  client.loop();

  //Leemos el estado del GPIO
  pulsador50 = digitalRead(pul50);
  pulsador51 = digitalRead(pul51);
  pulsador52 = digitalRead(pul52);
  pulsador60 = digitalRead(pul60);
  pulsador61 = digitalRead(pul61);
  pulsador62 = digitalRead(pul62);
  pulsadorPerUp50 = digitalRead(pulPerUp50);
  pulsadorPerDown50 = digitalRead(pulPerDown50);
  pulsadorPerUp51 = digitalRead(pulPerUp51);
  pulsadorPerDown51 = digitalRead(pulPerDown51);
  pulsadorPerUp52 = digitalRead(pulPerUp52);
  pulsadorPerDown52 = digitalRead(pulPerDown52);

  tempVal_50 = analogRead(temp50);
  temp_50 = map(tempVal_50, 0, 4095, 0, 5000);
  temperatura50 = temp_50/100;

  tempVal_51 = analogRead(temp51);
  temp_51 = map(tempVal_51, 0, 4095, 0, 5000);
  temperatura51 = temp_51/100;

  if (pulsador50 == HIGH){
    contador1++;
    delay(200);
    if (contador1 % 2 == 0){
      snprintf (msg, 50, "ON");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/comedor/pulsador0", msg);
    }if (contador1 % 2 == 1){
      snprintf (msg, 50, "OFF");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/comedor/pulsador0", msg);
    }
  }
  if (pulsador51 == HIGH){
    contador2++;
    if (contador2 == 5){
      contador2 = 0;
    }
    delay(200);
    switch (contador2){
      case 0:
        snprintf (msg, 50, "0");
        Serial.print("Publish message: ");
        Serial.println(msg);
        client.publish("home/comedor/pulsador1", msg);
      break;
      case 1:
        snprintf (msg, 50, "25");
```

```
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/comedor/pulsador1", msg);
    break;
    case 2:
      snprintf (msg, 50, "50");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/comedor/pulsador1", msg);
    break;
    case 3:
      snprintf (msg, 50, "75");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/comedor/pulsador1", msg);
    break;
    case 4:
      snprintf (msg, 50, "100");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/comedor/pulsador1", msg);
    break;
  }
}
if (pulsador52 == HIGH){
  contador3++;
  if (contador3 == 5){
    contador3 = 0;
  }
  delay(200);
  switch (contador3){
    case 0:
      snprintf (msg, 50, "0");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/comedor/pulsador2", msg);
    break;
    case 1:
      snprintf (msg, 50, "25");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/comedor/pulsador2", msg);
    break;
    case 2:
      snprintf (msg, 50, "50");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/comedor/pulsador2", msg);
    break;
    case 3:
      snprintf (msg, 50, "75");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/comedor/pulsador2", msg);
    break;
    case 4:
      snprintf (msg, 50, "100");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/comedor/pulsador2", msg);
    break;
```

```
    }
  }
  if (pulsador60 == HIGH){
    contador4++;
    delay(200);
    if (contador4 % 2 == 0){
      snprintf (msg, 50, "ON");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/patio/pulsador0", msg);
    }if (contador4 % 2 == 1){
      snprintf (msg, 50, "OFF");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/patio/pulsador0", msg);
    }
  }
   if (pulsador61 == HIGH){
    contador5++;
    delay(200);
    if (contador5 % 2 == 0){
      snprintf (msg, 50, "ON");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/patio/pulsador1", msg);
    }if (contador5 % 2 == 1){
      snprintf (msg, 50, "OFF");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/patio/pulsador1", msg);
    }
  }
   if (pulsador62 == HIGH){
    contador6++;
    delay(200);
    if (contador6 % 2 == 0){
      snprintf (msg, 50, "ON");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/patio/pulsador2", msg);
    }if (contador6 % 2 == 1){
      snprintf (msg, 50, "OFF");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/patio/pulsador2", msg);
    }
  }
   if (pulsador63 == HIGH){
    contador7++;
    delay(200);
    if (contador7 % 2 == 0){
      snprintf (msg, 50, "ON");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/patio/pulsador3", msg);
    }if (contador7 % 2 == 1){
      snprintf (msg, 50, "OFF");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/patio/pulsador3", msg);
    }
```

```
  }
  long now = millis();
  if (now -lastMsg > 2000){
    lastMsg = now;
    if (pulsadorPerUp50 == HIGH){
      snprintf (msg, 50, "ON");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/comedor/pulsadorUp0", msg);
    }
    if (pulsadorPerDown50 == HIGH){
      snprintf (msg, 50, "ON");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/comedor/pulsadorDown0", msg);
    }
    if (pulsadorPerUp51 == HIGH){
      snprintf (msg, 50, "ON");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/comedor/pulsadorUp1", msg);
    }
    if (pulsadorPerDown51 == HIGH){
      snprintf (msg, 50, "ON");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/comedor/pulsadorDown1", msg);
    }
    if (pulsadorPerUp52 == HIGH){
      snprintf (msg, 50, "ON");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/comedor/pulsadorUp2", msg);
    }
    if (pulsadorPerDown52 == HIGH){
      snprintf (msg, 50, "ON");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/comedor/pulsadorDown2", msg);
    }
    snprintf (msg, 50, "%f", temperatura50);
    Serial.print("Publish message: ");
    Serial.println(msg);
    client.publish("home/comedor/temperatura0", msg);
    snprintf (msg, 50, "%f", temperatura51);
    Serial.print("Publish message: ");
    Serial.println(msg);
    client.publish("home/comedor/temperatura1", msg);
  }
}
```

3.4.6 Programación ESP32 Patio

```
#include <WiFi.h>
#include <PubSubClient.h>

const char* ssid = "Nombre de la red wifi";
//nombre de la red wifi de la vivienda
const char* password = "contraseña de la red wifi";
//contraseña de la red wifi anteriormente mencionada
const char* mqtt_server = "192.168.1.39";
//dirección ip del broker mqtt

const int lamp60 = 32;
//GPIO al que está conectado la lámpara lamp6.0
const int lamp61 = 27;
//GPIO al que está conectado la lámpara lamp6.1
const int kUp1 = 26;
//GPIO al que está conectado el relé KMup1
const int kDown1 = 25;
//GPIO al que está conectado el relé KMDown1
const int kUp2 = 23;
//GPIO al que está conectado el relé KMUp2
const int kDown2 = 22;
//GPIO al que está conectado el relé KMDown2
const int kUp3 = 21;
//GPIO al que está conectado el relé KMUp3
const int kDown3 = 19;
//GPIO al que está conectado el relé KMDown3
const int k4 = 14;
//GPIO al que está conectado el relé K4

//El siguiente grupo de variables se utiliza para primero convertir el
meensaje que entra en una cadena y segundo
//convertir esta cadena en una variable float
String str;
char str0;
char str1;
char str2;
char str3;
char str4;
byte mat[10];
float temperatura;


WiFiClient espClient;
//Crea un cliente que se puede conectar a una determinada dirección IP
definida en
client.connect()

PubSubClient client(espClient);
//Crea un cliente para publicar y suscribirse a través de un
determinado cliente WiFi

void setup_wifi()
{

  delay(10);
  // We start by connecting to a WiFi network
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
```

```
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  randomSeed(micros());

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
  }

//Esta función es la que se encarga de recibir e interpretar los
mensajes MQTT de los topics suscritos
void callback(char* topic, byte* payload, unsigned int length) {
  if (strcmp(topic,"home/comedor/calefaccion")==0){
      str = "";
      for (int i = 0; i < length; i++){
        mat[i] = payload[i];
      }
      str0 = mat[0];
      str1 = mat[1];
      str2 = mat[2];
      str3 = mat[3];
      str4 = mat[4];
      str = String(str + str0 + str1 + str2 + str3 + str4);
      temperatura = str.toFloat();
      if (temperatura<15){
         digitalWrite(k4, HIGH);
      }if(temperatura>20){
        digitalWrite(k4, LOW);
      }
    }
  if (strcmp(topic, "home/patio/lampara0")==0){
    if ((char)payload[1] == 'N'){
    digitalWrite(lamp60, HIGH);
   }else{
    digitalWrite(lamp60, LOW);
   }
  }
  if (strcmp(topic, "home/patio/lampara1")==0){
    if ((char)payload[1] == 'N'){
    digitalWrite(lamp61, HIGH);
   }else{
    digitalWrite(lamp61, LOW);
   }
  }
  if (strcmp(topic, "home/comedor/persianaUp0")==0){
    if ((char)payload[1] == 'N'){
      digitalWrite(kUp1, HIGH);
      delay(15000);
      digitalWrite(kUp1, LOW);
    }
  }
  if (strcmp(topic, "home/comedor/persianaDown0")==0){
    if ((char)payload[1] == 'N'){
      digitalWrite(kDown1, HIGH);
```

```
      delay(15000);
      digitalWrite(kDown1, LOW);
    }
  }
  if (strcmp(topic, "home/comedor/persianaUp1")==0){
    if ((char)payload[1] == 'N'){
      digitalWrite(kUp2, HIGH);
      delay(15000);
      digitalWrite(kUp2, LOW);
    }
  }
  if (strcmp(topic, "home/comedor/persianaDown1")==0){
    if ((char)payload[1] == 'N'){
      digitalWrite(kDown2, HIGH);
      delay(15000);
      digitalWrite(kDown2, LOW);
    }
  }
  if (strcmp(topic, "home/comedor/persianaUp2")==0){
    if ((char)payload[1] == 'N'){
      digitalWrite(kUp3, HIGH);
      delay(15000);
      digitalWrite(kUp3, LOW);
    }
  }
  if (strcmp(topic, "home/comedor/persianaDown2")==0){
    if ((char)payload[1] == 'N'){
      digitalWrite(kDown3, HIGH);
      delay(15000);
      digitalWrite(kDown3, LOW);
    }
  }
 }
  //Esta función se utiliza para que el microcontrolador se mantenga
siempre conectado al servidor MQTT
void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Create a random client ID
    String clientId = "ESP8266Client-";
    clientId += String(random(0xffff), HEX);
    // Attempt to connect
    if (client.connect(clientId.c_str())) {
      Serial.println("connected");
      //Una vez conectado, envia un mensaje


      //Nos suscribimos a los topics de entrada
      client.subscribe("home/comedor/calefaccion");
      client.subscribe("home/comedor/lampara0");
      client.subscribe("home/comedor/lampara1");
      client.subscribe("home/patio/lampara0");
      client.subscribe("home/patio/lampara1");
      client.subscribe("home/comedor/persianaUp0");
      client.subscribe("home/comedor/persianaDown0");
      client.subscribe("home/comedor/persianaUp1");
      client.subscribe("home/comedor/persianaDown1");
      client.subscribe("home/comedor/persianaUp2");
      client.subscribe("home/comedor/persianaDown2");
```

```
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}
void setup() {
  //Definimos que GPIOs son salidas
  pinMode(lamp60, OUTPUT);
  pinMode(lamp61, OUTPUT);
  pinMode(k4, OUTPUT);
  pinMode(kUp1, OUTPUT);
  pinMode(kDown1, OUTPUT);
  pinMode(kUp2, OUTPUT);
  pinMode(kDown2, OUTPUT);
  pinMode(kUp3, OUTPUT);
  pinMode(kDown3, OUTPUT);

  //Definimos la velocidad de transmisión del puerto serie
  Serial.begin(115200);
  //Llamamos a la función que conecta con la red WiFi de la vivienda
  setup_wifi();
  //Definimos el puerto que tiene abierto el servidor para la
comunicación MQTT
  client.setServer(mqtt_server, 1883);
  //Definimos la función que recibe los mensajes de entrada
  client.setCallback(callback);
}

void loop(){
  if (!client.connected()){
    reconnect();
  }
  client.loop();
}
```

3.4.7 Programación ESP32 Cocina/Baño2

```
#include <WiFi.h>
#include <PubSubClient.h>

const char* ssid = "Nombre de la red wifi";
//nombre de la red wifi de la vivienda
const char* password = "contraseña de la red wifi";
//contraseña de la red wifi anteriormente mencionada
const char* mqtt_server = "192.168.1.39";
//dirección ip del broker mqtt

const int temp80 = 33;
//GPIO al que está conectado al sensor de temperatura
const int gas80 = 35;
//GPIO al que está conectado al sensor de gas
const int pul80 = 34;
//GPIO al que está conectado el pulsador pul8.0
const int pul81 = 32;
//GPIO al que está conectado el pulsador pul8.1
const int pul82 = 27;
//GPIO al que está conectado el pulsador pul8.2
const int pul70 = 26;
//GPIO al que está conectado el pulsador pul7.0

const int k5 = 23;
//GPIO al que está conectado al relé k5
const int p1 = 22;
//GPIO al que está conectado al zumbador p1
const int lamp80 = 21;
//GPIO al que está conectado la lámpara lamp8.0
const int lamp70 = 19;
//GPIO al que está conectado la lámpara lamp7.0
const int lamp81 = 15;
//GPIO al que está conectado la lámpara lamp8.1

int pulsador80 = 0;
//Esta variable contiene el valor de la entrada digital a la que está
conectado el pulsador pul8.0
int pulsador81 = 0;
//Esta variable contiene el valor de la entrada digital a la que está
conectado el pulsador pul8.1
int pulsador82 = 0;
//Esta variable contiene el valor de la entrada digital a la que está
conectado el pulsador pul8.2
int pulsador70 = 0;
//Esta variable contiene el valor de la entrada digital a la que está
conectado el pulsador pul7.0
int sensorGas80 = 0;
//Esta variable contiene el valor de la entrada digital a la que está
conectado al sensor de gas gas8.0
int contador1 = 0;
//Esta variable es usada para conocer el estado del pulsador pul8.0
int contador2 = 0;
//Esta variable es usada para conocer el estado del pulsador pul8.1
int contador3 = 0;
//Esta variable es usada para conocer el estado del pulsador pul8.2
int contador4 = 0;
//Esta variable es usada para conocer el estado del pulsador pul7.0

//Propiedades de las salidas PWM
```

```
const int freq = 5000;
const int dimChannel = 1;
const int resolution = 8;

//Variables usadas para la lectura y conversión de la señal del sensor
de temperatura temp5.0
float temp;
float tempVal;
float temperaturaMedida;

//Variables usadas para la lectura del mensaje que se recibe por MQTT
String str;
char str0;
char str1;
char str2;
char str3;
char str4;
byte mat[10];
float temperatura;

WiFiClient espClient;
//Crea un cliente que se puede conectar a una determinada dirección IP
definida en
client.connect()

PubSubClient client(espClient);
//Crea un cliente para publicar y suscribirse a través de un
determinado cliente WiFi

char msg[50];
//Se declara la variable que contiene el mensaje que se envía por
MQTT
long lastMsg = 0;

//Esta función establece conexión con la red WiFi de la vivienda
void setup_wifi()
{

  delay(10);
  // We start by connecting to a WiFi network
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  randomSeed(micros());

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

//Esta función es la que se encarga de recibir e interpretar los
mensajes MQTT de los topics suscritos
```

```
void callback(char* topic, byte* payload, unsigned int length) {
  if (strcmp(topic,"home/cocina/calefaccion")==0){
   str =
"";
   for (int i = 0;i < length; i++){
    mat[i] = payload[i];
   }
   str0 = mat[0];
   str1 = mat[1];
   str2 = mat[2];
   str3 = mat[3];
   str4 = mat[4];
   //concatenamos los caracteres que creemos convenientes
   str = String(str + str0 + str1 + str2 + str3 + str4);
   //y convertimos esa variable String en un float
   temperatura = str.toFloat();
   //Dependiendo de la temperatura recibida se activará o desactivará
la calefacción
   if (temperatura<15){
    digitalWrite(k5, HIGH);
   }if (temperatura>20){
    digitalWrite(k5, LOW);
   }
  }
  if (strcmp(topic, "home/cocina/alarmaGas")){
    if ((char)payload[1] == 'N'){
      digitalWrite(p1, HIGH);
      delay(1000);
      digitalWrite(p1, LOW);
    }
  }
  if (strcmp(topic,"home/bano2/lampara0")==0){
   if ((char)payload[1] == 'N'){
    digitalWrite(lamp70, HIGH);
   }else{
    digitalWrite(lamp70, LOW);
   }
   }
  if (strcmp(topic,"home/cocina/lampara0")==0){
   if ((char)payload[1] == 'N'){
    digitalWrite(lamp80, HIGH);
   }else{
    digitalWrite(lamp80, LOW);
   }
   }
   if (strcmp(topic,"home/comedor/lampara0")==0){
    String str;
    byte mat[2];
    for (int i = 0; i < length; i++){
      mat[i] = payload[i];
    }
    char str0 = mat[0];
    char str1 = mat[1];
    char str2 = mat[2];
    str = String(str + str0 + str1 + str2);
    int porcentaje = str.toInt();
    switch (porcentaje){
      case 0:
        ledcWrite(dimChannel, 0);
      break;
      case 25:
```

```
        ledcWrite(dimChannel,63.75);
      break;
      case 50:
        ledcWrite(dimChannel, 127.5);
      break;
      case 75:
        ledcWrite(dimChannel, 192.25);
      break;
      case 10:
        ledcWrite(dimChannel, 255);
      break;
    }
  }
}

//Esta función se utiliza para que el microcontrolador se mantenga
siempre conectado al servidor MQTT
void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Create a random client ID
    String clientId = "ESP8266Client-";
    clientId += String(random(0xffff), HEX);
    // Attempt to connect
    if (client.connect(clientId.c_str())) {
      Serial.println("connected");
      //Una vez conectado, envia un mensaje


      //Nos suscribimos a los topics de entrada
      client.subscribe("home/cocina/lampara0");
      client.subscribe("home/cocina/lampara1");
      client.subscribe("home/cocina/alarmaGas");
      client.subscribe("home/cocina/calefaccion");


    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}

void setup(){
  //Definimos que GPIO son salidas PWM
  ledcSetup(dimChannel, freq, resolution);
  ledcAttachPin(lamp81, dimChannel);

  //Definimos que GPIOs son entradas
  pinMode(pul80,INPUT);
  pinMode(pul81,INPUT);
  pinMode(pul82,INPUT);
  pinMode(pul70,INPUT);
  pinMode(gas80,INPUT);

  //Definimos que GPIO son salidas
  pinMode(k5,OUTPUT);
```

```
  pinMode(p1,OUTPUT);
  pinMode(lamp80,OUTPUT);
  pinMode(lamp81,OUTPUT);
  pinMode(lamp70,OUTPUT);

  Serial.begin(115200);
  setup_wifi();
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);
}

void loop() {

  if (!client.connected()) {
    reconnect();
  }
  client.loop();

  //Leemos el estado del GPIO
  pulsador80 = digitalRead(pul80);
  pulsador81 = digitalRead(pul81);
  pulsador70 = digitalRead(pul70);
  sensorGas80 = digitalRead(gas80);

  tempVal = analogRead(temp);
  temp = map(tempVal, 0, 4095, 0, 5000);
  temperatura = temp/100;

  if (pulsador80 == HIGH){
    contador1++;
    if (contador1 == 5){
      contador1 = 0;
    }
    delay(200);
    switch (contador1){
      case 0:
        snprintf (msg, 50, "0");
        Serial.print("Publish message: ");
        Serial.println(msg);
        client.publish("home/cocina/pulsador0", msg);
      break;
      case 1:
        snprintf (msg, 50, "25");
        Serial.print("Publish message: ");
        Serial.println(msg);
        client.publish("home/cocina/pulsador0", msg);
      break;
      case 2:
        snprintf (msg, 50, "50");
        Serial.print("Publish message: ");
        Serial.println(msg);
        client.publish("home/cocina/pulsador0", msg);
      break;
      case 3:
        snprintf (msg, 50, "75");
        Serial.print("Publish message: ");
        Serial.println(msg);
        client.publish("home/cocina/pulsador0", msg);
      break;
      case 4:
        snprintf (msg, 50, "100");
```

```
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/cocina/pulsador0", msg);
    break;
  }
}
if (pulsador81 == HIGH){
  contador2++;
  if (contador2 == 5){
    contador2 = 0;
  }
  delay(200);
  switch (contador2){
    case 0:
      snprintf (msg, 50, "0");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/cocina/pulsador1", msg);
    break;
    case 1:
      snprintf (msg, 50, "25");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/cocina/pulsador1", msg);
    break;
    case 2:
      snprintf (msg, 50, "50");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/cocina/pulsador1", msg);
    break;
    case 3:
      snprintf (msg, 50, "75");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/cocina/pulsador1", msg);
    break;
    case 4:
      snprintf (msg, 50, "100");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/cocina/pulsador1", msg);
    break;
  }
}
if (pulsador82 == HIGH){
  contador3++;
  delay(200);
  if (contador3 % 2 == 0){
    snprintf (msg, 50, "ON");
    Serial.print("Publish message: ");
    Serial.println(msg);
    client.publish("home/cocina/pulsador2", msg);
  }if (contador3 % 2 == 1){
    snprintf (msg, 50, "OFF");
    Serial.print("Publish message: ");
    Serial.println(msg);
    client.publish("home/cocina/pulsador2", msg);
  }
}
if (pulsador70 == HIGH){
```

```
    contador4++;
    delay(200);
    if (contador4 % 2 == 0){
      snprintf (msg, 50, "ON");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/bano2/pulsador0", msg);
    }if (contador4 % 2 == 1){
      snprintf (msg, 50, "OFF");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/bano2/pulsador0", msg);
    }
  }
  long now = millis();
  if (now -lastMsg > 2000){
    lastMsg = now;
    if (sensorGas80 == HIGH){
      snprintf (msg, 50, "ON");
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("home/cocina/gas", msg);
    }
    snprintf (msg, 50, "%f", temperatura);
    Serial.print("Publish message: ");
    Serial.println(msg);
    client.publish("home/cocina/temperatura0", msg);
  }
}
```

4. Hardware

Por la naturaleza del sistema de control se puede decir que se trata de un sistema de control distribuido, ya que, tanto la Raspberry Pi y los microcontroladores disponen de una unidad de control.

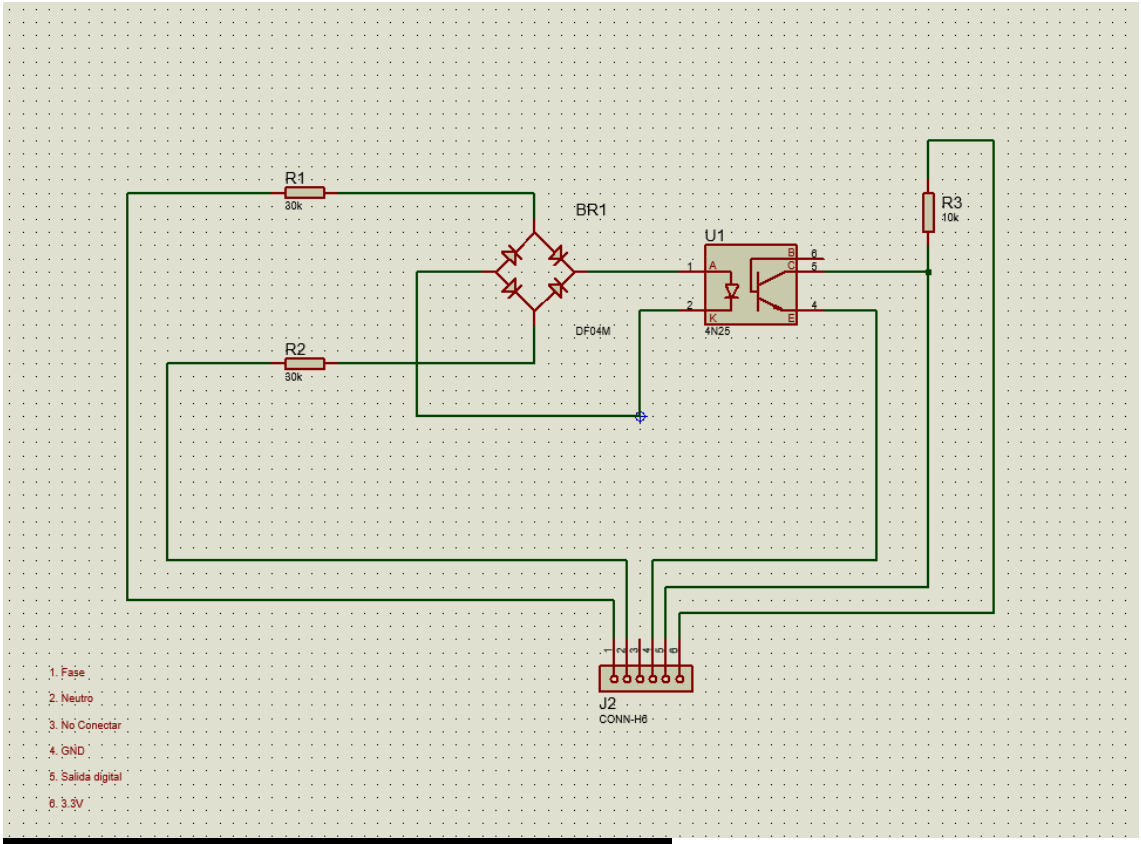
El sistema de control se comunica mediante Wifi. Los sensores están cableados a los microcontroladores y los actuadores están cableados también al microcontrolador, pero en este caso disponen de circuitos intermedios capaces de suministrar la energía eléctrica en el modo que el receptor lo necesite debido a que las salidas de los microcontroladores son de muy baja potencia.

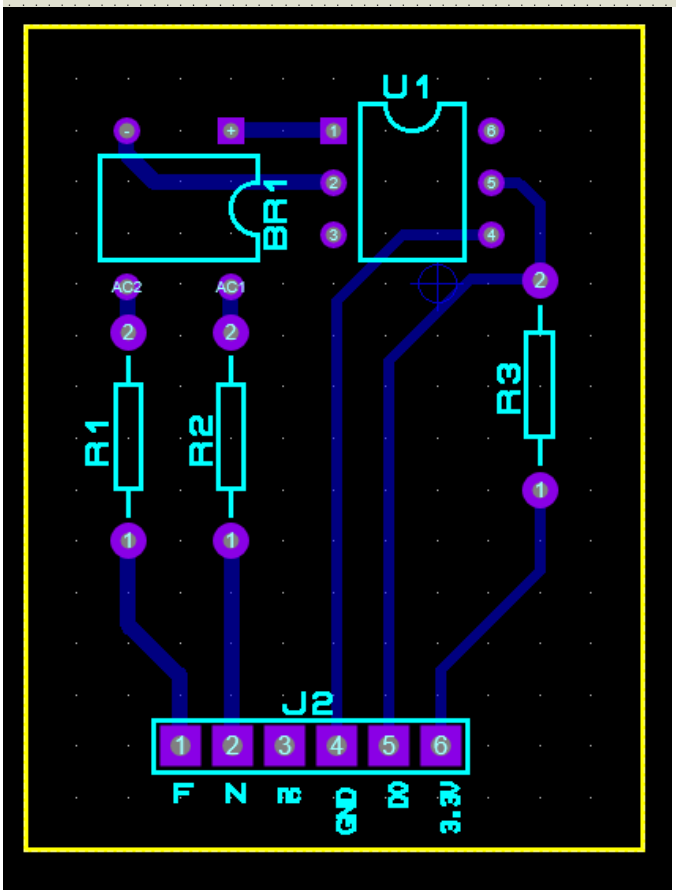
4.1 Circuitos de acondicionamiento

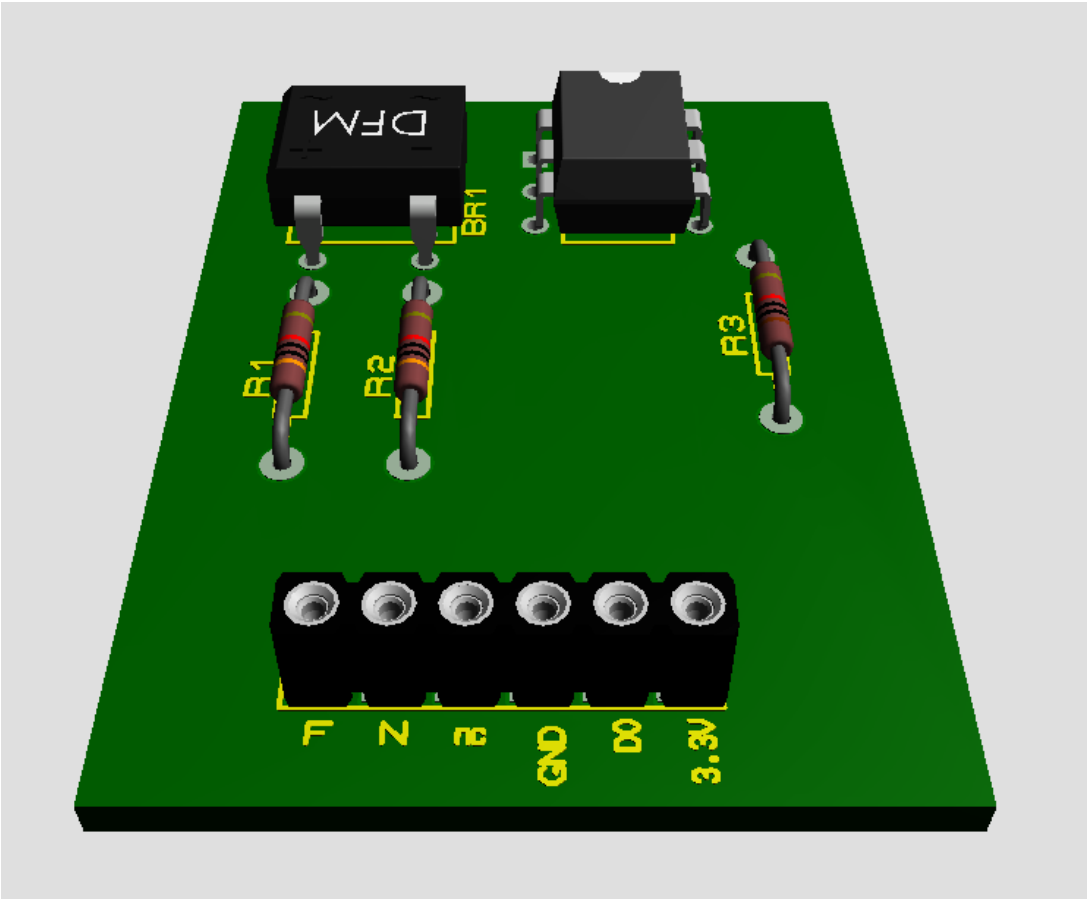
4.1.1 Sensor PIR

Los sensores infrarrojos pasivos (PIR) son dispositivos para la detección de movimiento.
Los sensores PIR se basan en la medición de la radiación infrarroja. Todos los cuerpos (vivos o no) emiten una cierta cantidad de energía infrarroja, mayor cuanto mayor es su temperatura. Los dispositivos PIR disponen de un sensor piro eléctrico capaz de captar esta radiación y convertirla en una señal eléctrica.
Ahora bien, la mayoría de sensores PIR se alimentan con fase y neutro y cuando detectan algo se activa un contacto que suministra 230V de corriente alterna, este sensor está pensado para conectar una carga entre la salida del mismo y el neutro.

Debido a que queremos tener una señal que pueda interpretar un ESP32 sin dañarlo, debemos tener un circuito intermedio.

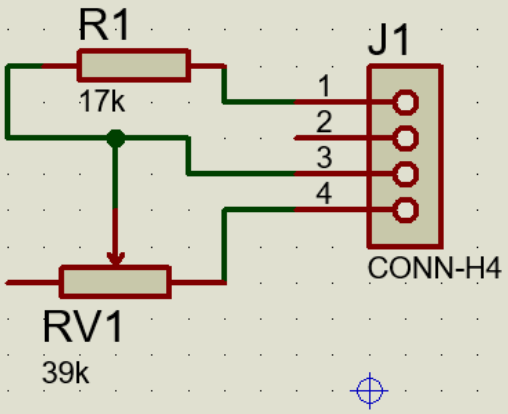






4.1.2 Sensor de gas MQ2

Este sensor se encarga de medir concentraciones de gas natural en el aire. Puede detectar concentraciones desde 300 hasta 10000 ppm.
El módulo posee una salida analógica que proviene del divisor de voltaje que forma el sensor y una resistencia de carga. También tiene una salida digital que se calibra con un potenciómetro, esta salida tiene un Led indicador.
La resistencia del sensor cambia de acuerdo a la concentración del gas en el aire.
El MQ2 es sensible a LPG, i-butano, propano, metano, alcohol, hidrogeno y humo.
El sensor tiene un calentador que solo funciona a 5V por lo que serán proporcionados por una fuente externa.
Usaremos un divisor de tensión en la salida para adaptar la señal de 0 a 5V, a 0 a 3,3V, debido a que la entrada analógica de nuestro microcontrolador, el ESP32 es de 0 a 3,3V.
Para calibrar el sensor usaremos el código proporcionado por sandbox electronics, con la única diferencia que tenemos que dividir por 0,66 debido a que el código está preparado para una entrada analógica de 0 a 5V y como he mencionado anteriormente nuestra tensión es de 0 a 3,3V.
El potenciómetro de RV1 se debe configurar para una resistencia de 33 KOhmios
Fuente → sandboxelectronic.com/?p=165
El divisor de tensión utilizado es el siguiente:

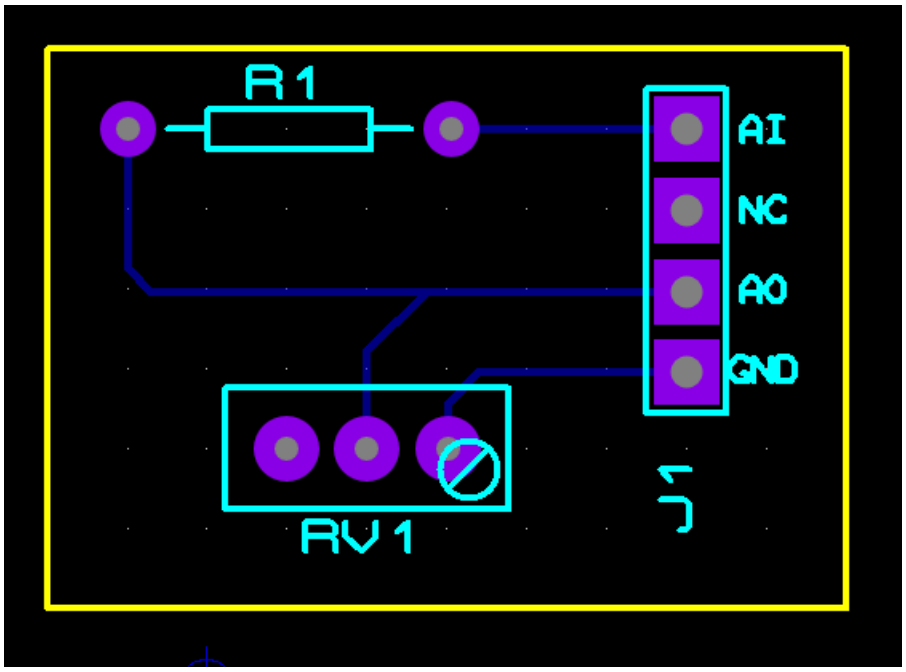


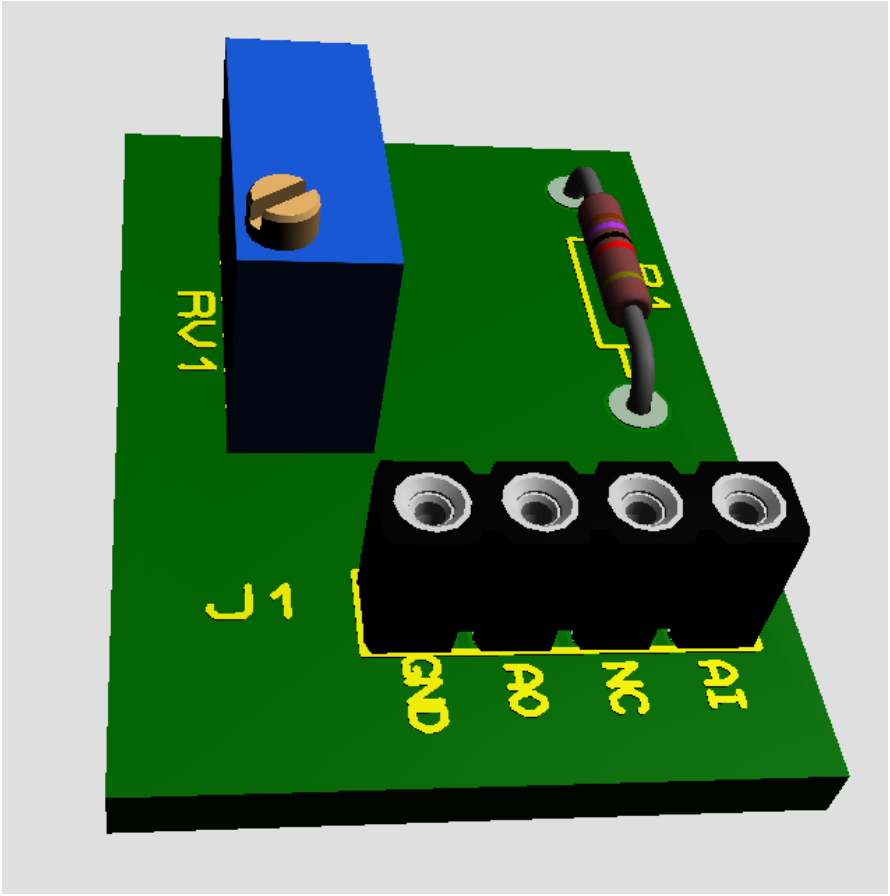
1. Entrada analógica

2. No conectado

3. Salida analógica

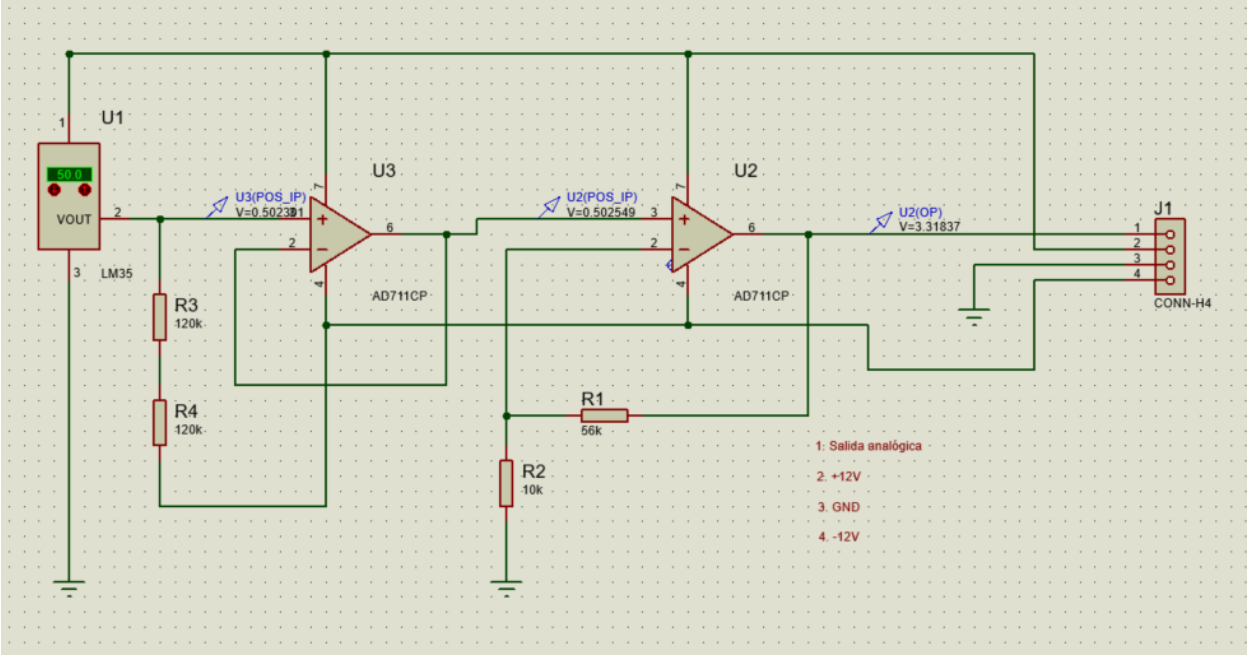
4. GND

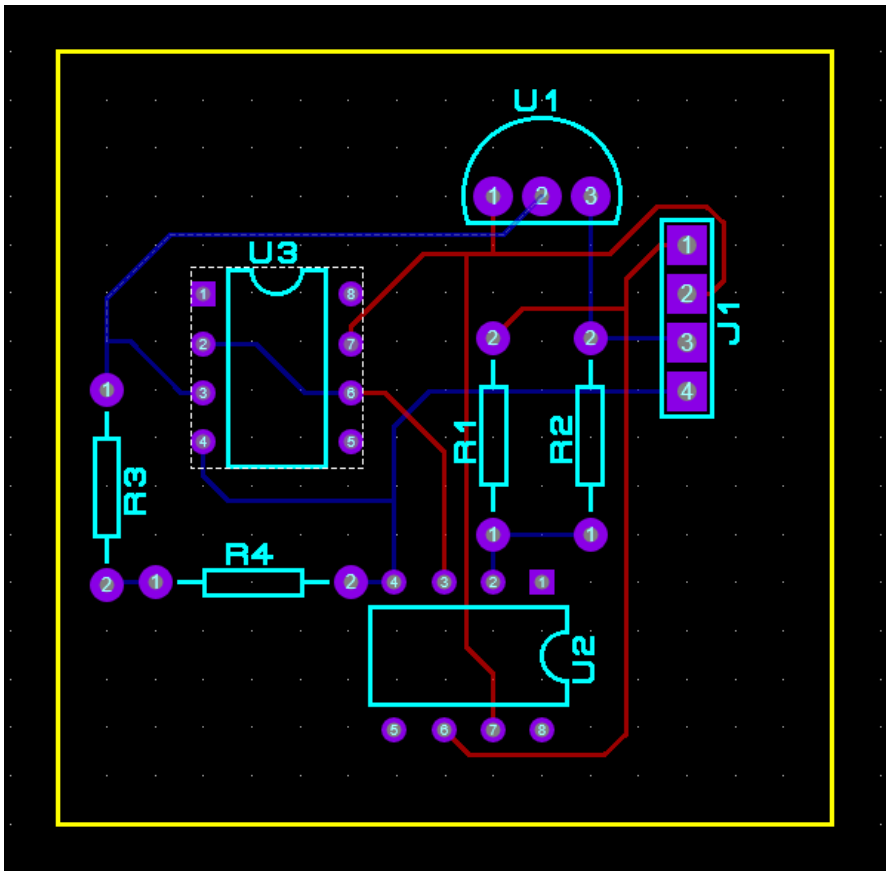

R1
AI
NC
AO
GND
RV1
J1

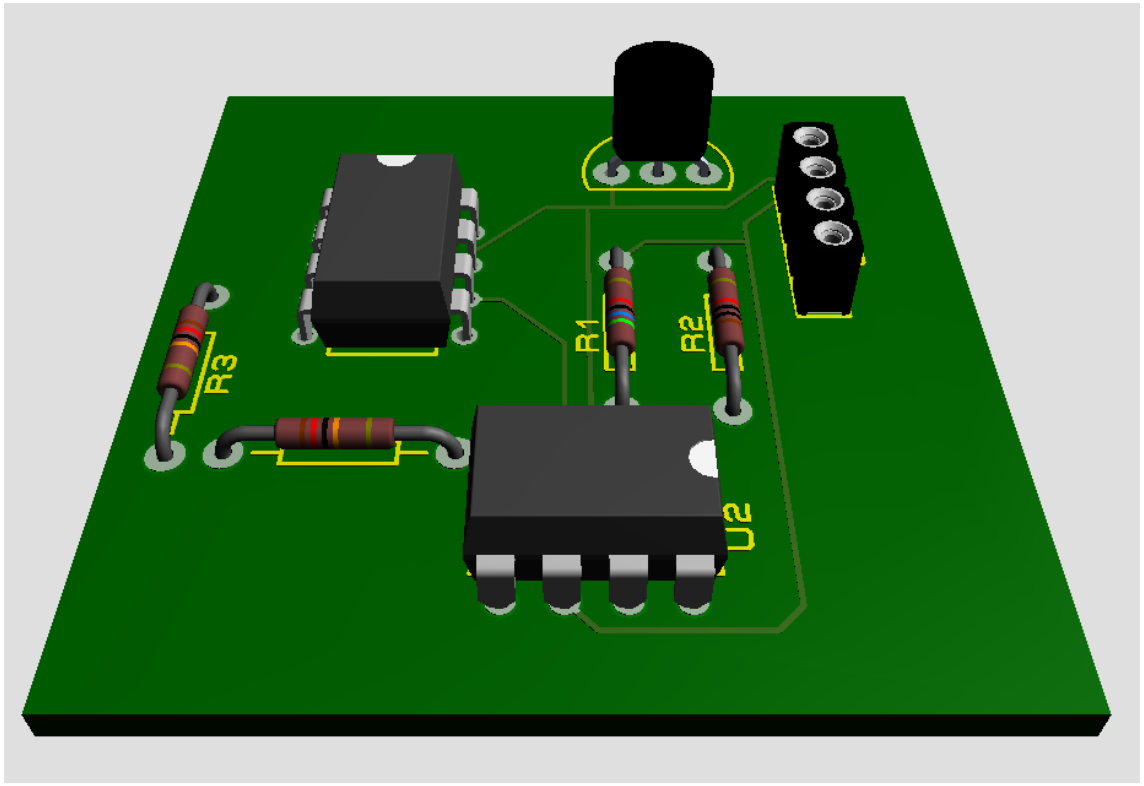

RV1
R1
J1
GND
AO
NC
AI

4.1.3 Sensor de temperatura LM35

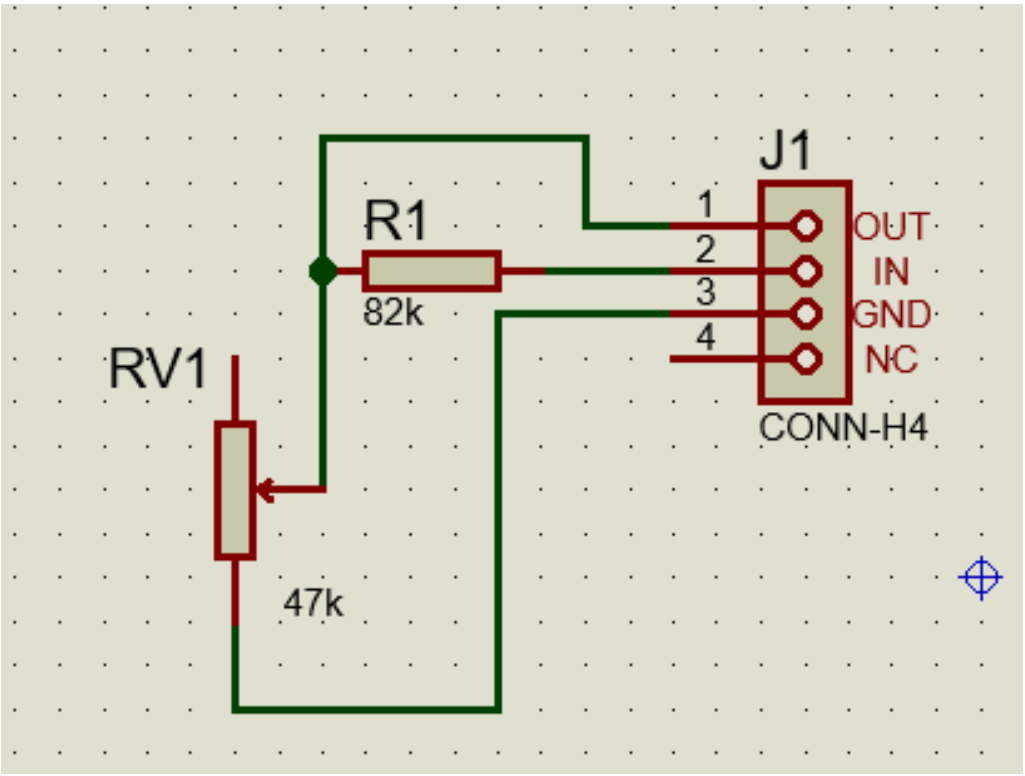
El LM35 es un sensor de temperatura digital. A diferencia de otros dispositivos como los termistores en los que la medición de temperatura se obtiene de la medición de su resistencia eléctrica, el LM35 es un integrado con su propio circuito de control, que proporciona una salida de voltaje proporcional a la temperatura.
La salida del LM35 es lineal con la temperatura, incrementando el valor a razón de 10mV por cada grado centígrado. El rango de medición es de -55ºC (-550mV) a 150ºC (1500mV).
Su precisión a temperatura ambiente es de 0,5ºC.
El siguiente circuito lo usamos para aumentar la tensión de medición para que nos de unos 9,98V a 150ºC y 3,31V a 50ºC, debido a la separación del sensor y el microcontrolador una tensión mayor implica que la caída de tensión afecte menos al valor que estamos intentando leer, por esta razón, existe un segundo circuito que se sitúa cerca del microcontrolador para adaptar la tensión a valores que no dañe y que pueda interpretar el microcontrolador.

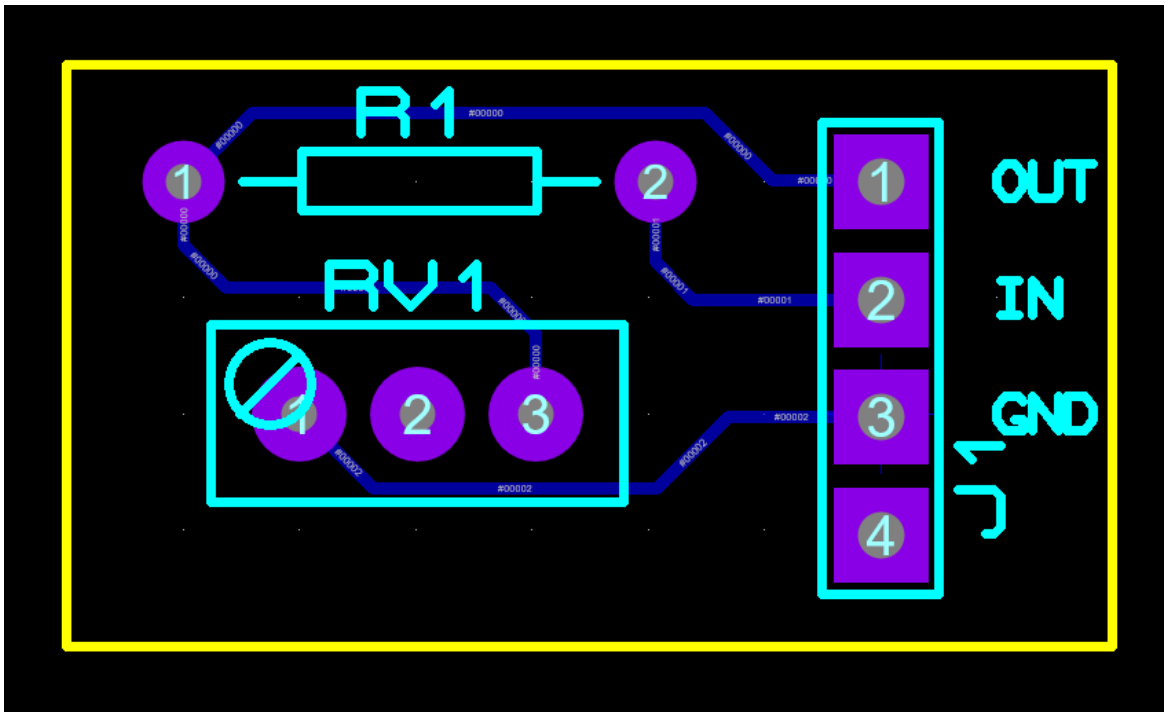


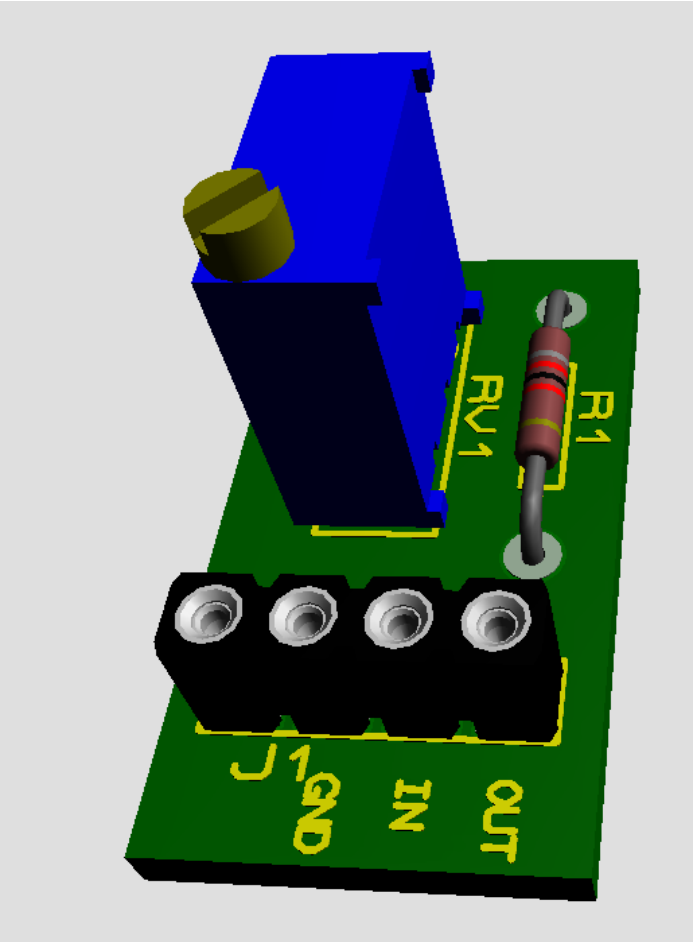

U1
U3
J1
R1
R2
R3
R4
U2


R1
R2
R3
U2

El circuito más próximo al microcontrolador se trata de un divisor de tensión que ajustaremos para que transforme una tensión de 9,9V a 3,3V, ajustándose debidamente también a la caída de tensión que realizamos midiendo la señal que tenemos y con la ayuda de otro sensor de temperatura.
El potenciómetro lo ajustaremos a un valor próximo a los 39 KOhmios.

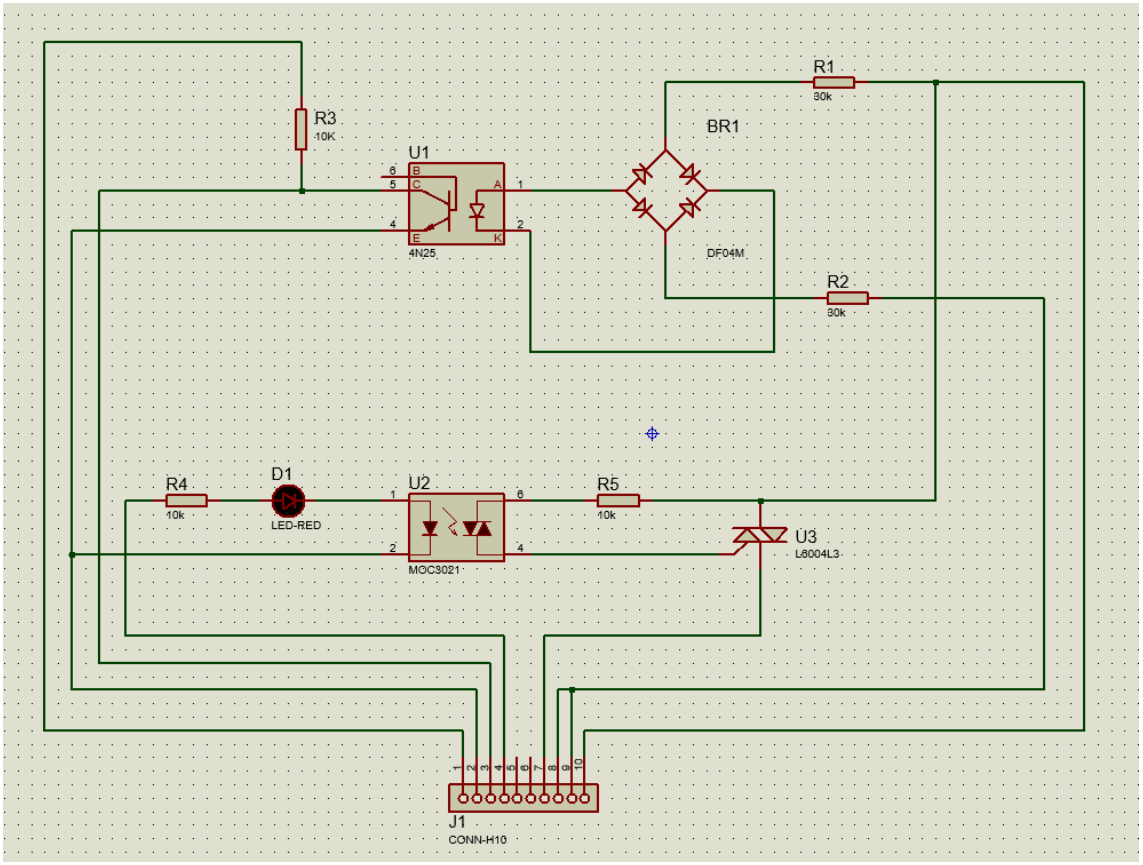


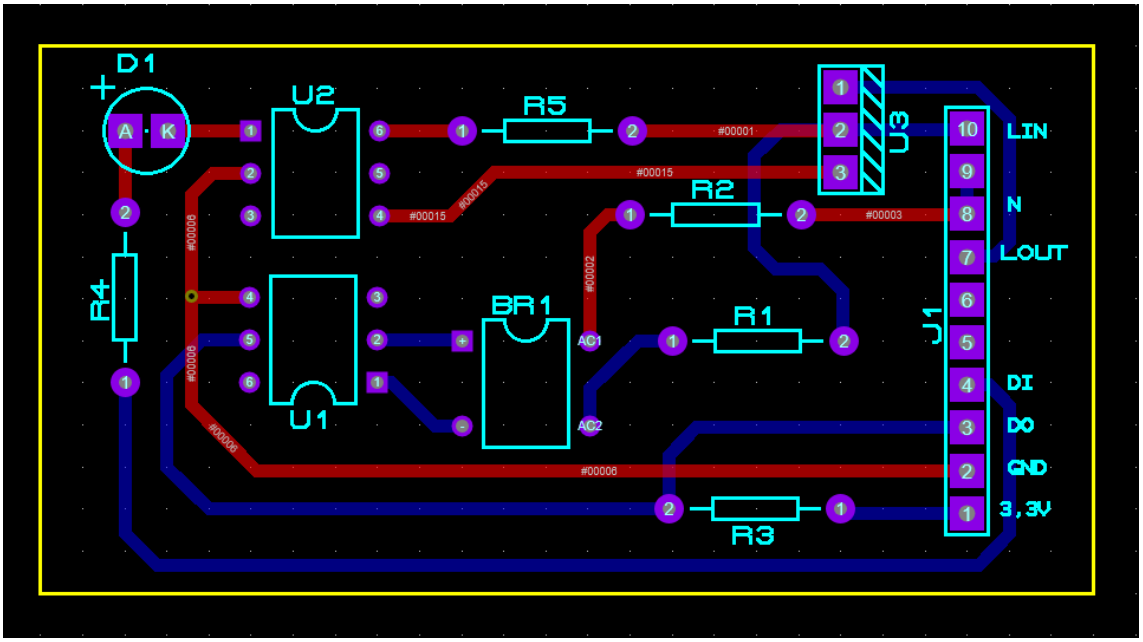


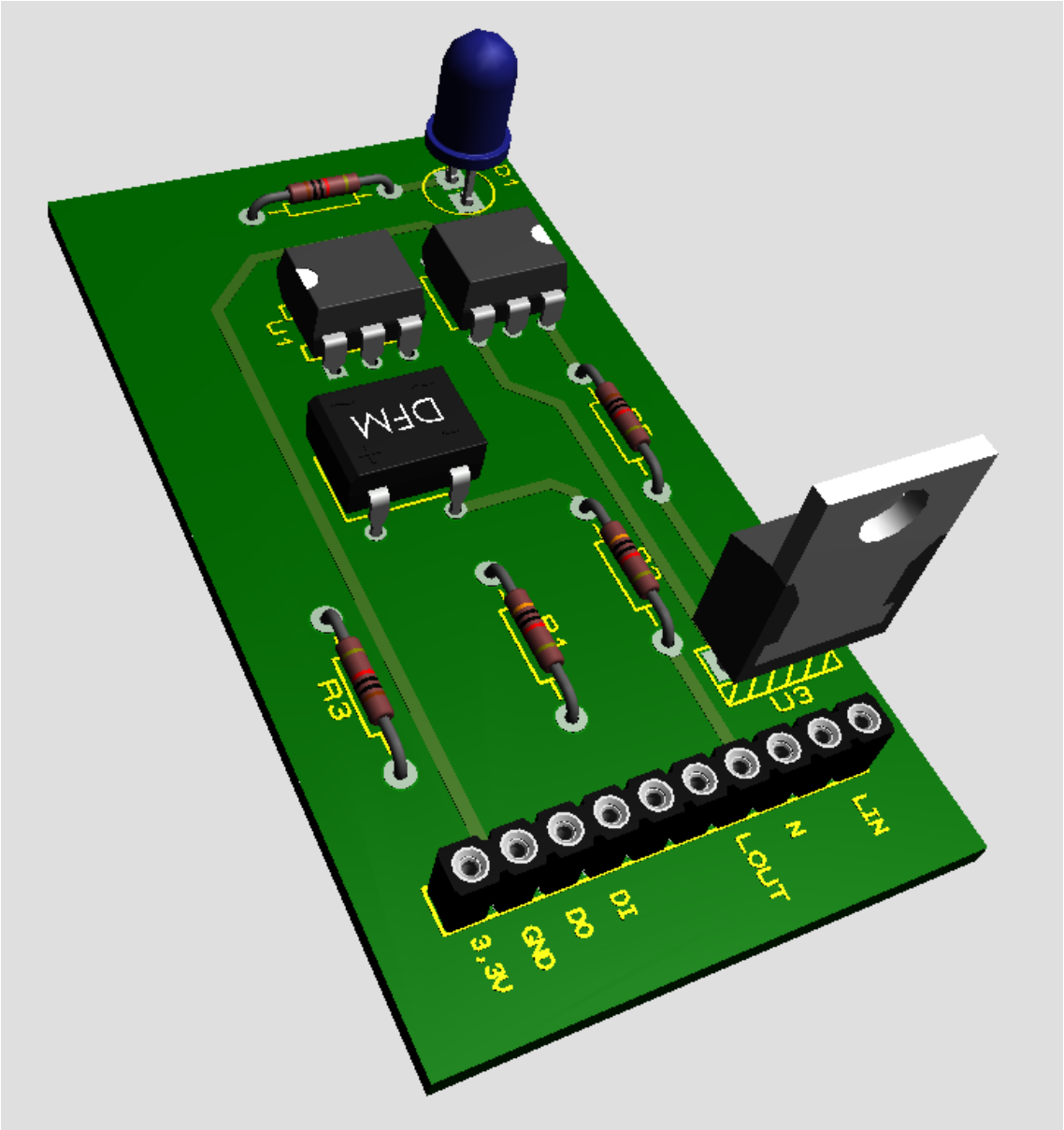

RV1
R1
J1
GND
IN
OUT

4.1.4 Dimmer

El funcionamiento de este actuador se basa en recortar la onda senoidal, de este modo se reduce el voltaje medio y desciende la potencia de la luminaria.

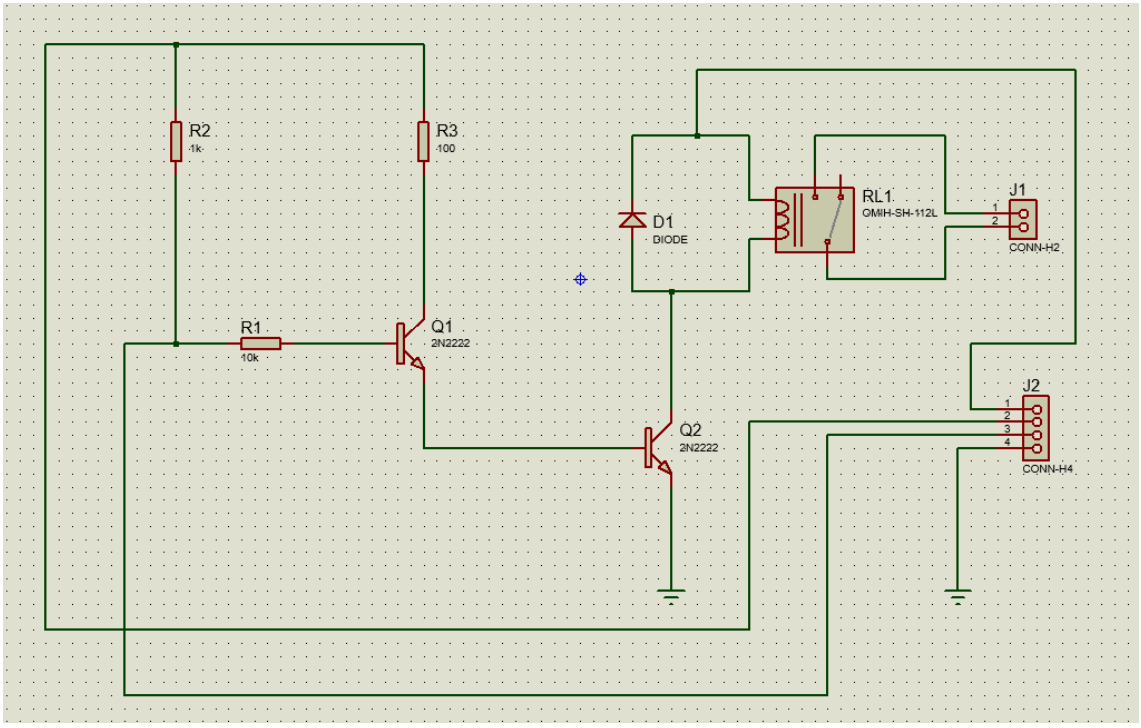


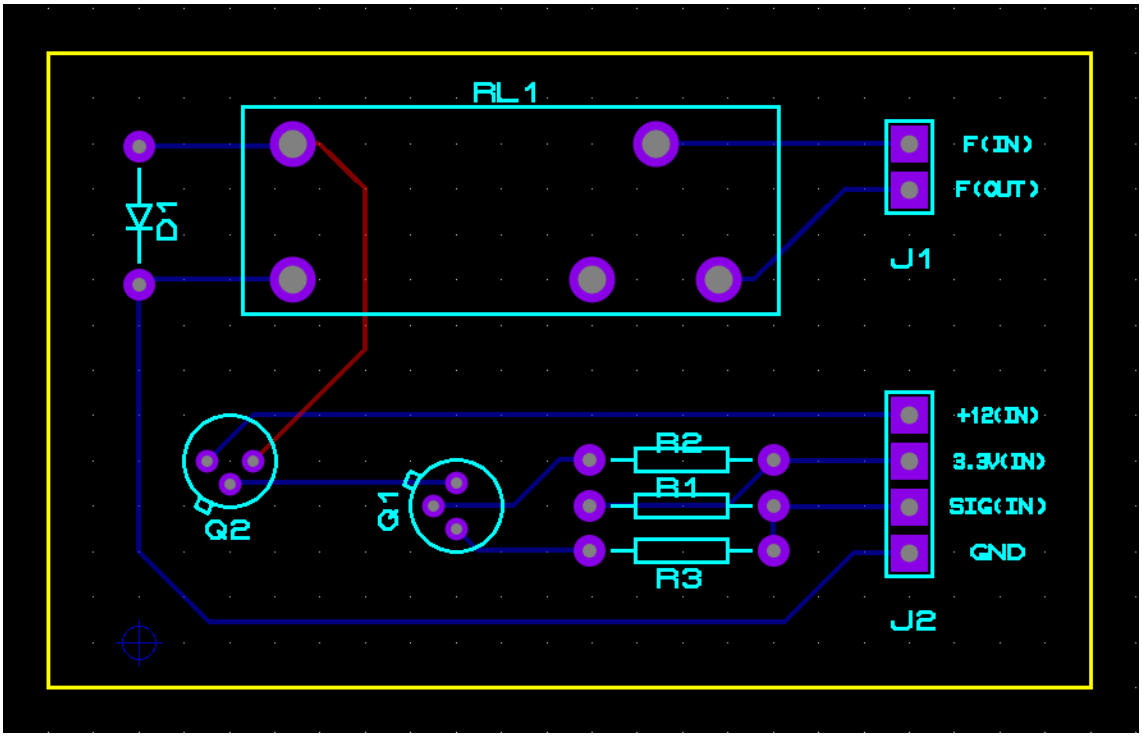


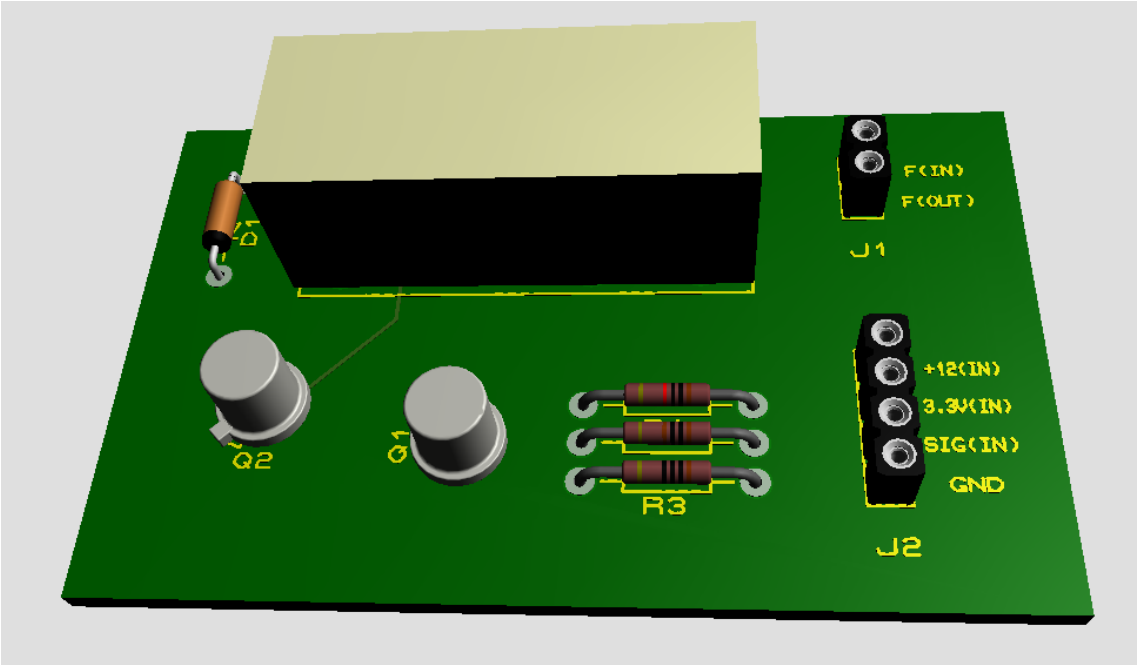

D1
U1
DFM
R4
R3
U3
3,3V
GND
DO
DI
LOUT
N
LIN

4.1.5 Actuador todo-nada

Este es un actuador todo-nada en el que activaremos las cargas que no necesiten ser reguladas.






D1
Q2
Q1
R3
F(IN)
F(OUT)
J1
+12(IN)
3.3V(IN)
SIG(IN)
GND
J2

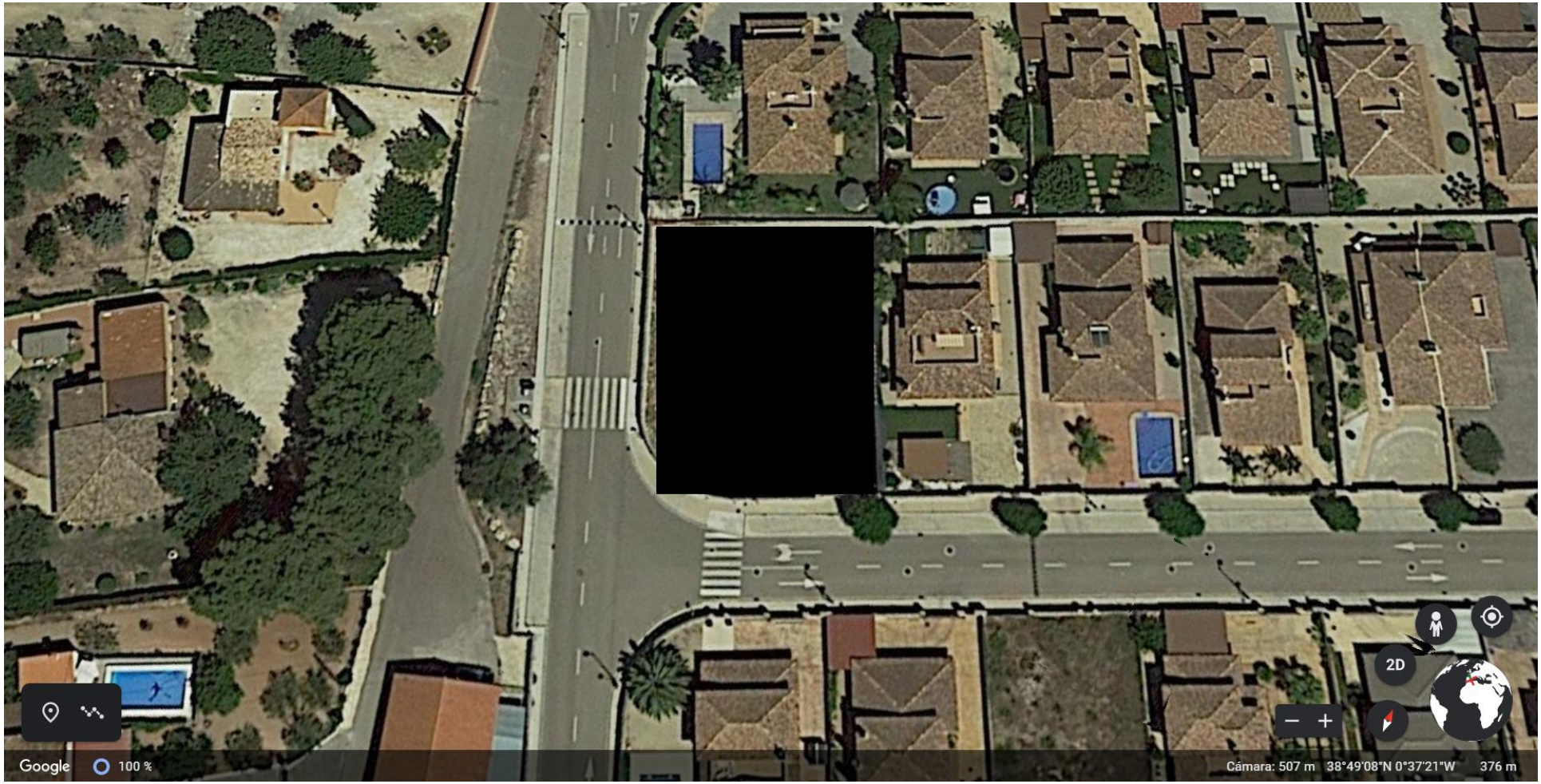
5.Presupuesto

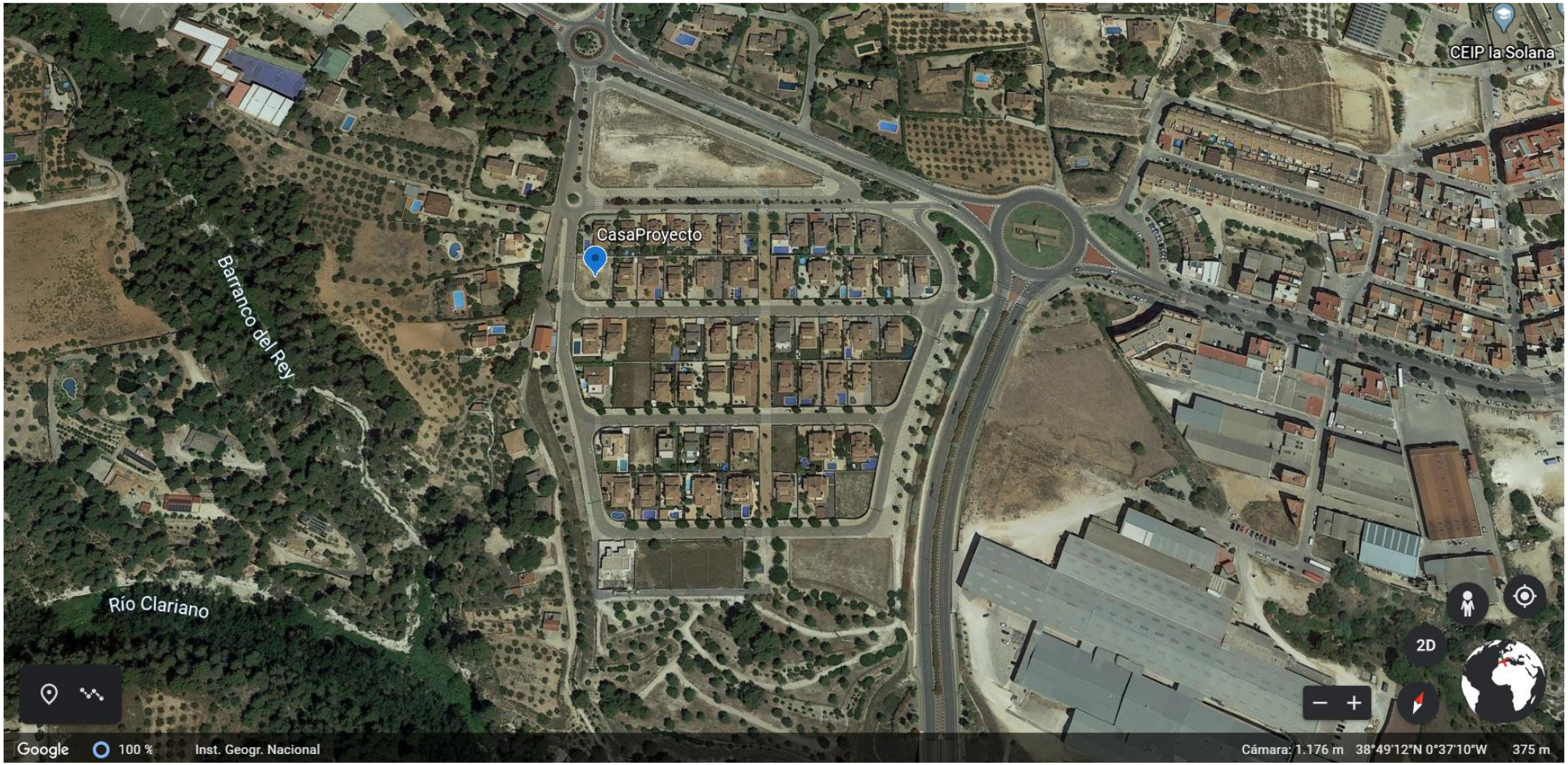
Descripción	€/unidad	nº unidad	Precio
Cable libre de halógenos 6mm POR METROS H07Z1-K AS 750V. Azul.	1,24 €	25	31,00 €
Cable libre de halógenos 6mm POR METROS H07Z1-K AS 750V. Marrón.	1,24 €	25	31,00 €
Cable libre de halógenos 6mm POR METROS H07Z1-K AS 750V. Verde/amarillo.	1,24 €	25	31,00 €
Cable libre de halógenos 4mm POR METROS H07Z1-K AS 750V. Azul.	0,84 €	40	33,60 €
Cable libre de halógenos 4mm POR METROS H07Z1-K AS 750V. Marrón.	0,84 €	40	33,60 €
Cable libre de halógenos 4mm POR METROS H07Z1-K AS 750V. Verde/amarillo.	0,84 €	40	33,60 €
Rollo 100m 2,5mm2. Cable flexible libre de halogeno. Azul.	27,65 €	2	55,30 €
Rollo 100m 2,5mm2. Cable flexible libre de halogeno. Marrón.	27,65 €	2	55,30 €
Rollo 100m 2,5mm2. Cable flexible libre de halogeno. Verde/amarillo.	27,65 €	2	55,30 €
Rollo 100m 1,5mm2. Cable flexible libre de halogeno. Azul.	16,90 €	2	33,80 €
Rollo 100m 1,5mm2. Cable flexible libre de halogeno. Marrón.	16,90 €	2	33,80 €
Rollo 100m 1,5mm2. Cable flexible libre de halogeno. Verde/amarillo.	16,90 €	2	33,80 €
Rollo 100m 2,5mm2. Cable flexible libre de halogeno. Rojo.	27,65 €	2	55,30 €
Rollo 100m 2,5mm2. Cable flexible libre de halogeno. Negro.	27,65 €	2	55,30 €
Base de enchufe para horno y vitro ME2134481	4,85 €	1	4,85 €
Base enchufe Schuko Simon 27472-65	3,23 €	42	135,66 €
Marco 1 elemento Simon 27 27601	1,82 €	6	10,92 €
Marco 2 elemento Simon 27 27620	3,60 €	10	36,00 €
Marco 3 elemento Simon 27 27630	5,74 €	3	17,22 €
Marco 4 elemento Simon 27 27640	7,81 €	5	39,05 €
Pulsador ancho Simon 27 27659-65	3,49 €	18	62,82 €
Pulsador persiana Simon 27 PLAY 27332	20,26 €	3	60,78 €
Doble conmutador estanco superficie LumiteK	5,50 €	2	11,00 €
Base schucko estanco superficie LumiteK	5,99 €	2	11,98 €
Pulsador estrecho Simon 27 27659-64	3,54 €	1	3,54 €
Caja Empotrar 1 elemento enlazable 65x65x40	3,41 €	63	214,83 €
BeMatik - Caja de distribución eléctrica metálica con protección IP65 para fijación en pared 300x300x200mm	56,06 €	7	392,42 €
Carril DIN aluminio 100cm Maxge	7,60 €	2	15,20 €
Diseño de componentes electrónicos	20,00 €	15	300,00 €
Programación	20,00 €	20	400,00 €
Raspberry Pi	39,90 €	1	39,90 €
ESP32	9,99 €	8	79,92 €
Motor de persiana radio RTS MRR 3000 de 30NM	229,00 €	2	458,00 €
Motor de persiana radio RTS MRR 1000 de 10NM	209,00 €	1	209,00 €
Fuente de alimentación MeanWell, RS-15-12	12,40 €	1	12,40 €
Fuente de alimentación MeanWell, RS-15-3,3	12,95 €	1	12,95 €
Fuente de alimentación MeanWell, RT-65B	18,68 €	5	93,40 €

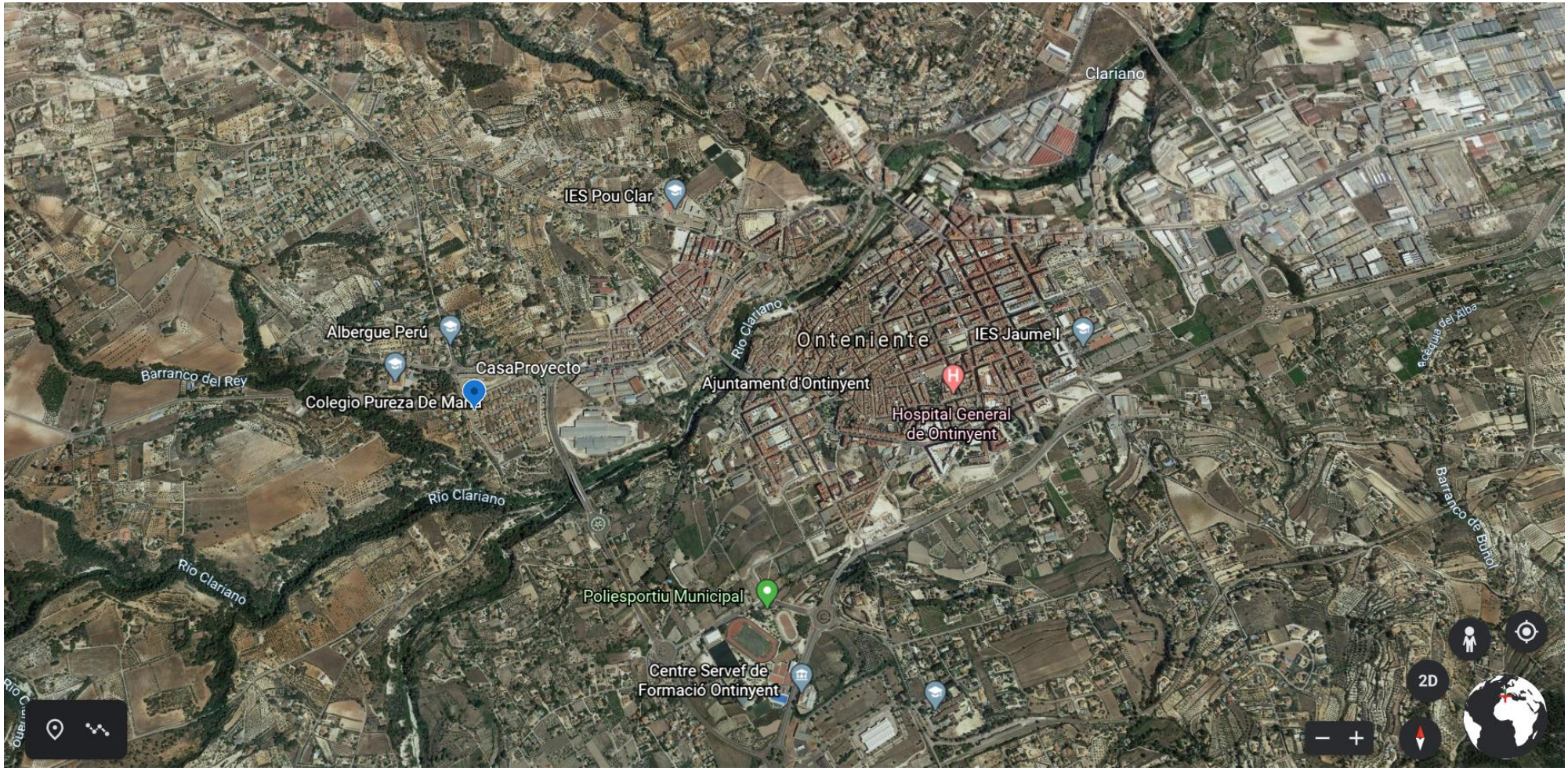
total --> 3273,44€

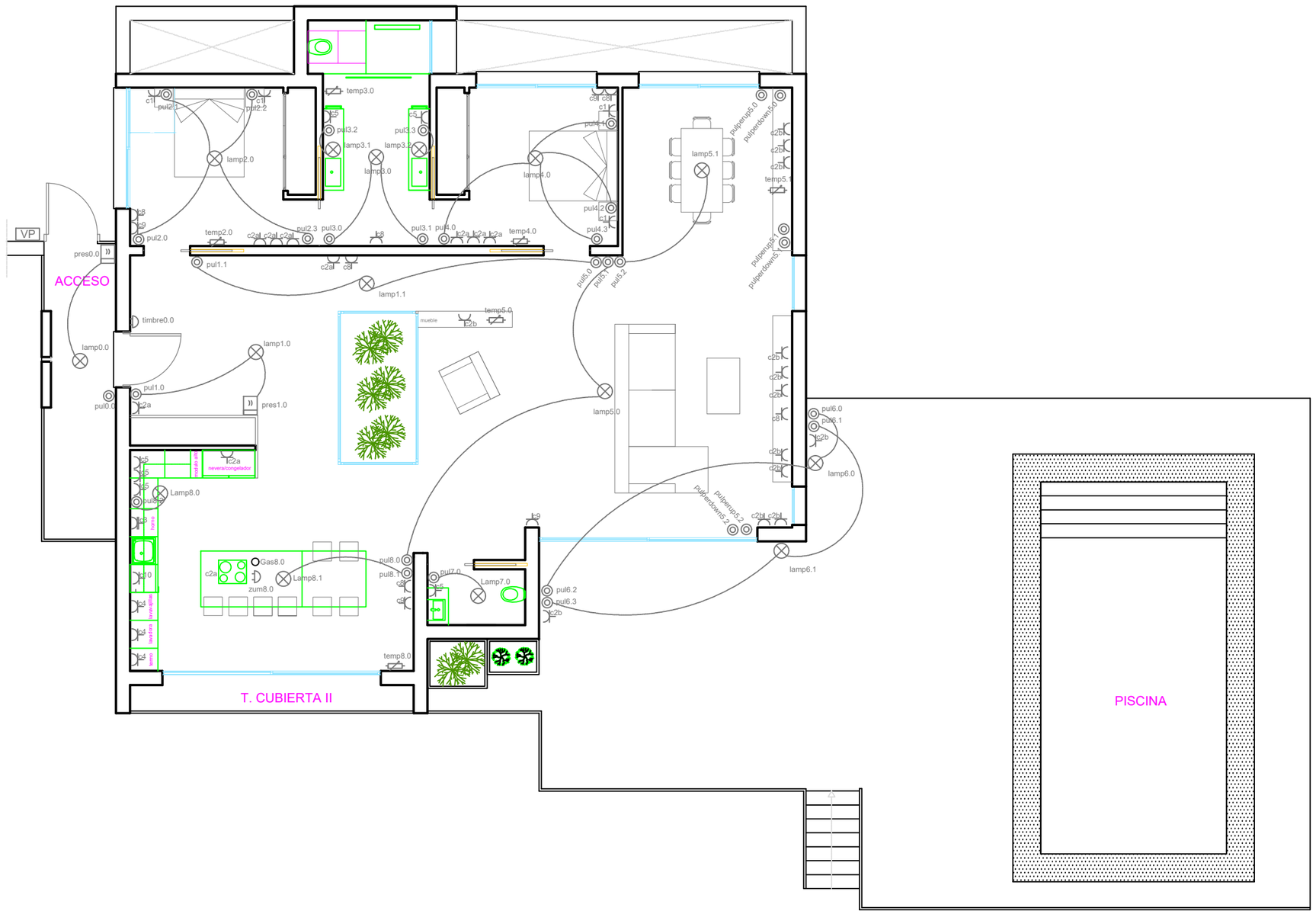
6.Planos y esquemas

La vivienda está situada en Carrer dels cavallets, 30 Ontinyent, Valéncia.


Google
100 %
2D
Cámara: 507 m 38°49'08"N 0°37'21"W 376 m


CEIP la Solana
CasaProyecto
Barranco del Rey
Río Clariano
2D
Google
100 %
Inst. Geogr. Nacional
Cámara: 1.176 m 38°49'12"N 0°37'10"W 375 m


Clariano
IES Pou Clar
Albergue Perú
CasaProyecto
Barranco del Rey
Colegio Pureza De María
Rio Clariano
Ontenient
IES Jaume I
Ajuntament d'Ontinyent
Hospital General
de Ontinyent
Acequia del Alba
Barranco de Buñol
Rio Clariano
Rio Clariano
Poliesportiu Municipal
Centre Servef de
Formació Ontinyent
2D



	Fecha	Nombre	Firmas	
Dibujado	5/07/2020	SBP		
Comprobado	15/07/2020	JDA		
Ids normas				

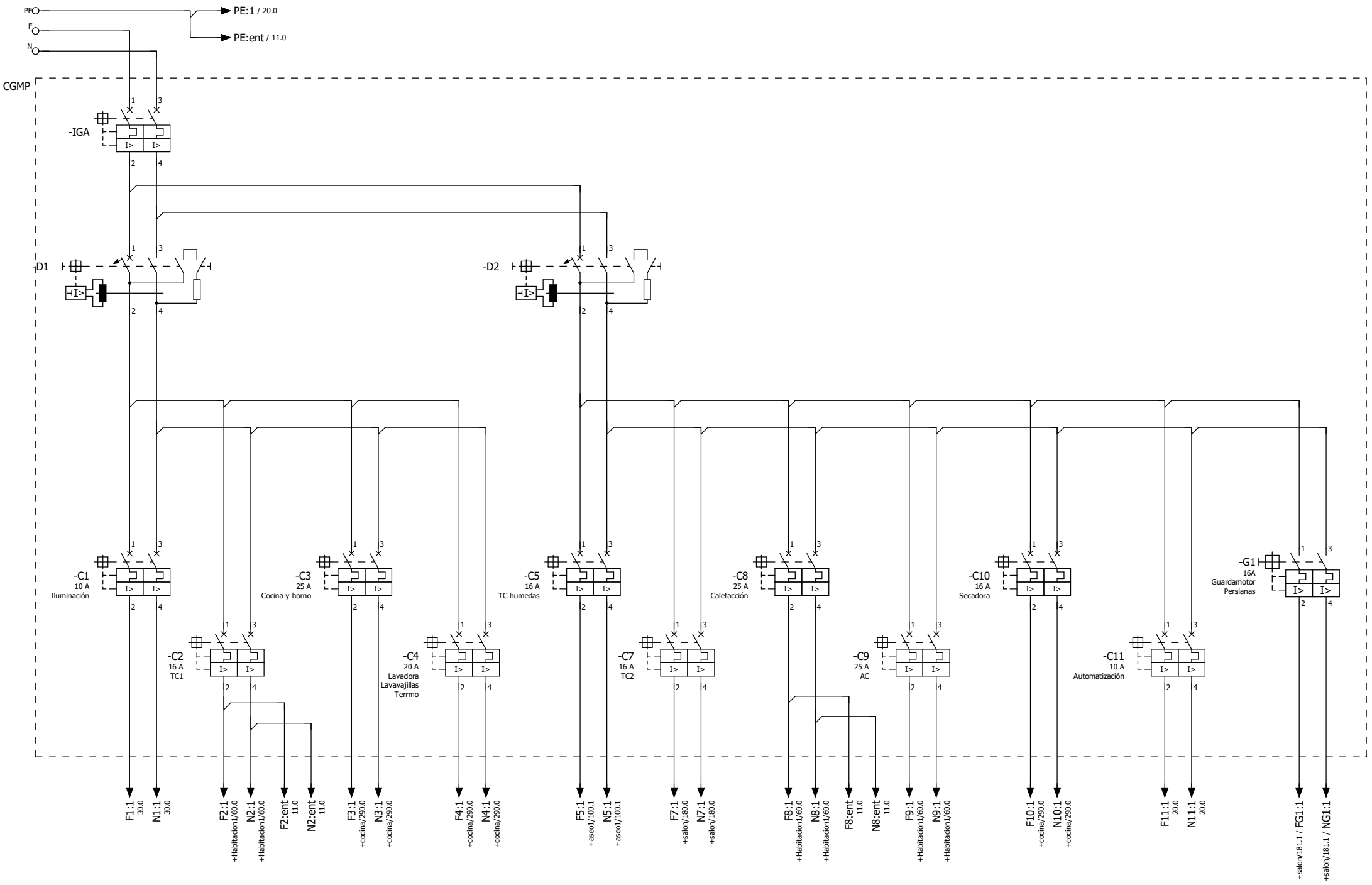
Escala 1:100		Sustituye a Sustituido por

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

CAMPUS D'ALCOI

Descripción de proyecto	Instalación domótica con Raspberry Pi y ESP32 mediante MQTT
Nombre de proyecto	InstalacionDomotica
Creado	08/06/2020
Modificado	28/07/2020
Número de páginas	36

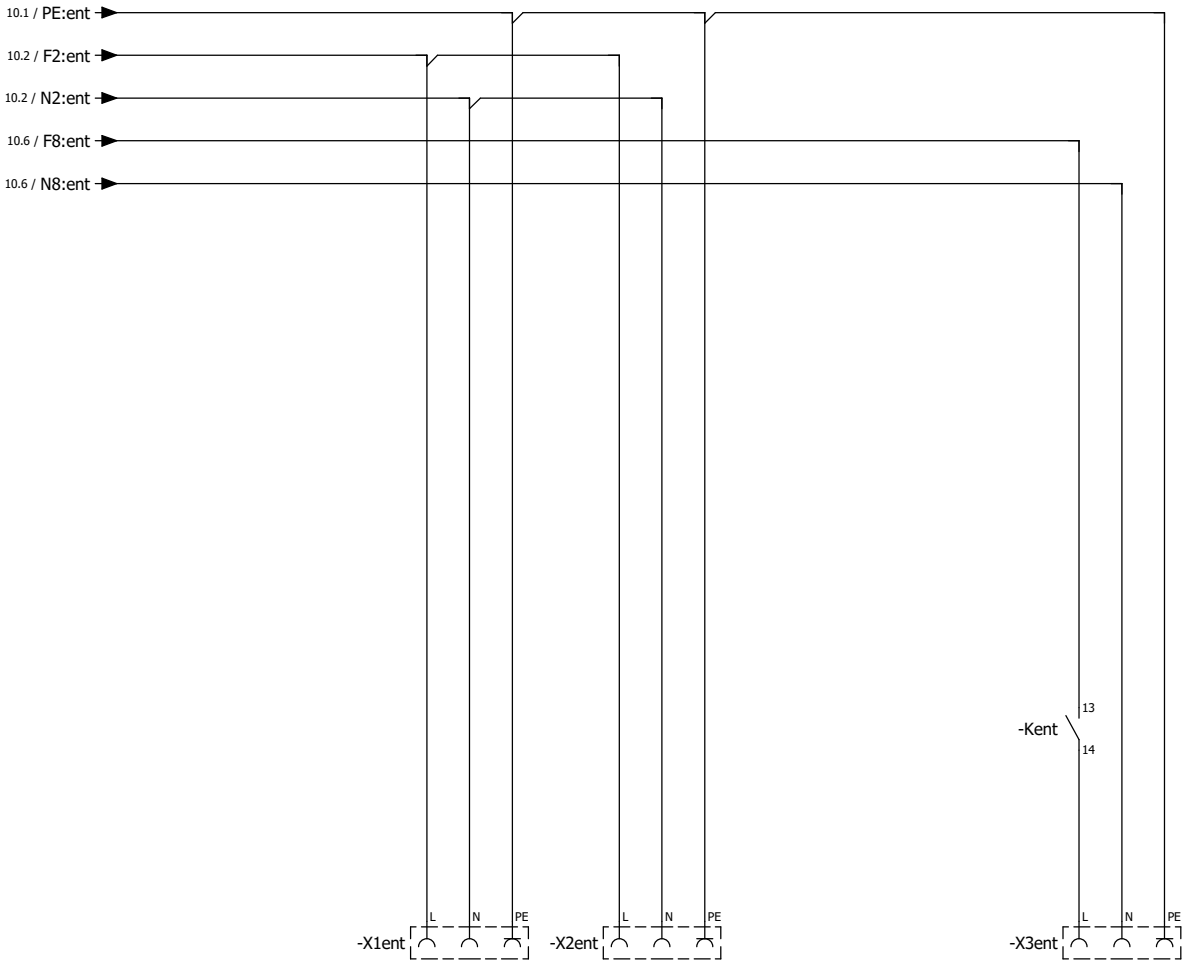
			Fecha	23/06/2020	EPLAN	EPLAN Software & Service GmbH & Co. KG			
			Resp.	SERBO					
			Probado		Instalación domótica con Raspberry Pi y ESP32 mediante MQTT			IEC_tpl003	Hoja 1
Cambio	Fecha	Nombre	Original		Sustitución por	Sustituido por			Página 1 / 36



=+/1

11

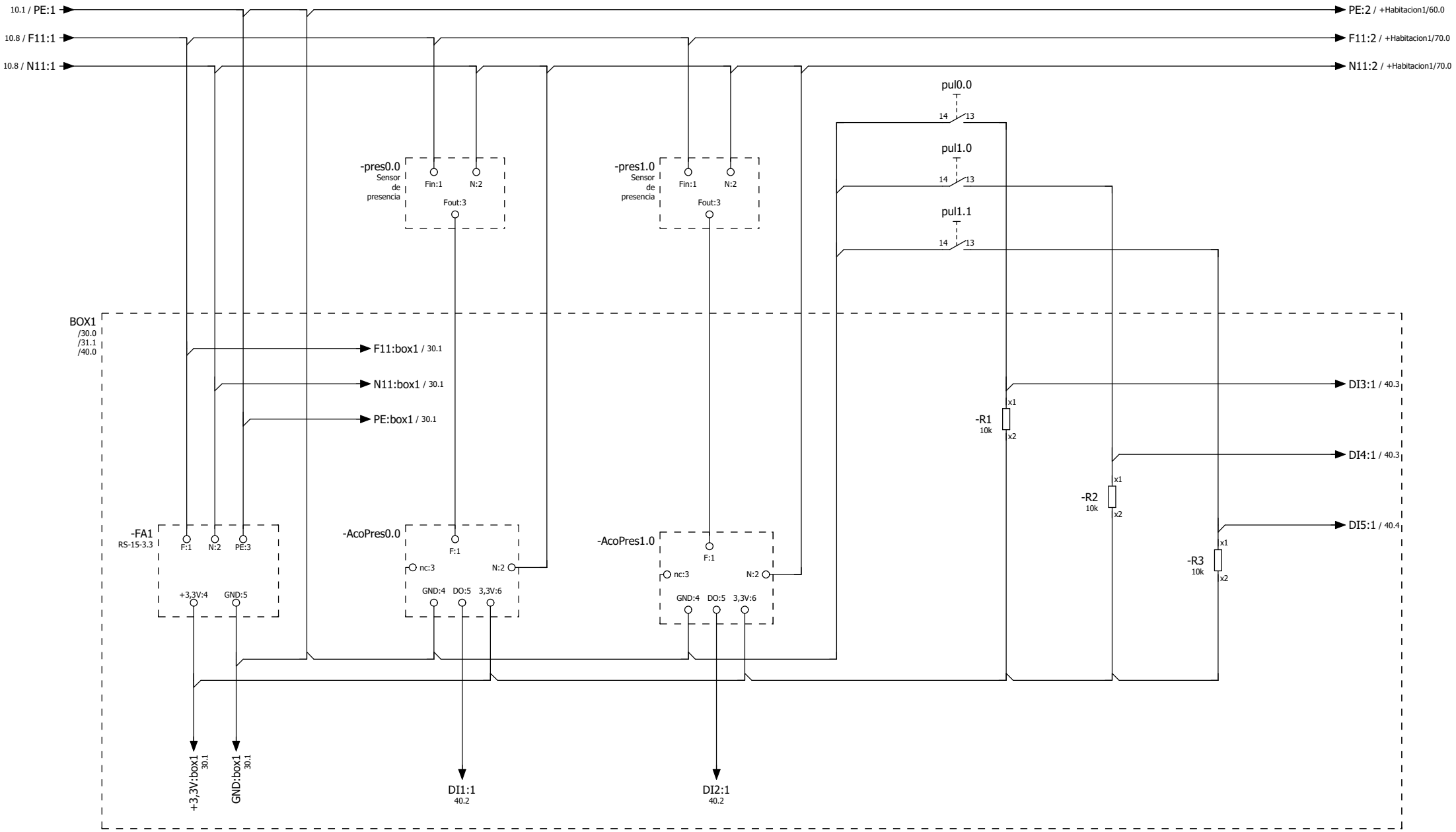
			Fecha	23/06/2020	EPLAN	EPLAN Software & Service GmbH & Co. KG		
			Resp.	SERBO				
			Probado		Instalación domótica con Raspberry Pi y ESP32 mediante MQTT		IEC_tpl003	Hoja 10
Cambio	Fecha	Nombre	Original		Sustitución por	Sustituido por		Página 2 / 36



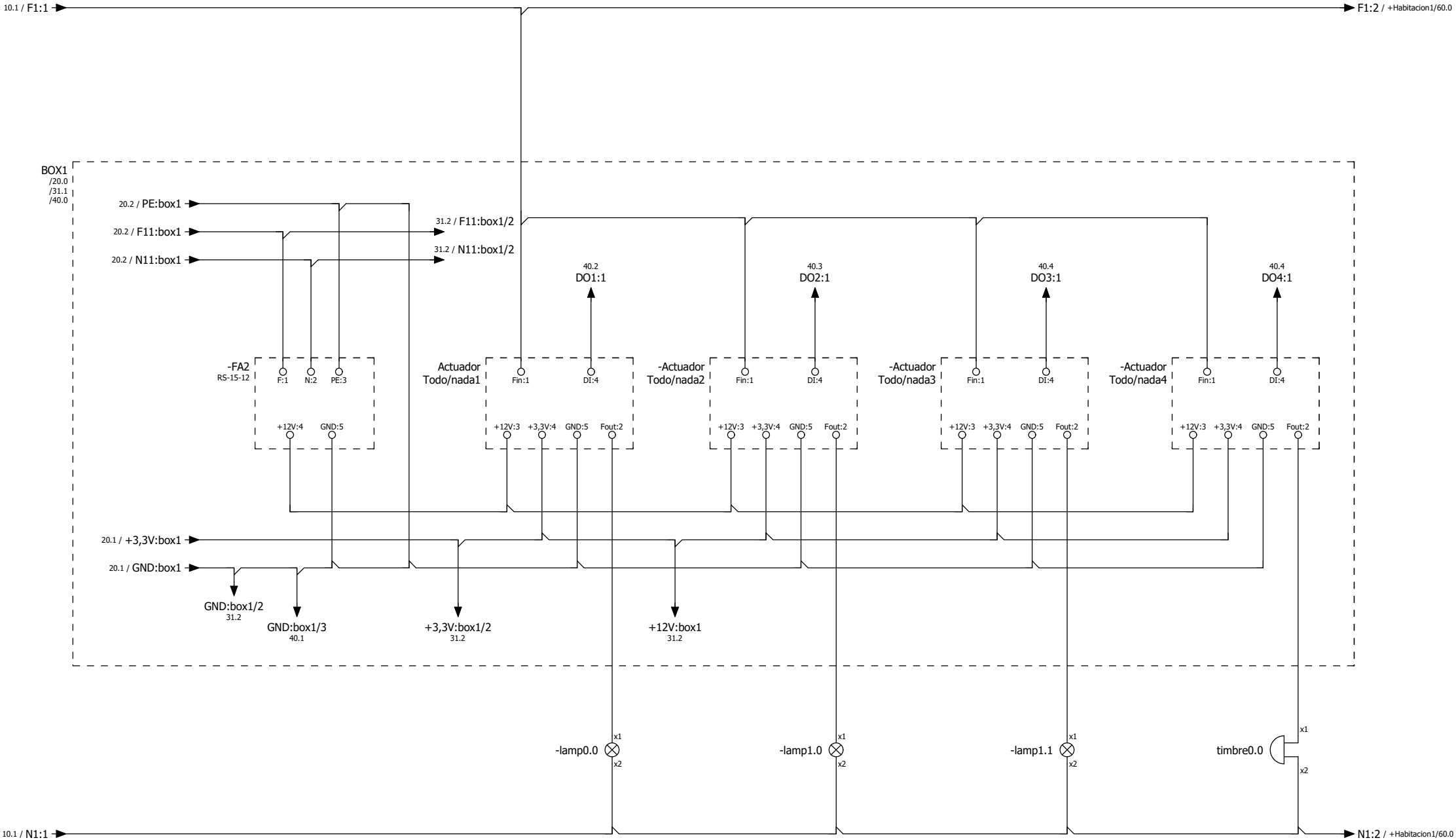
10

20

			Fecha	15/06/2020	EPLAN		EPLAN Software & Service GmbH & Co. KG		
			Resp.	SERBO					
			Probado		Instalación domótica con Raspberry Pi y ESP32 mediante MQTT			IEC_tpl003	Hoja 11
Cambio	Fecha	Nombre	Original		Sustitución por	Sustituido por			Página 3 / 36

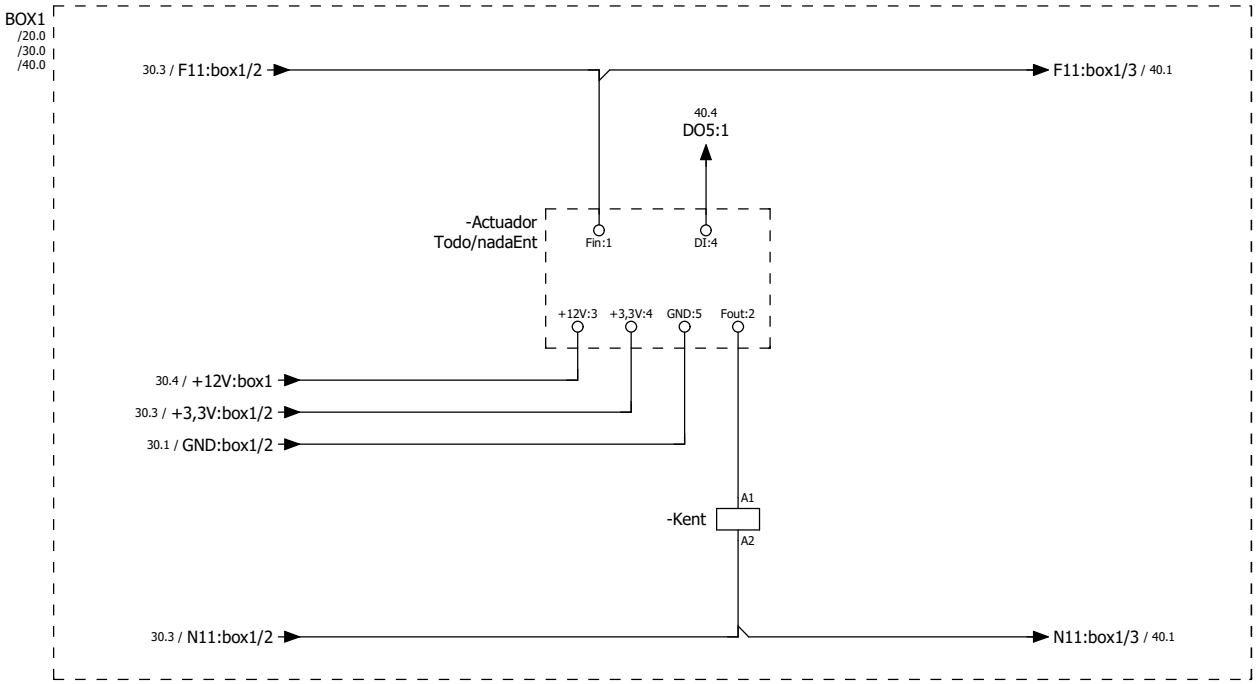


			Fecha	15/06/2020	EPLAN	EPLAN Software & Service GmbH & Co. KG			
			Resp.	SERBO					
			Probado		Instalación domótica con Raspberry Pi y ESP32 mediante MQTT			IEC_tpl003	Hoja 20
Cambio	Fecha	Nombre	Original		Sustitución por	Sustituido por			Página 4 / 36

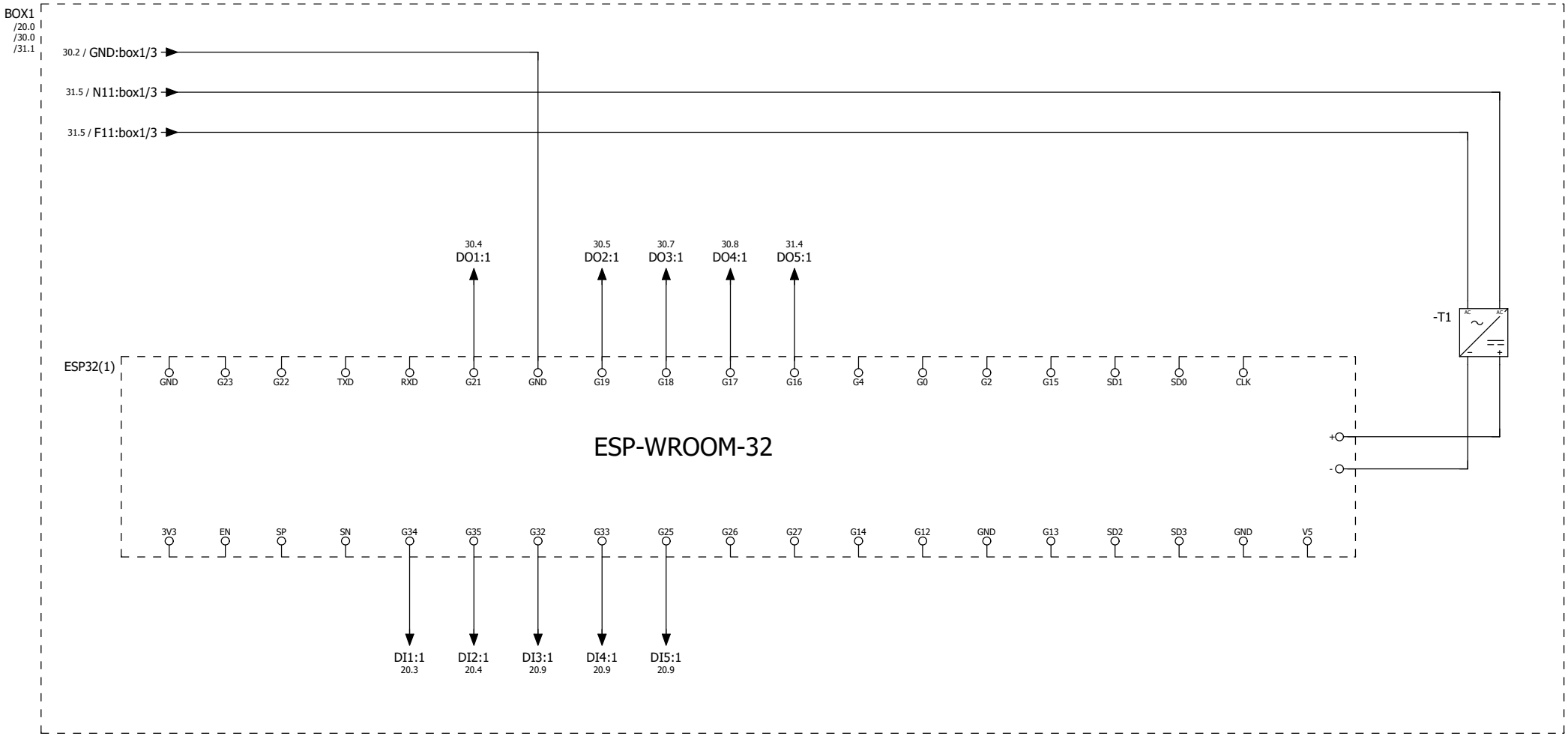


20 31

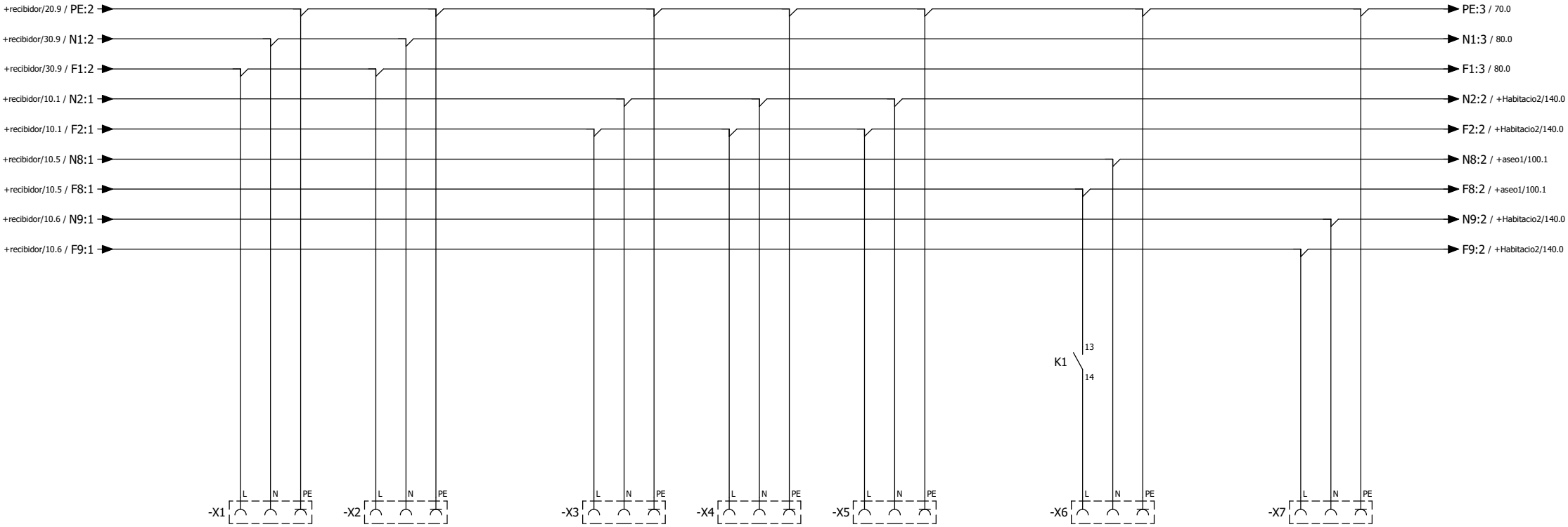
			Fecha	13/06/2020	EPLAN	EPLAN Software & Service GmbH & Co. KG			
			Resp.	SERBO					
			Probado		Instalación domótica con Raspberry Pi y ESP32 mediante MQTT			IEC_tpl003	Hoja 30
Cambio	Fecha	Nombre	Original		Sustitución por	Sustituido por			Página 5 / 36



			Fecha	13/06/2020	EPLAN		EPLAN Software & Service GmbH & Co. KG		
			Resp.	SERBO					
			Probado		Instalación domótica con Raspberry Pi y ESP32 mediante MQTT			IEC_tpl003	Hoja 31
Cambio	Fecha	Nombre	Original		Sustitución por	Sustituido por			Página 6 / 36

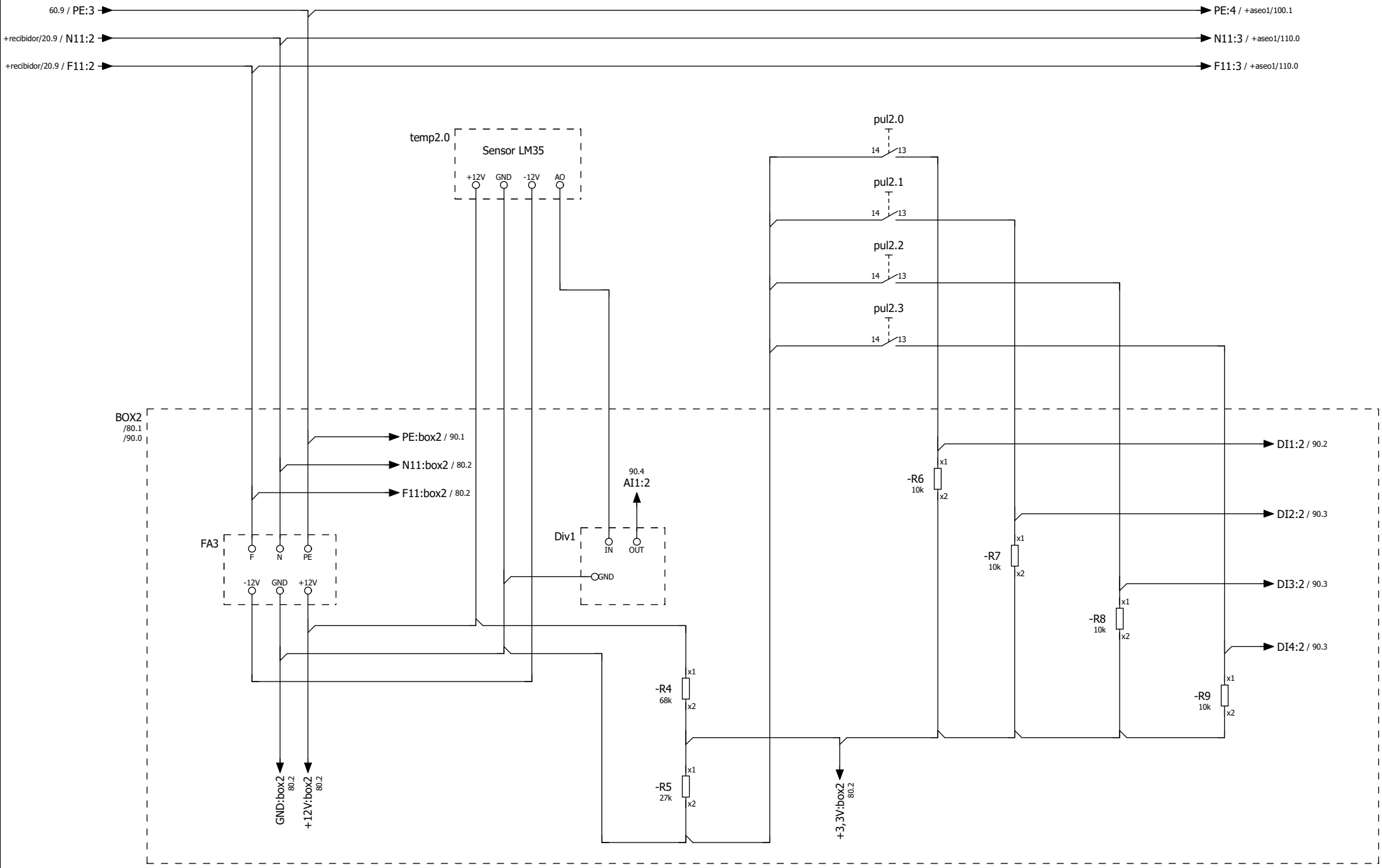


			Fecha	15/06/2020	EPLAN	EPLAN Software & Service GmbH & Co. KG		
			Resp.	SERBO				
			Probado		Instalación domótica con Raspberry Pi y ESP32 mediante MQTT		IEC_tpl003	Hoja 40
Cambio	Fecha	Nombre	Original		Sustitución por	Sustituido por		Página 7 / 36



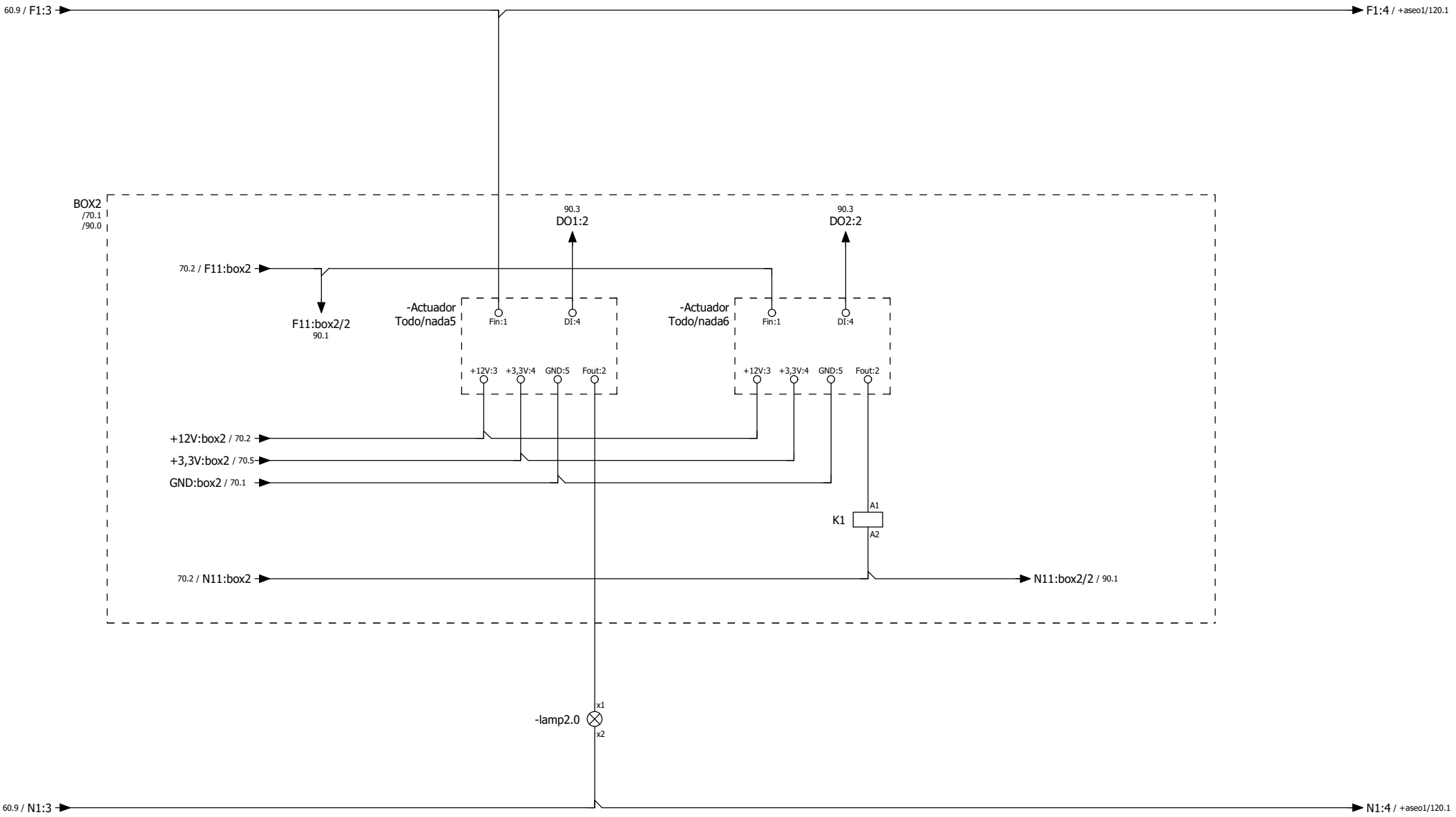
+recibidor/40

			Fecha	15/06/2020	EPLAN	EPLAN Software & Service GmbH & Co. KG			
			Resp.	SERBO					
			Probado		Instalación domótica con Raspberry Pi y ESP32 mediante MQTT			IEC_tpl003	Hoja 60
Cambio	Fecha	Nombre	Original		Sustitución por	Sustituido por			Página 8 / 36



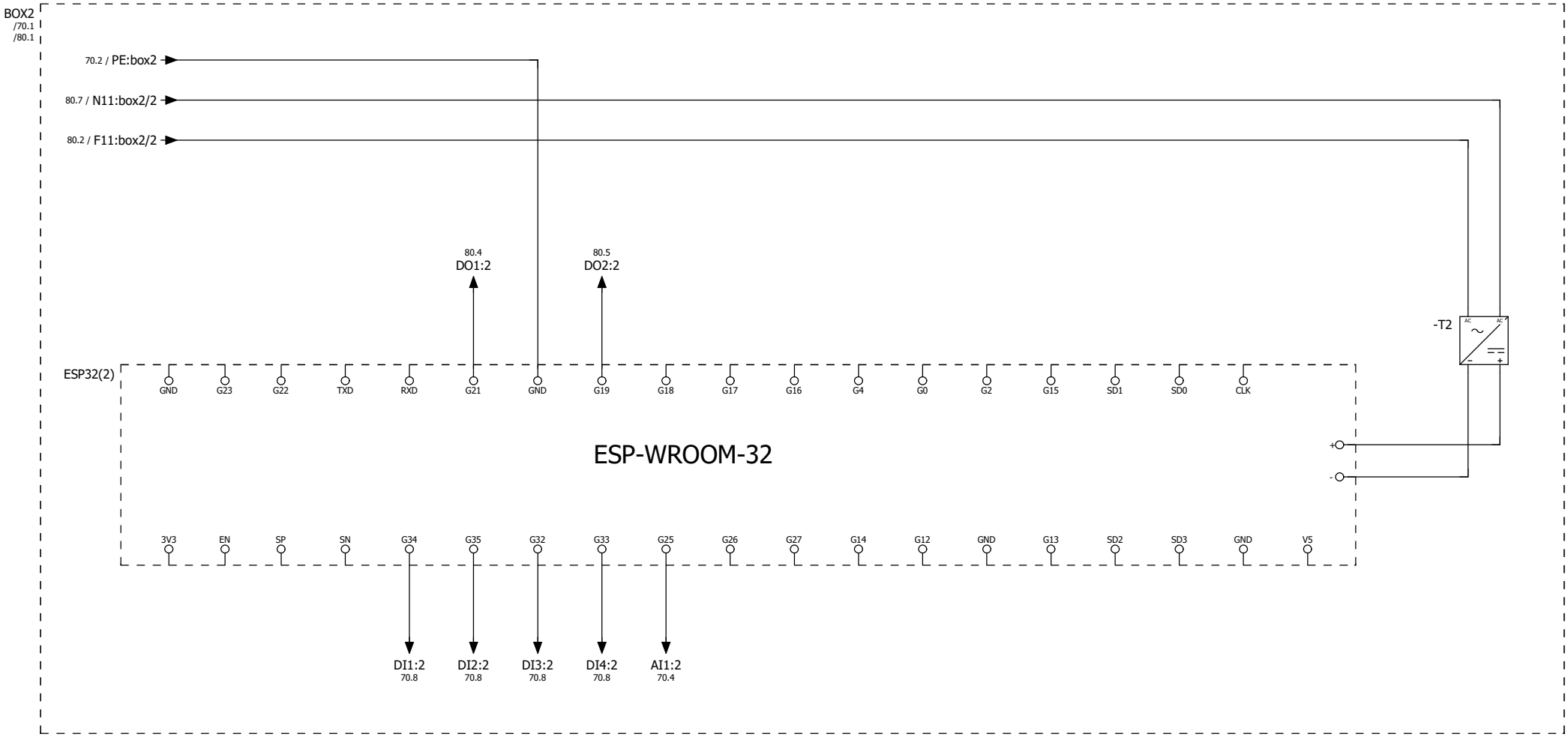
60 80

			Fecha	14/06/2020	EPLAN		EPLAN Software & Service GmbH & Co. KG		
			Resp.	SERBO					
			Probado		Instalación domótica con Raspberry Pi y ESP32 mediante MQTT			IEC_tpl003	Hoja 70
Cambio	Fecha	Nombre	Original		Sustitución por	Sustituido por			Página 9 / 36

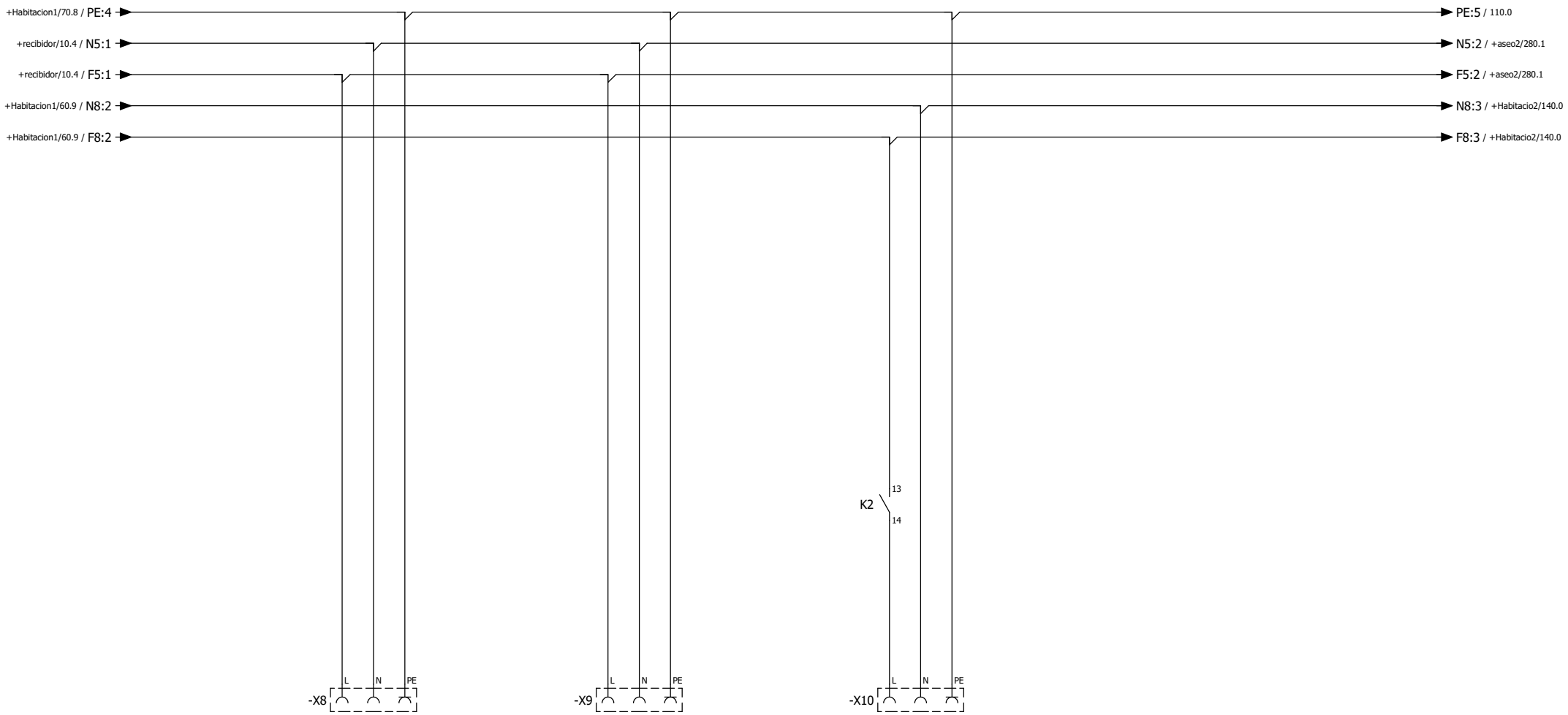


70 90

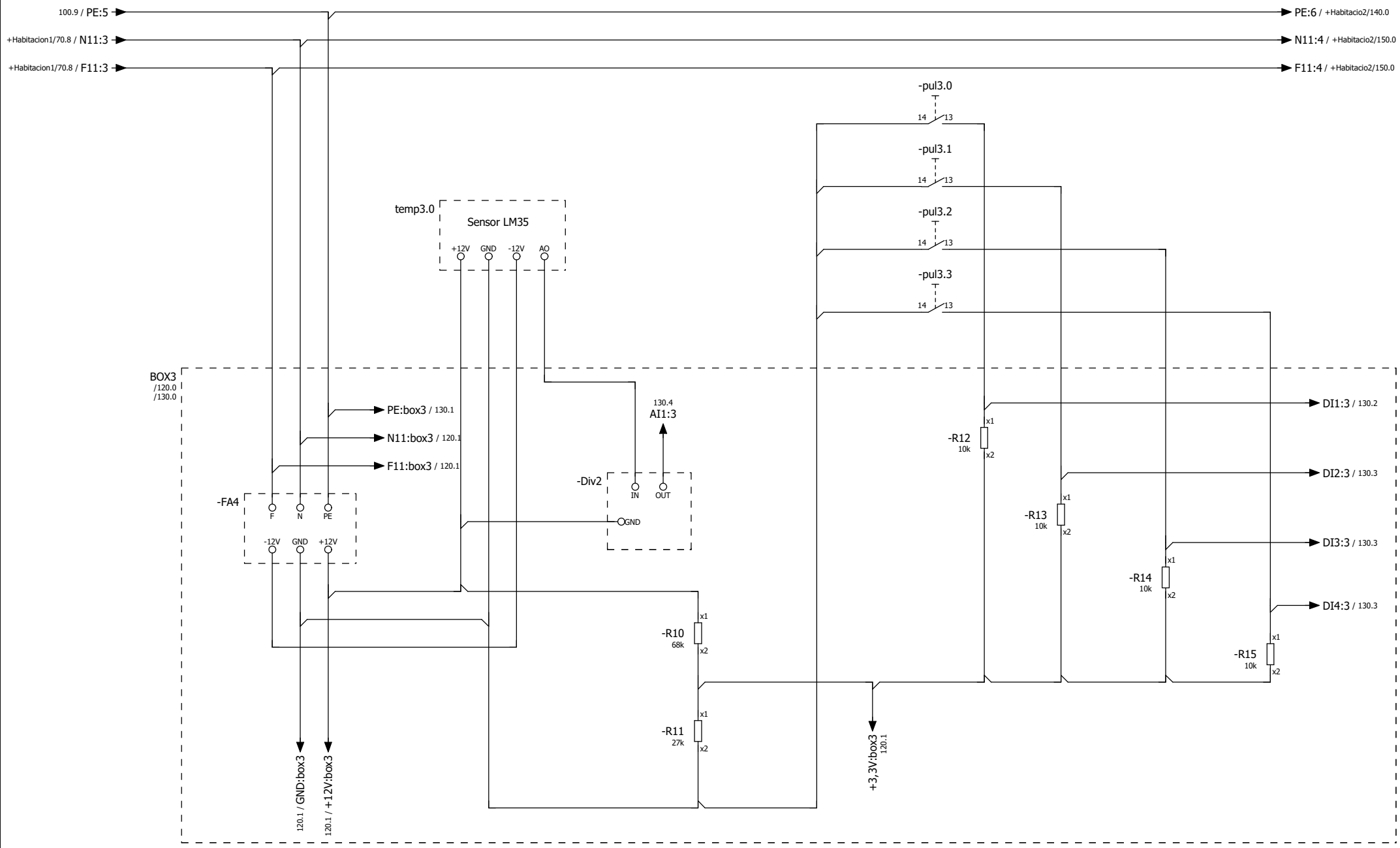
			Fecha	15/06/2020	EPLAN		EPLAN Software & Service GmbH & Co. KG		
			Resp.	SERBO					
			Probado		Instalación domótica con Raspberry Pi y ESP32 mediante MQTT			IEC_tpl003	Hoja 80
Cambio	Fecha	Nombre	Original		Sustitución por	Sustituido por			Página 10 / 36



			Fecha	15/06/2020	EPLAN	EPLAN Software & Service GmbH & Co. KG		
			Resp.	SERBO				
			Probado		Instalación domótica con Raspberry Pi y ESP32 mediante MQTT		IEC_tpl003	Hoja 90
Cambio	Fecha	Nombre	Original		Sustitución por	Sustituido por		Página 11 / 36

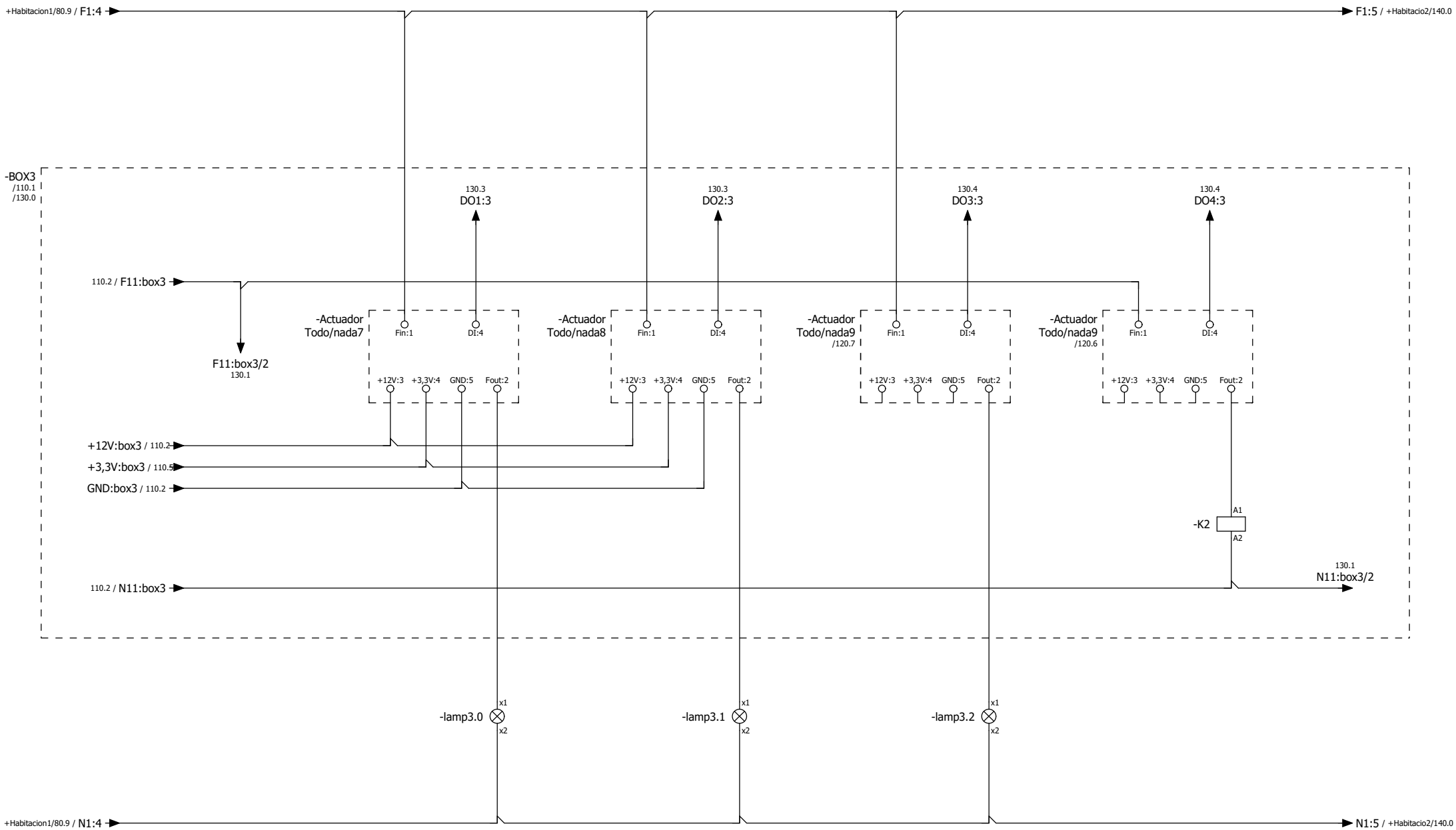


			Fecha	16/06/2020	EPLAN		EPLAN Software & Service GmbH & Co. KG		
			Resp.	SERBO					
			Probado		Instalación domótica con Raspberry Pi y ESP32 mediante MQTT			IEC_tpl003	Hoja 100
Cambio	Fecha	Nombre	Original		Sustitución por	Sustituido por			Página 12 / 36



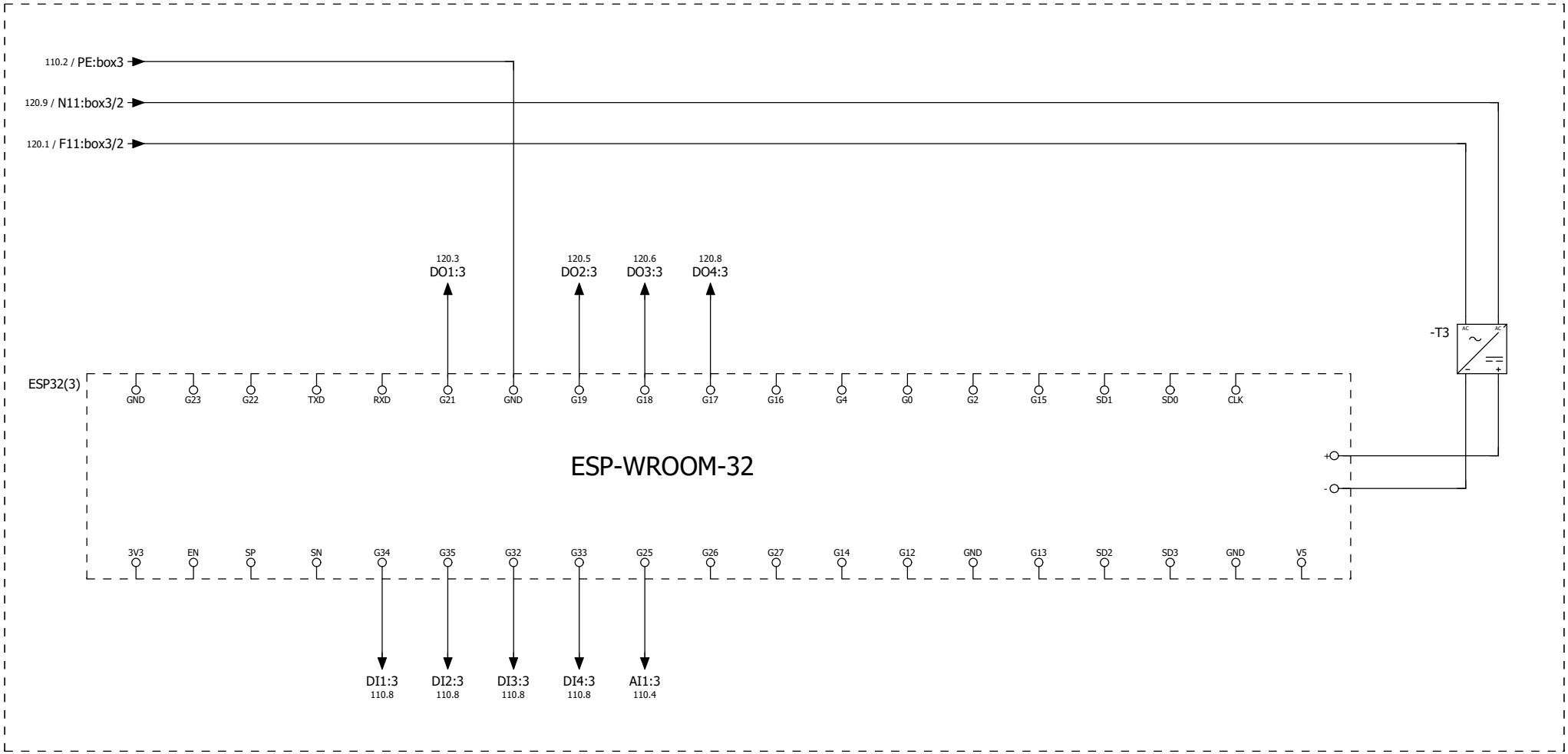
100 120

			Fecha	14/06/2020	EPLAN		EPLAN Software & Service GmbH & Co. KG		
			Resp.	SERBO					
			Probado		Instalación domótica con Raspberry Pi y ESP32 mediante MQTT			IEC_tpl003	Hoja 110
Cambio	Fecha	Nombre	Original		Sustitución por	Sustituido por			Página 13 / 36

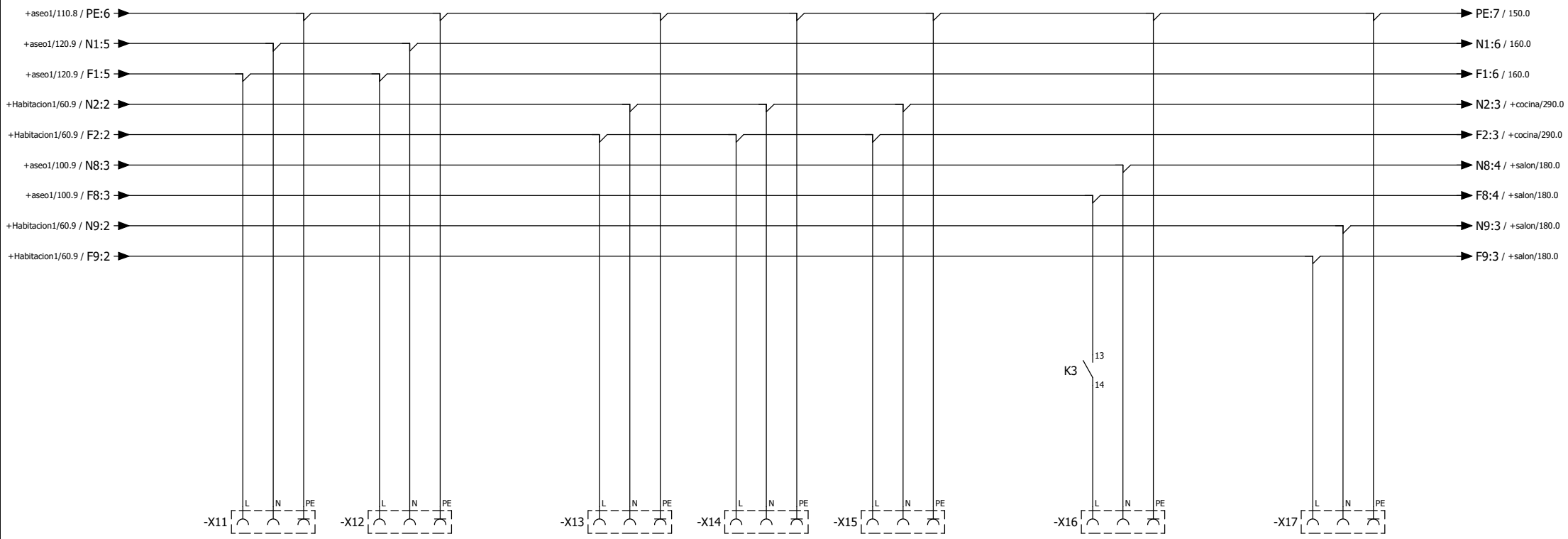


110 130

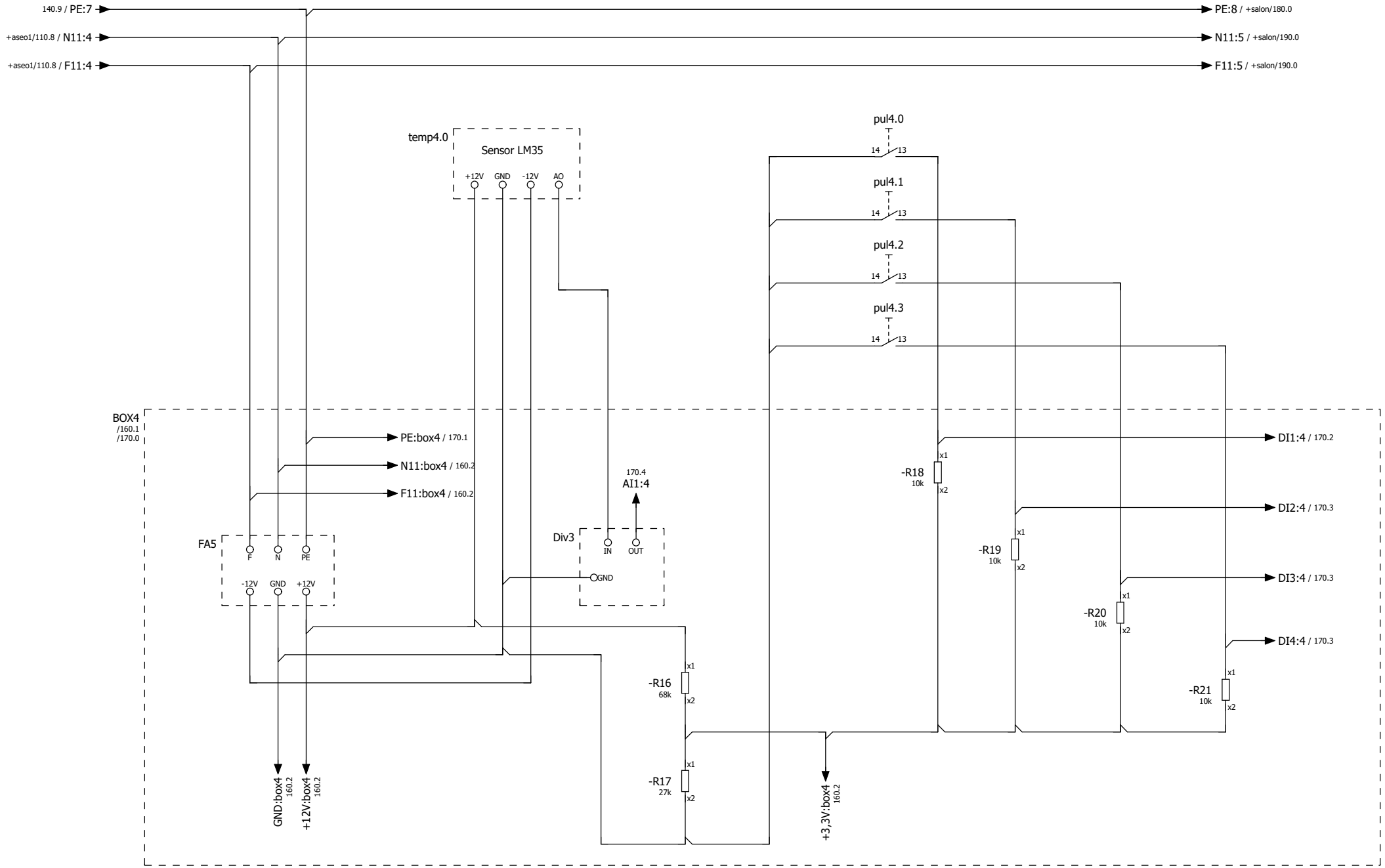
			Fecha	15/06/2020	EPLAN		EPLAN Software & Service GmbH & Co. KG		
			Resp.	SERBO					
			Probado		Instalación domótica con Raspberry Pi y ESP32 mediante MQTT			IEC_tpl003	Hoja 120
Cambio	Fecha	Nombre	Original		Sustitución por	Sustituido por			Página 14 / 36



			Fecha	13/06/2020	EPLAN	EPLAN Software & Service GmbH & Co. KG		
			Resp.	SERBO				
			Probado		Instalación domótica con Raspberry Pi y ESP32 mediante MQTT		IEC_tpl003	Hoja 130
Cambio	Fecha	Nombre	Original		Sustitución por	Sustituido por		Página 15 / 36

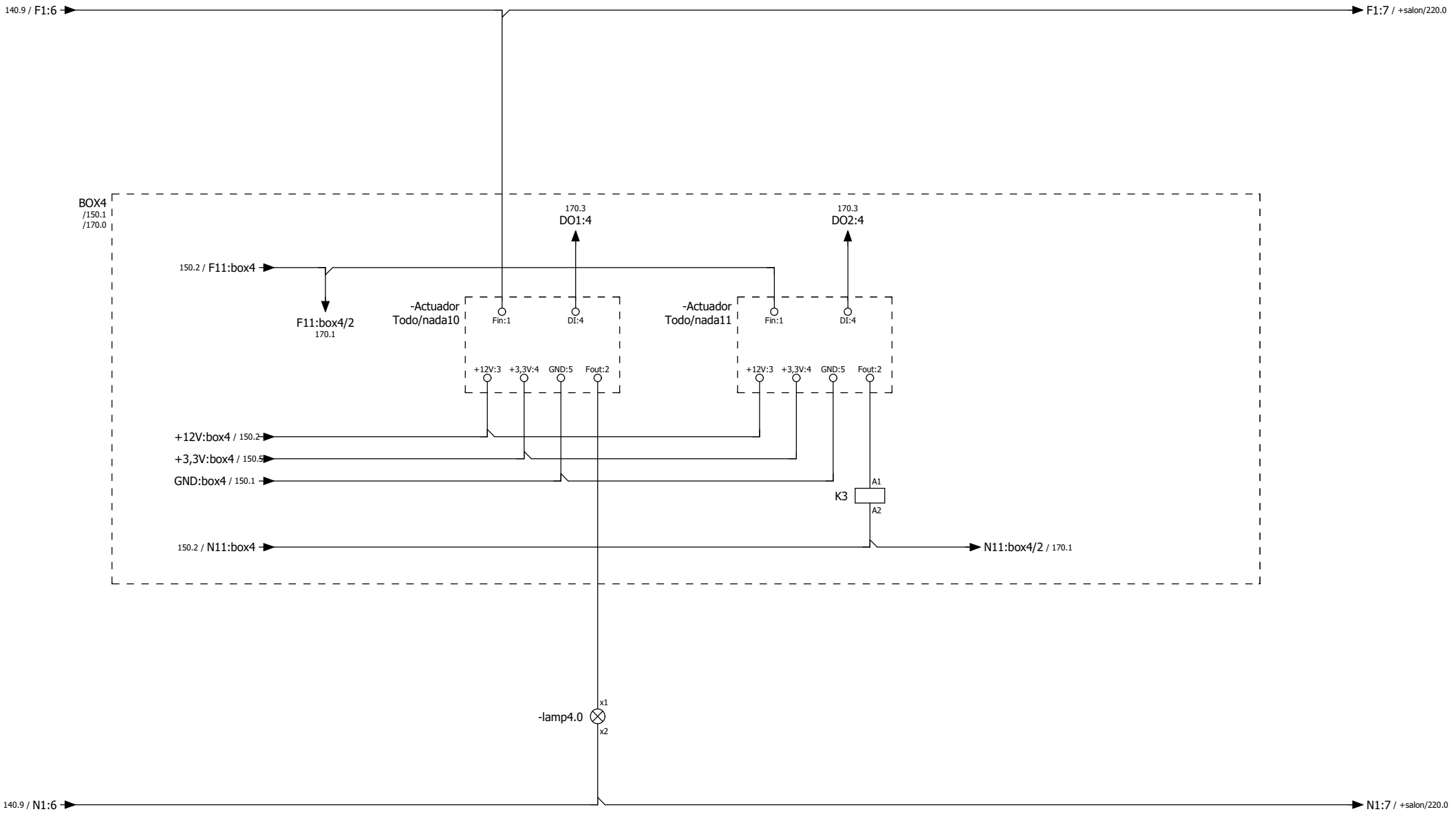


			Fecha	16/06/2020	EPLAN		EPLAN Software & Service GmbH & Co. KG		
			Resp.	SERBO					
			Probado		Instalación domótica con Raspberry Pi y ESP32 mediante MQTT			IEC_tpl003	Hoja 140
Cambio	Fecha	Nombre	Original		Sustitución por	Sustituido por			Página 16 / 36



140 160

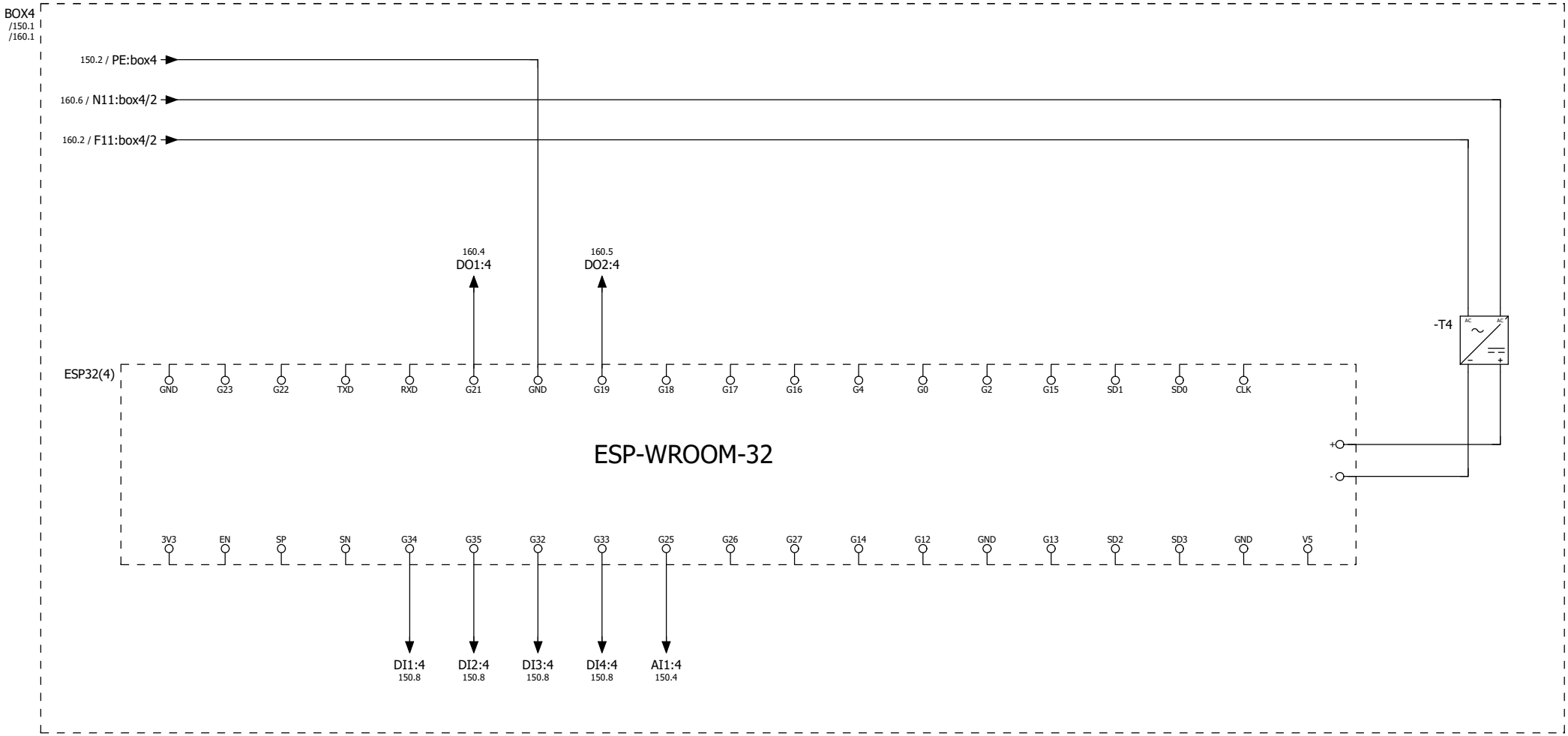
			Fecha	15/06/2020	EPLAN		EPLAN Software & Service GmbH & Co. KG		
			Resp.	SERBO					
			Probado		Instalación domótica con Raspberry Pi y ESP32 mediante MQTT			IEC_tpl003	Hoja 150
Cambio	Fecha	Nombre	Original		Sustitución por	Sustituido por			Página 17 / 36



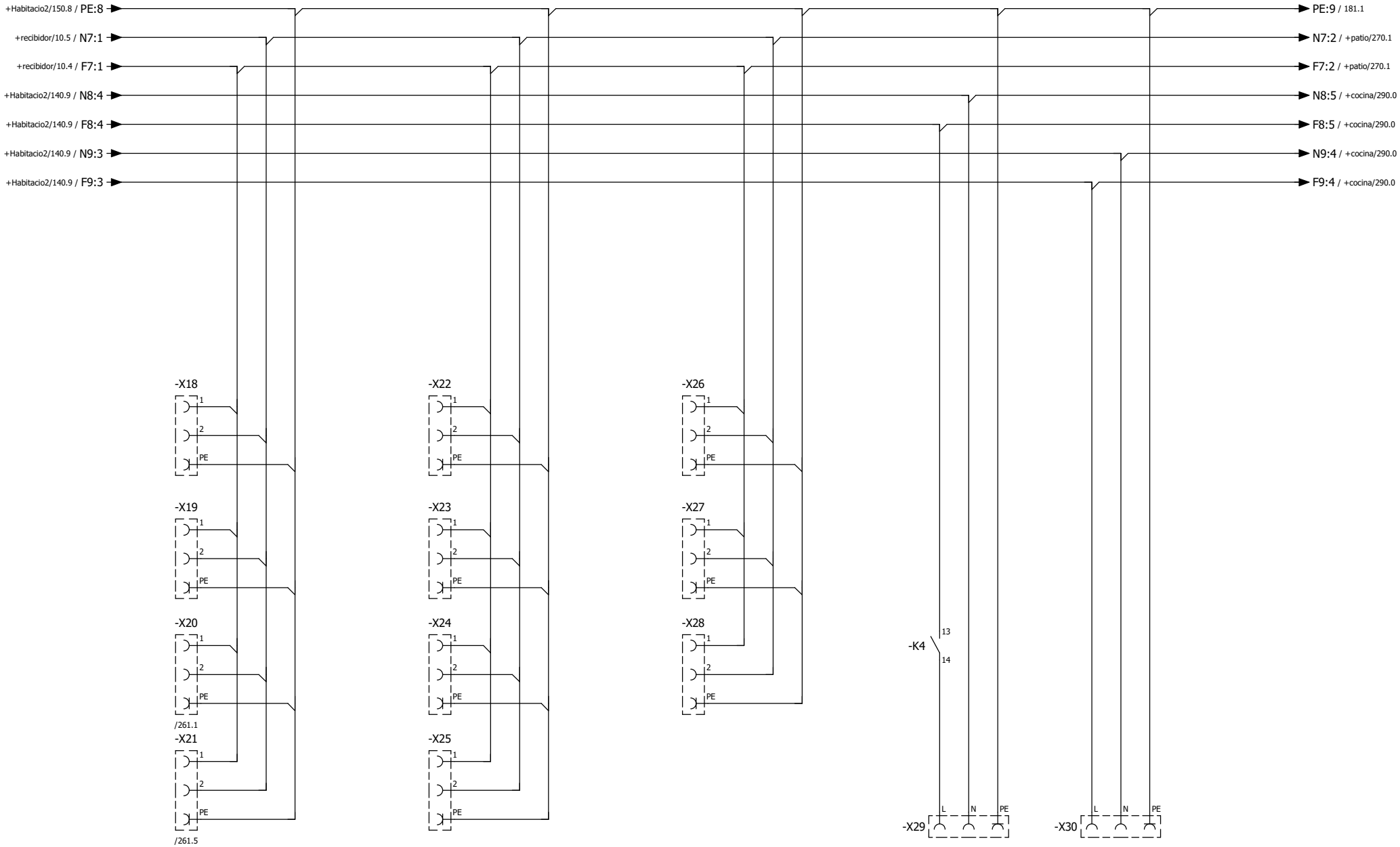
150

170

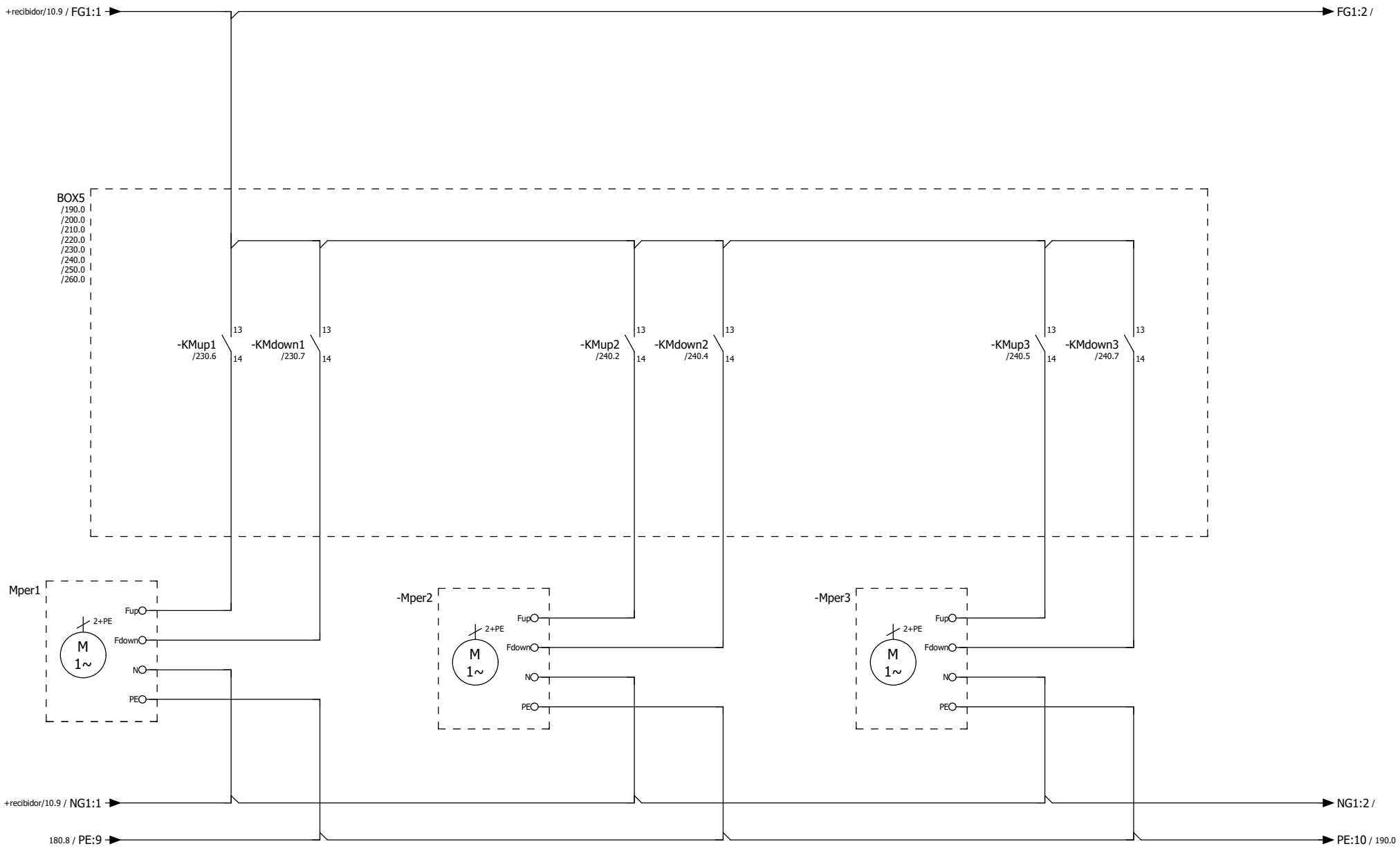
			Fecha	15/06/2020	EPLAN		EPLAN Software & Service GmbH & Co. KG		
			Resp.	SERBO					
			Probado		Instalación domótica con Raspberry Pi y ESP32 mediante MQTT			IEC_tpl003	Hoja 160
Cambio	Fecha	Nombre	Original		Sustitución por	Sustituido por			Página 18 / 36



			Fecha	13/06/2020	EPLAN	EPLAN Software & Service GmbH & Co. KG		
			Resp.	SERBO				
			Probado		Instalación domótica con Raspberry Pi y ESP32 mediante MQTT		IEC_tpl003	Hoja 170
Cambio	Fecha	Nombre	Original		Sustitución por	Sustituido por		Página 19 / 36

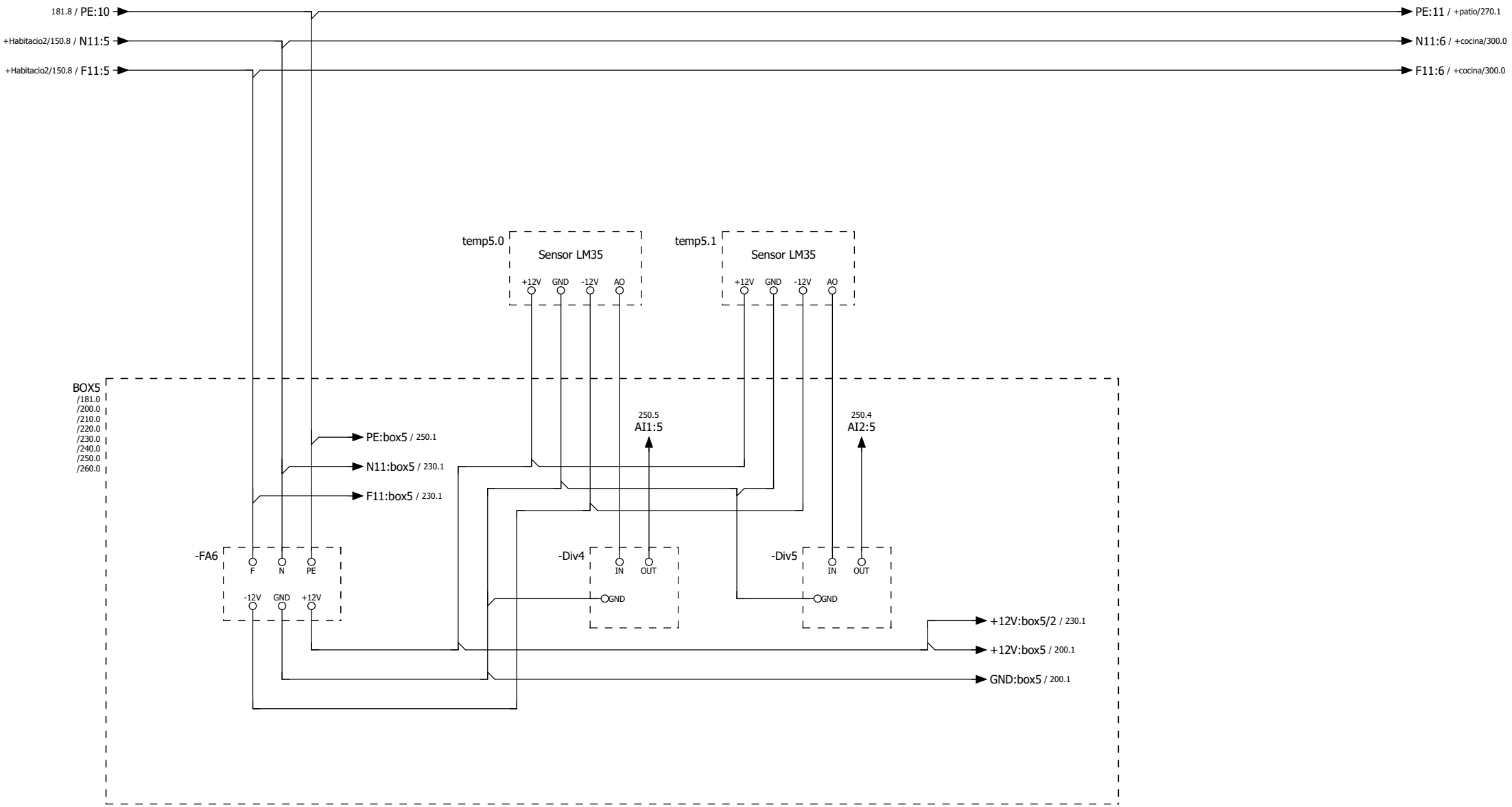


			Fecha	28/07/2020	EPLAN		EPLAN Software & Service GmbH & Co. KG	
			Resp.	SERBO				
			Probado		Instalación domótica con Raspberry Pi y ESP32 mediante MQTT			IEC_tpl003 Hoja 180
Cambio	Fecha	Nombre	Original		Sustitución por	Sustituido por		Página 20 / 36



180 190

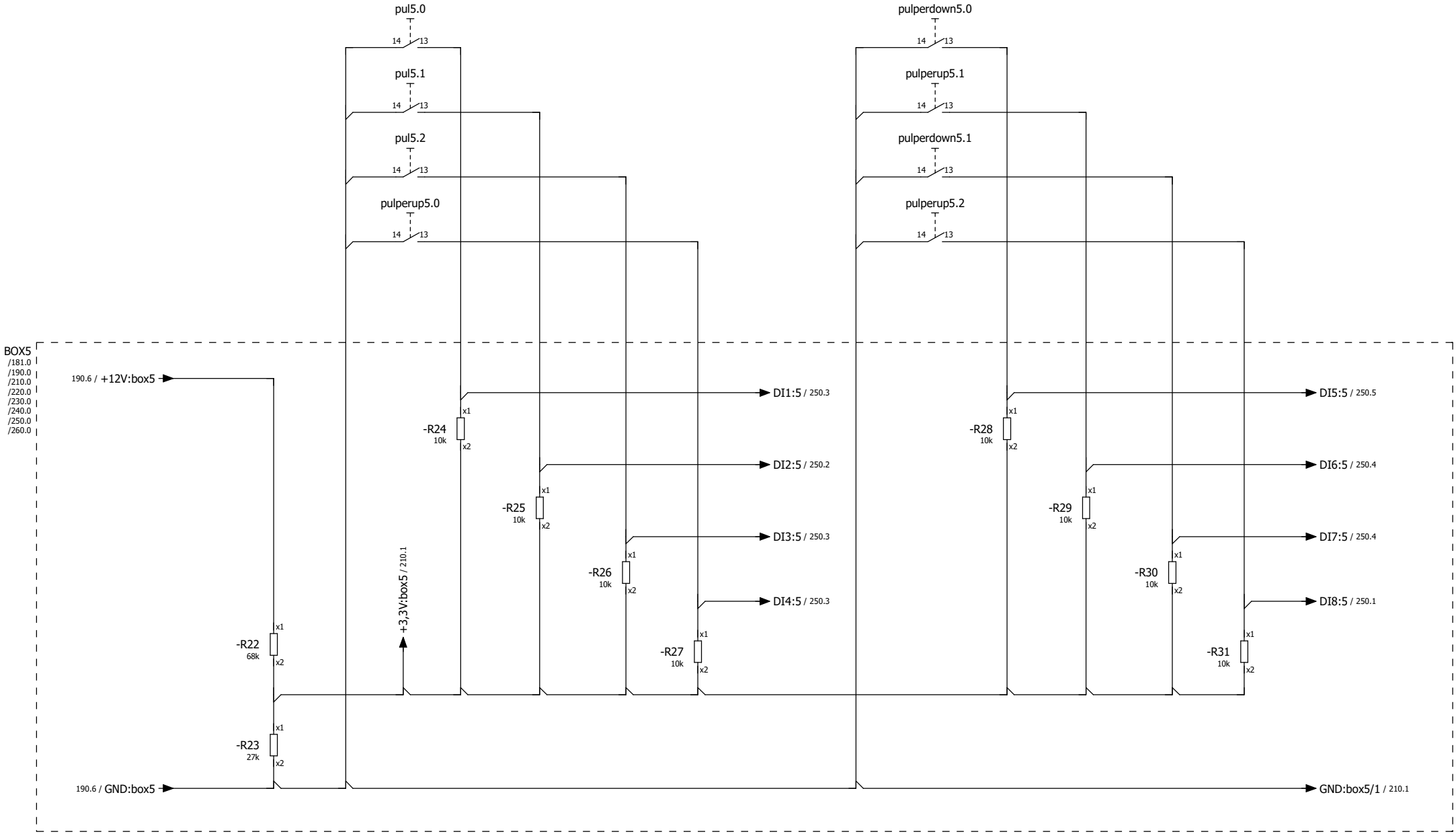
			Fecha	15/06/2020	EPLAN	EPLAN Software & Service GmbH & Co. KG			
			Resp.	SERBO					
			Probado		Instalación domótica con Raspberry Pi y ESP32 mediante MQTT			IEC_tpl003	Hoja 181
Cambio	Fecha	Nombre	Original		Sustitución por	Sustituido por			Página 21 / 36



181

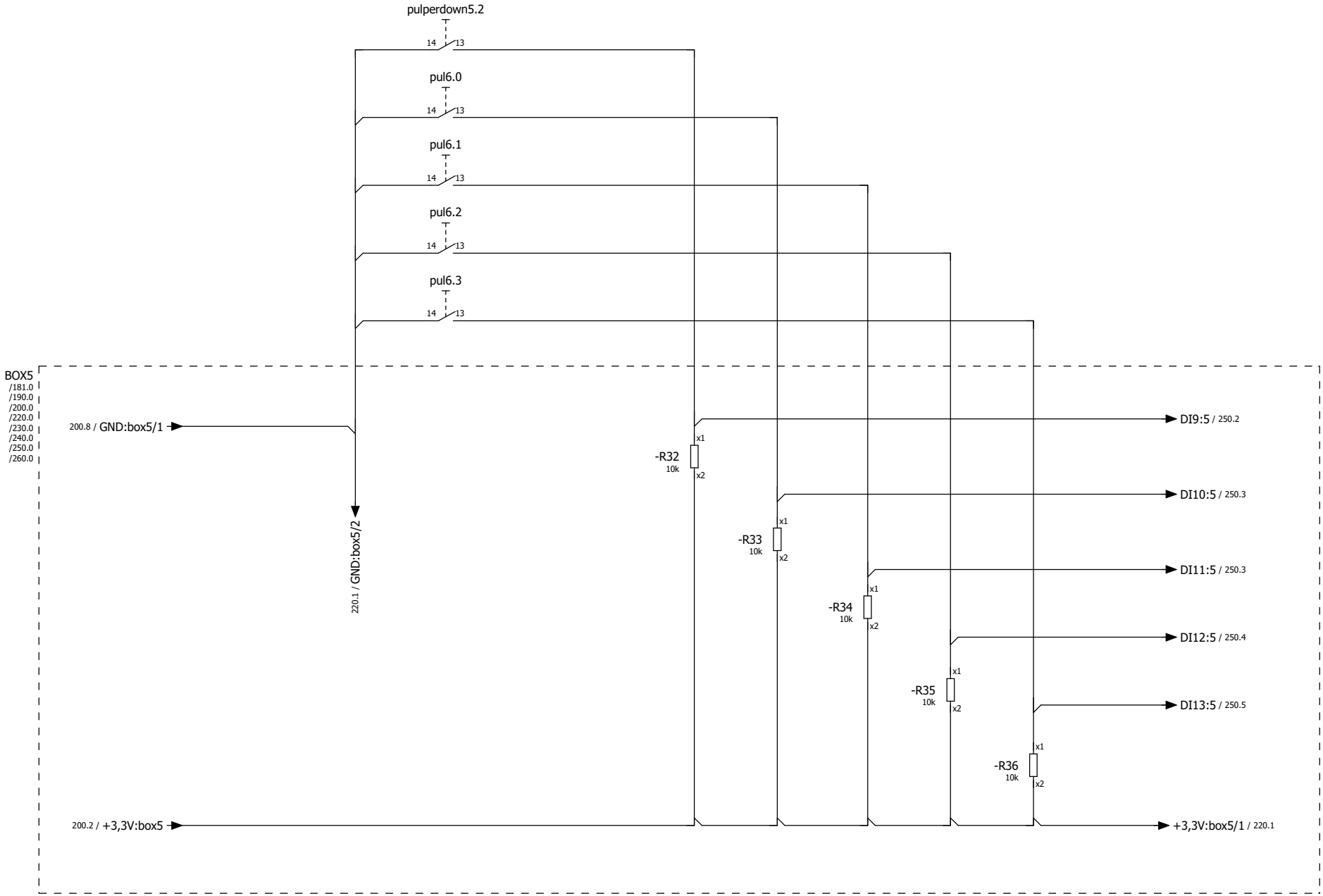
200

			Fecha	17/06/2020	EPLAN		EPLAN Software & Service GmbH & Co. KG		
			Resp.	SERBO					
			Probado		Instalación domótica con Raspberry Pi y ESP32 mediante MQTT			IEC_tpl003	Hoja 190
Cambio	Fecha	Nombre	Original		Sustitución por	Sustituido por			Página 22 / 36



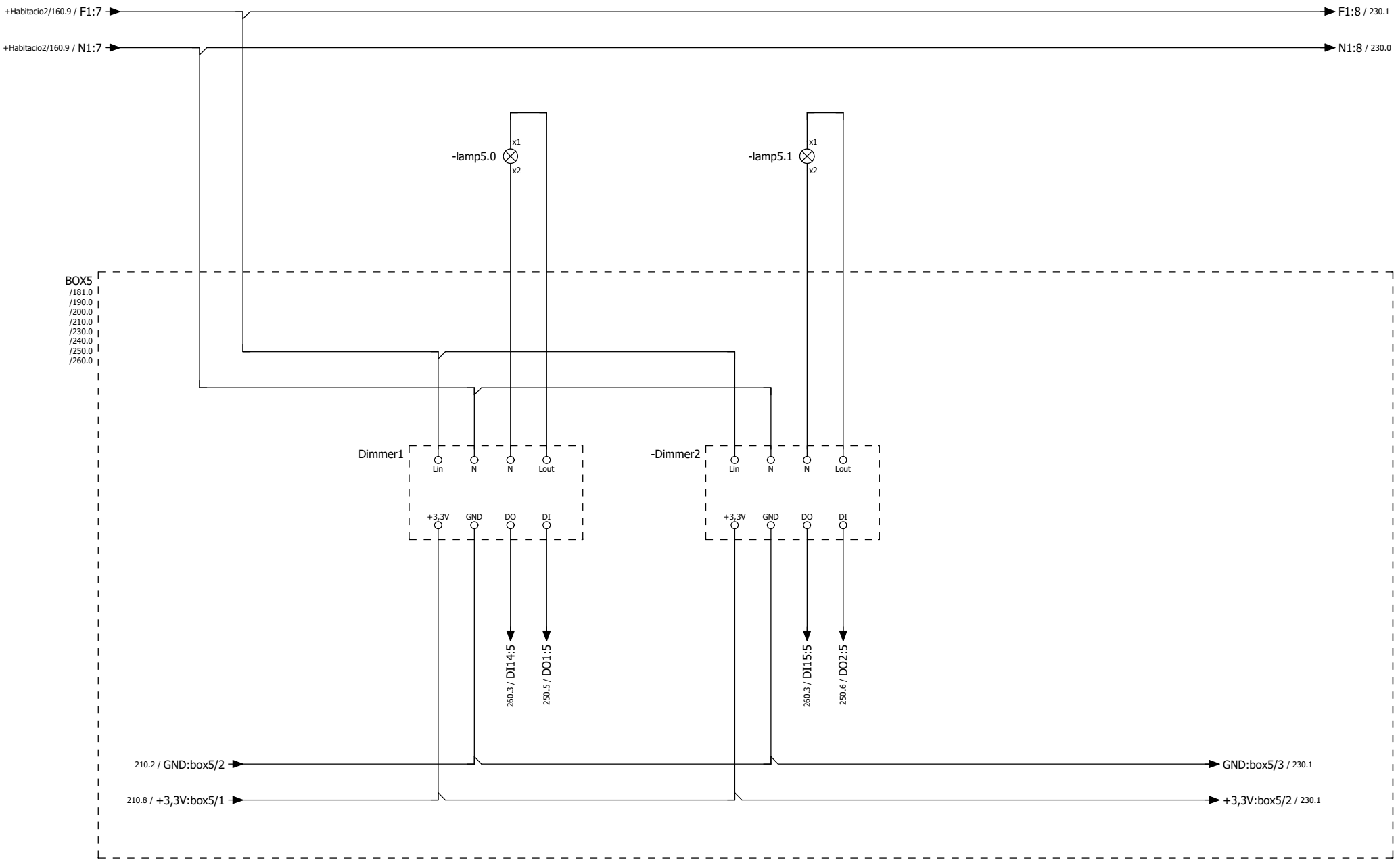
190 210

			Fecha	15/06/2020	EPLAN		EPLAN Software & Service GmbH & Co. KG		
			Resp.	SERBO					
			Probado		Instalación domótica con Raspberry Pi y ESP32 mediante MQTT			IEC_tpl003	Hoja 200
Cambio	Fecha	Nombre	Original		Sustitución por	Sustituido por			Página 23 / 36



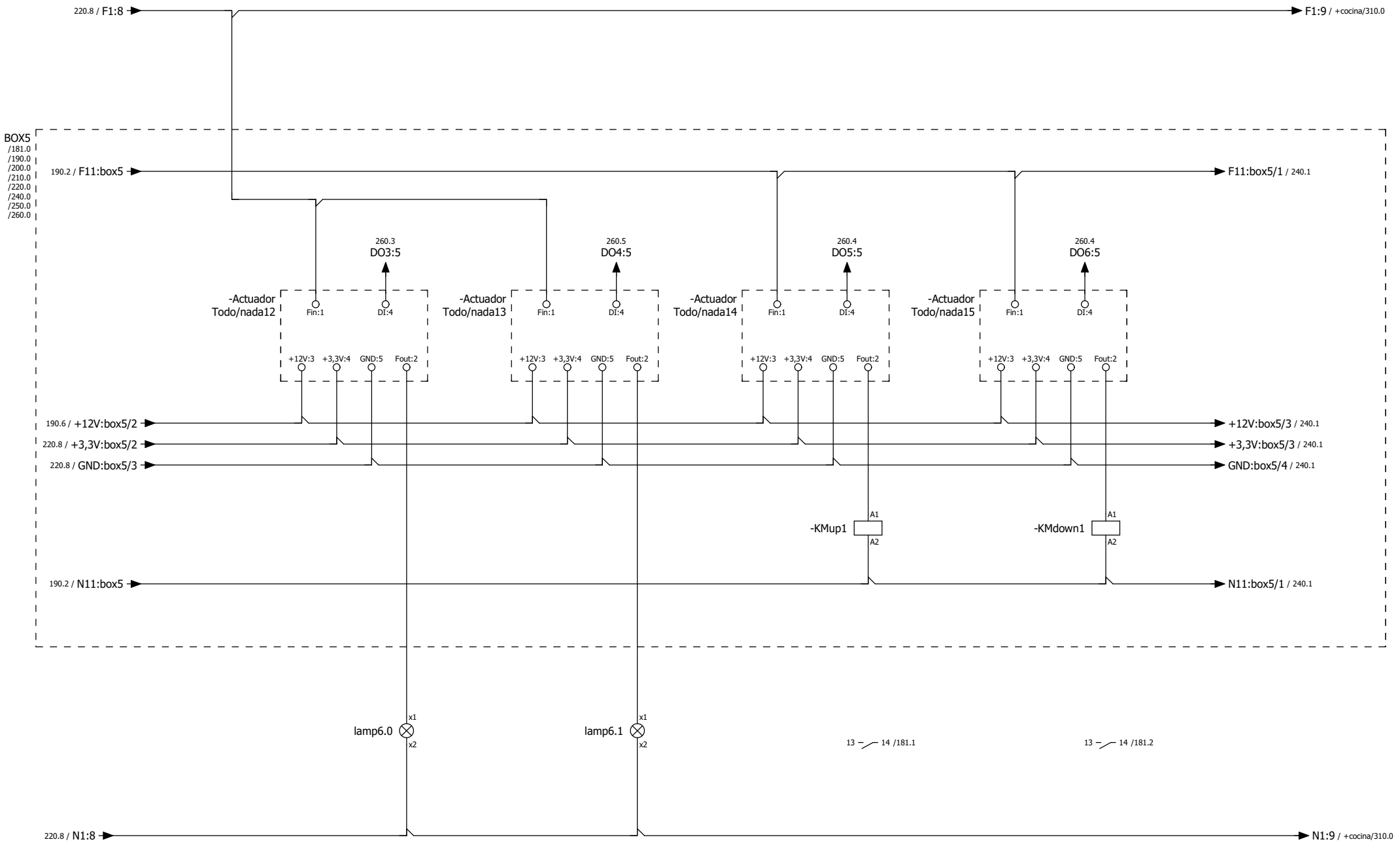
200 220

			Fecha	15/06/2020	EPLAN	EPLAN Software & Service GmbH & Co. KG		
			Resp.	SERBO				
			Probado		Instalación domótica con Raspberry Pi y ESP32 mediante MQTT		IEC_tpl003	Hoja 210
Cambio	Fecha	Nombre	Original		Sustitución por	Sustituido por		Página 24 / 36



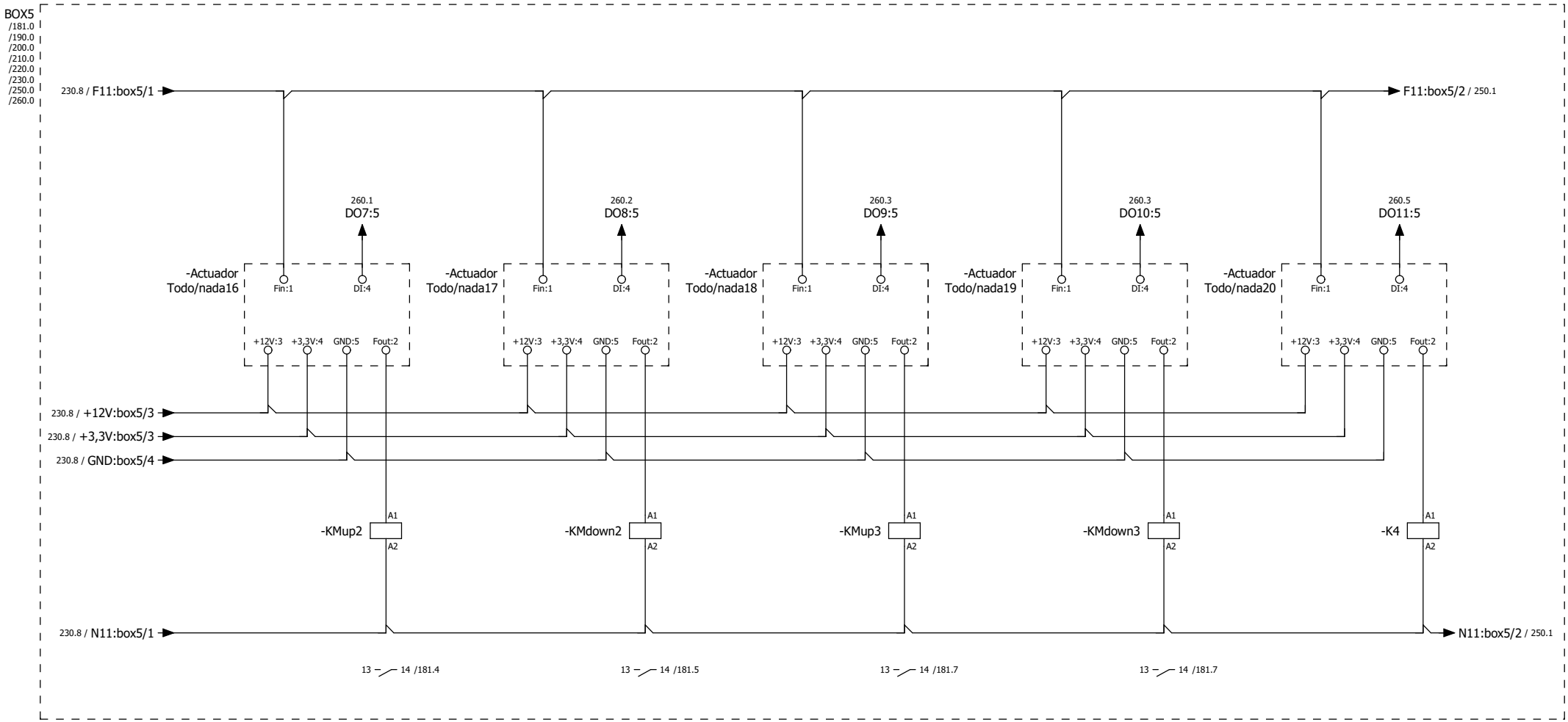
210 230

			Fecha	15/06/2020	EPLAN		EPLAN Software & Service GmbH & Co. KG		
			Resp.	SERBO					
			Probado		Instalación domótica con Raspberry Pi y ESP32 mediante MQTT			IEC_tpl003	Hoja 220
Cambio	Fecha	Nombre	Original		Sustitución por	Sustituido por			Página 25 / 36



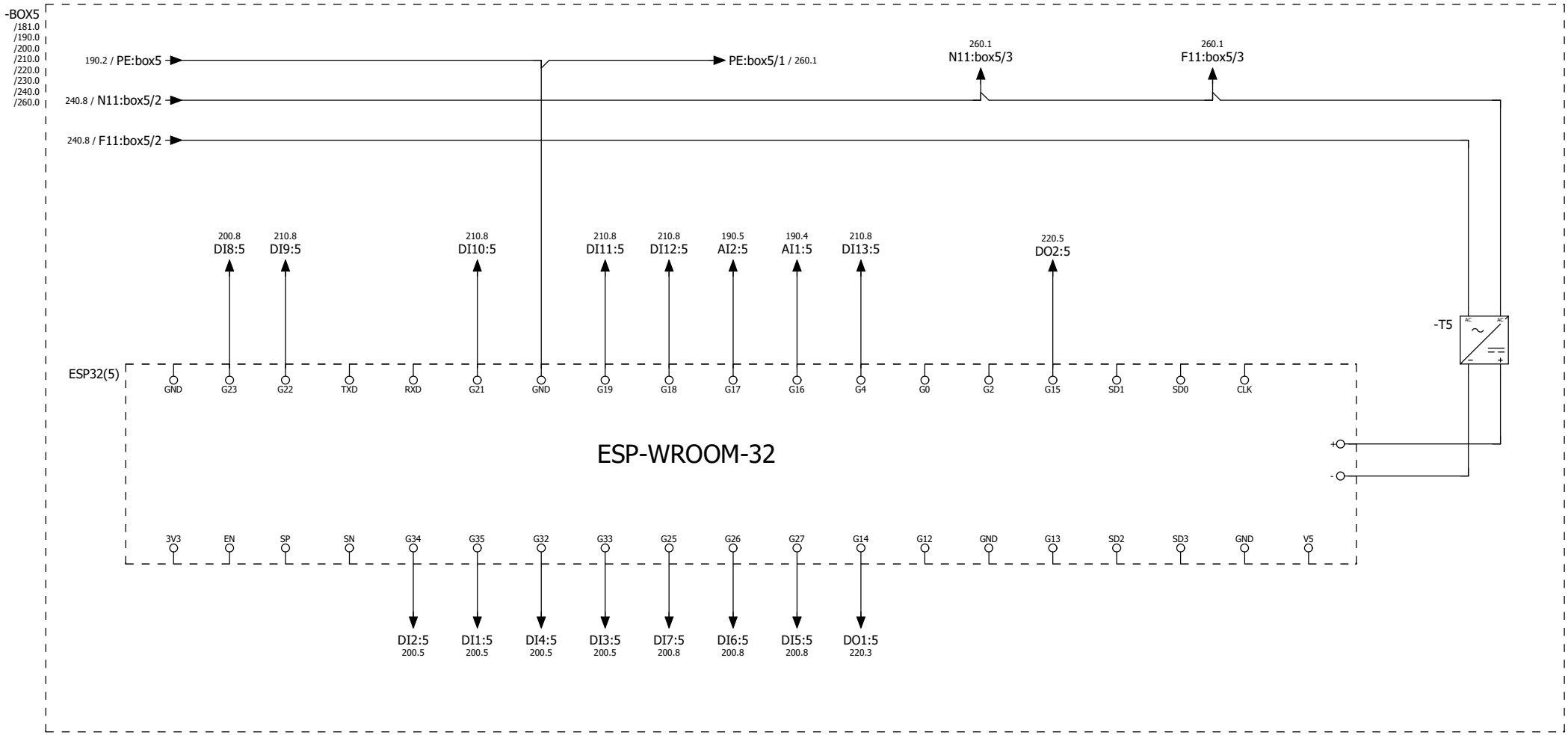
220 240

			Fecha	28/07/2020	EPLAN		EPLAN Software & Service GmbH & Co. KG		
			Resp.	SERBO					
			Probado		Instalación domótica con Raspberry Pi y ESP32 mediante MQTT			IEC_tpl003	Hoja 230
Cambio	Fecha	Nombre	Original		Sustitución por	Sustituido por			Página 26 / 36



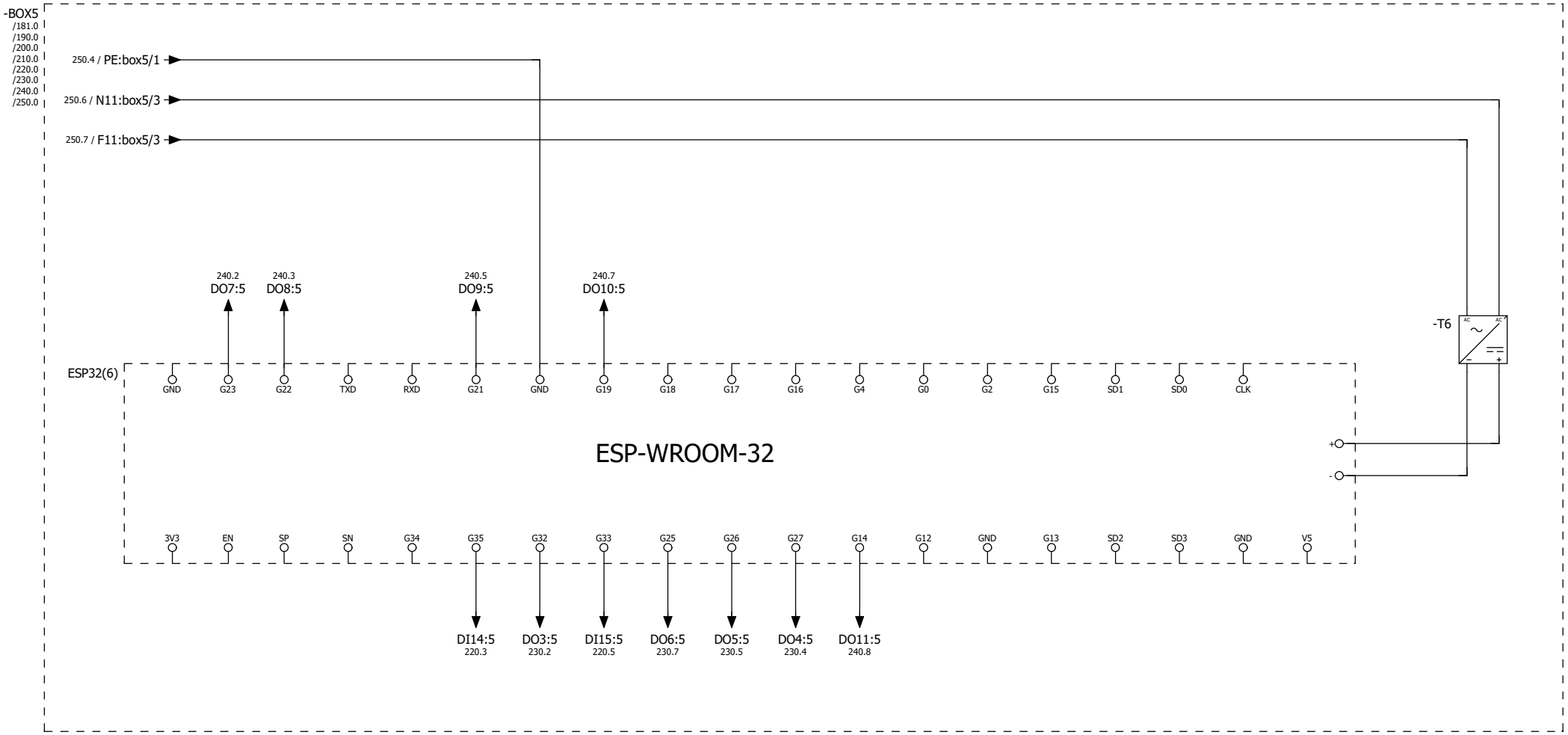
230 250

			Fecha	28/07/2020	EPLAN		EPLAN Software & Service GmbH & Co. KG		
			Resp.	SERBO					
			Probado		Instalación domótica con Raspberry Pi y ESP32 mediante MQTT			IEC_tpl003	Hoja 240
Cambio	Fecha	Nombre	Original		Sustitución por	Sustituido por			Página 27 / 36

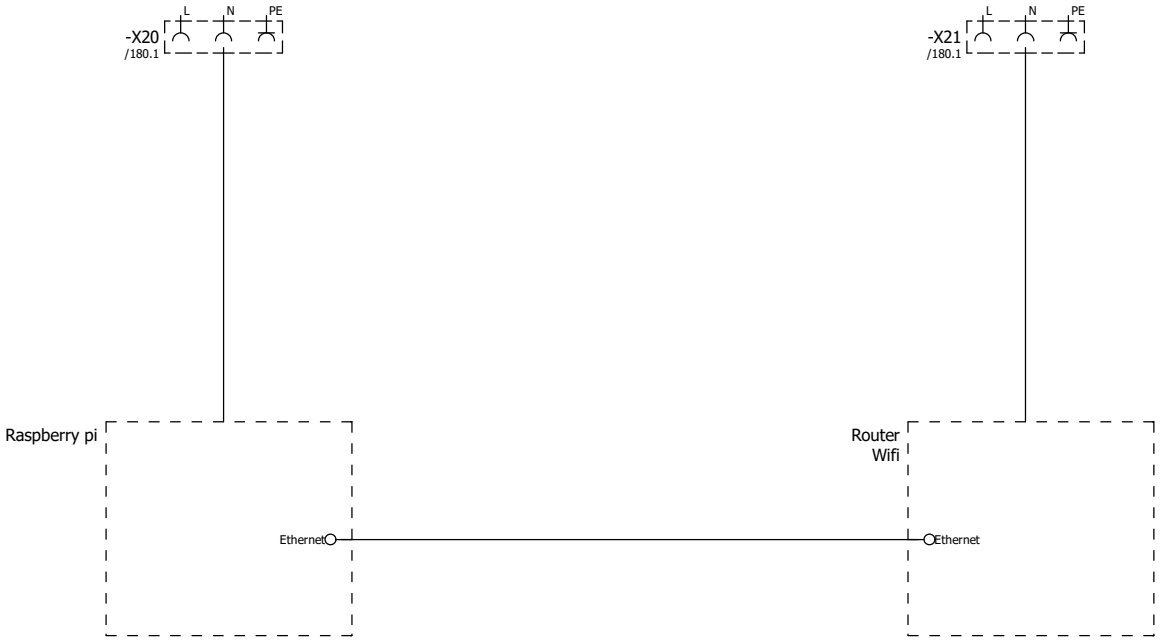


240 260

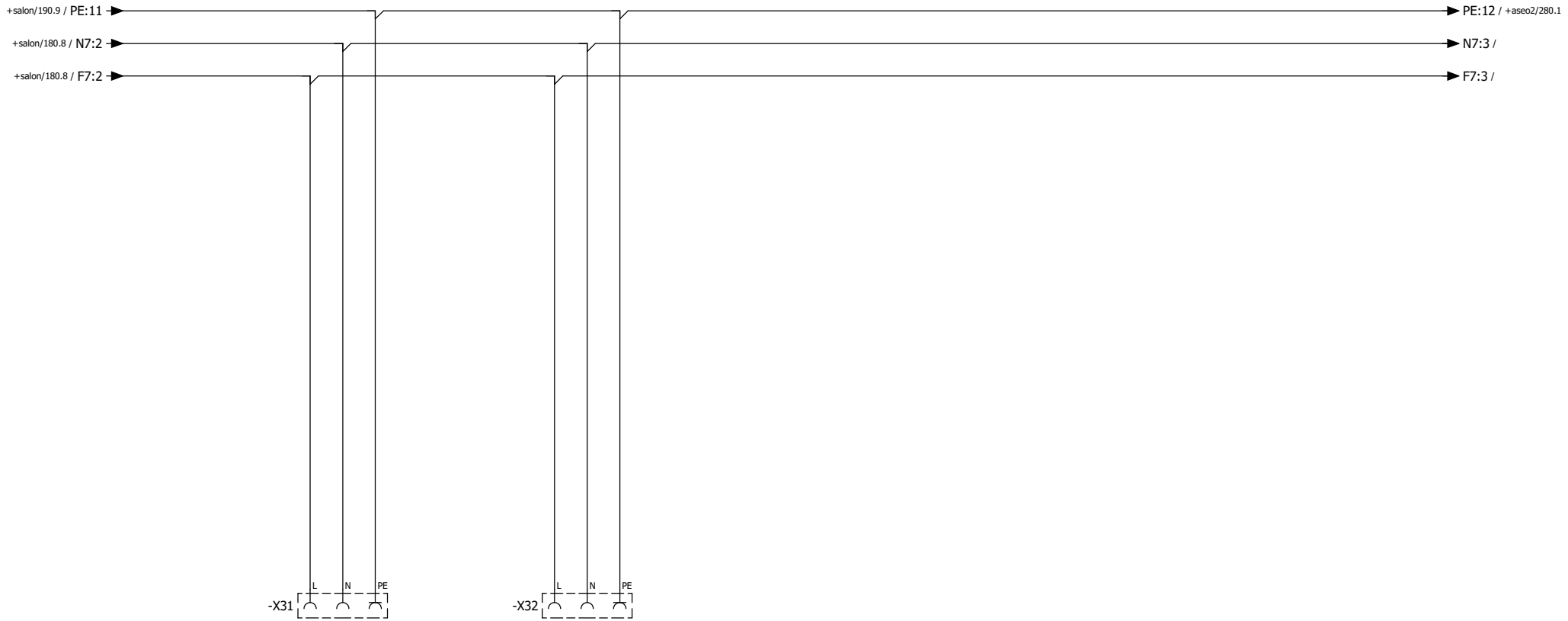
			Fecha	28/07/2020	EPLAN	EPLAN Software & Service GmbH & Co. KG		
			Resp.	SERBO				
			Probado		Instalación domótica con Raspberry Pi y ESP32 mediante MQTT		IEC_tpl003	Hoja 250
Cambio	Fecha	Nombre	Original		Sustitución por	Sustituido por		Página 28 / 36



			Fecha	28/07/2020	EPLAN	EPLAN Software & Service GmbH & Co. KG		
			Resp.	SERBO				
			Probado		Instalación domótica con Raspberry Pi y ESP32 mediante MQTT		IEC_tpl003	Hoja 260
Cambio	Fecha	Nombre	Original		Sustitución por	Sustituido por		Página 29 / 36



			Fecha	16/06/2020	EPLAN		EPLAN Software & Service GmbH & Co. KG		
			Resp.	SERBO					
			Probado		Instalación domótica con Raspberry Pi y ESP32 mediante MQTT			IEC_tpl003	Hoja 261
Cambio	Fecha	Nombre	Original		Sustitución por	Sustituido por			Página 30 / 36



+salon/261

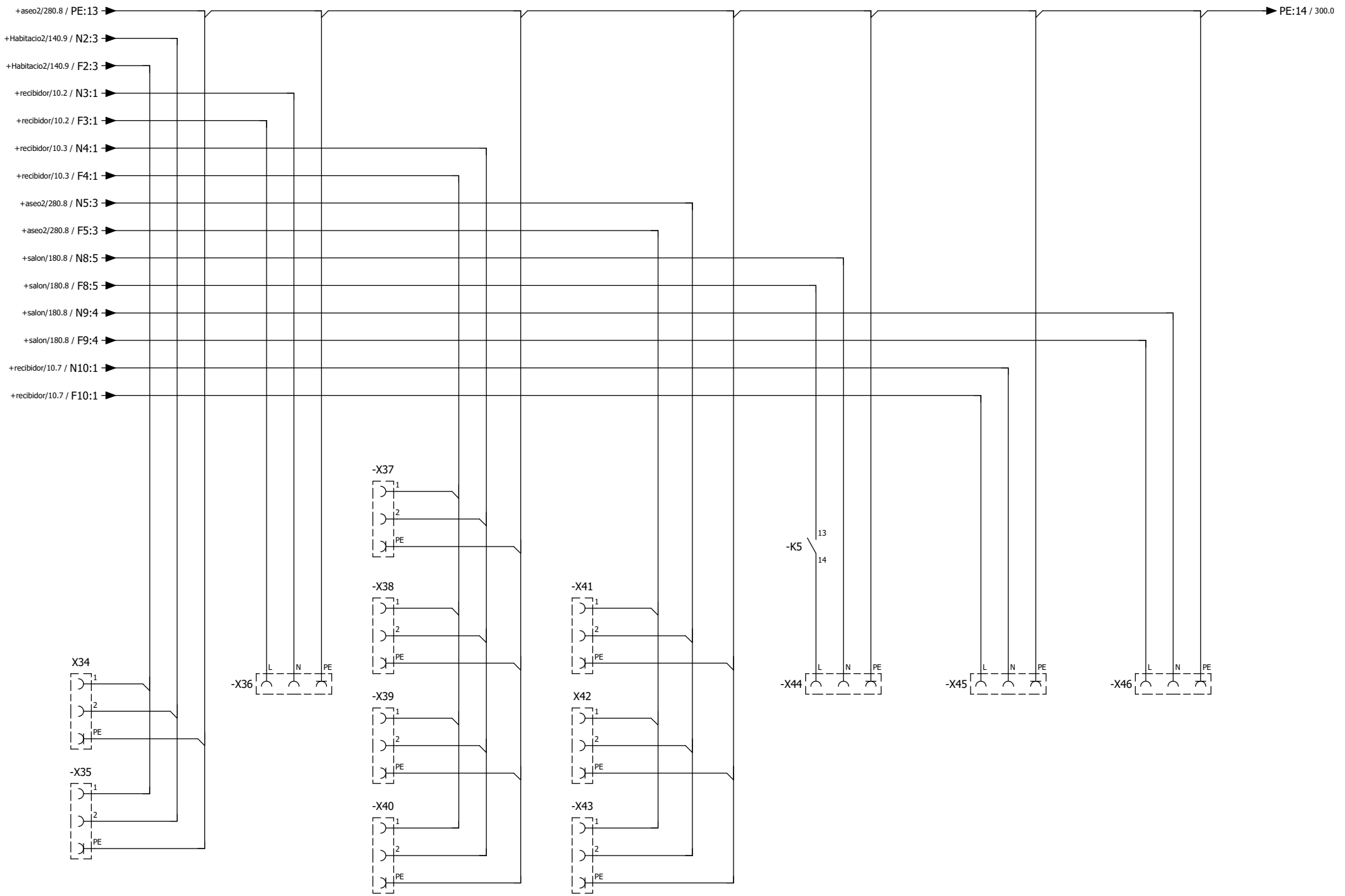
+aseo2/280

			Fecha	16/06/2020	EPLAN		EPLAN Software & Service GmbH & Co. KG		
			Resp.	SERBO					
			Probado		Instalación domótica con Raspberry Pi y ESP32 mediante MQTT			IEC_tpl003	Hoja 270
Cambio	Fecha	Nombre	Original		Sustitución por	Sustituido por			Página 31 / 36

+patio/270

+cocina/290

			Fecha	16/06/2020	EPLAN	EPLAN Software & Service GmbH & Co. KG		
			Resp.	SERBO				
			Probado		Instalación domótica con Raspberry Pi y ESP32 mediante MQTT		IEC_tpl003	Hoja 280
Cambio	Fecha	Nombre	Original		Sustitución por	Sustituido por		Página 32 / 36



+aseo2/280

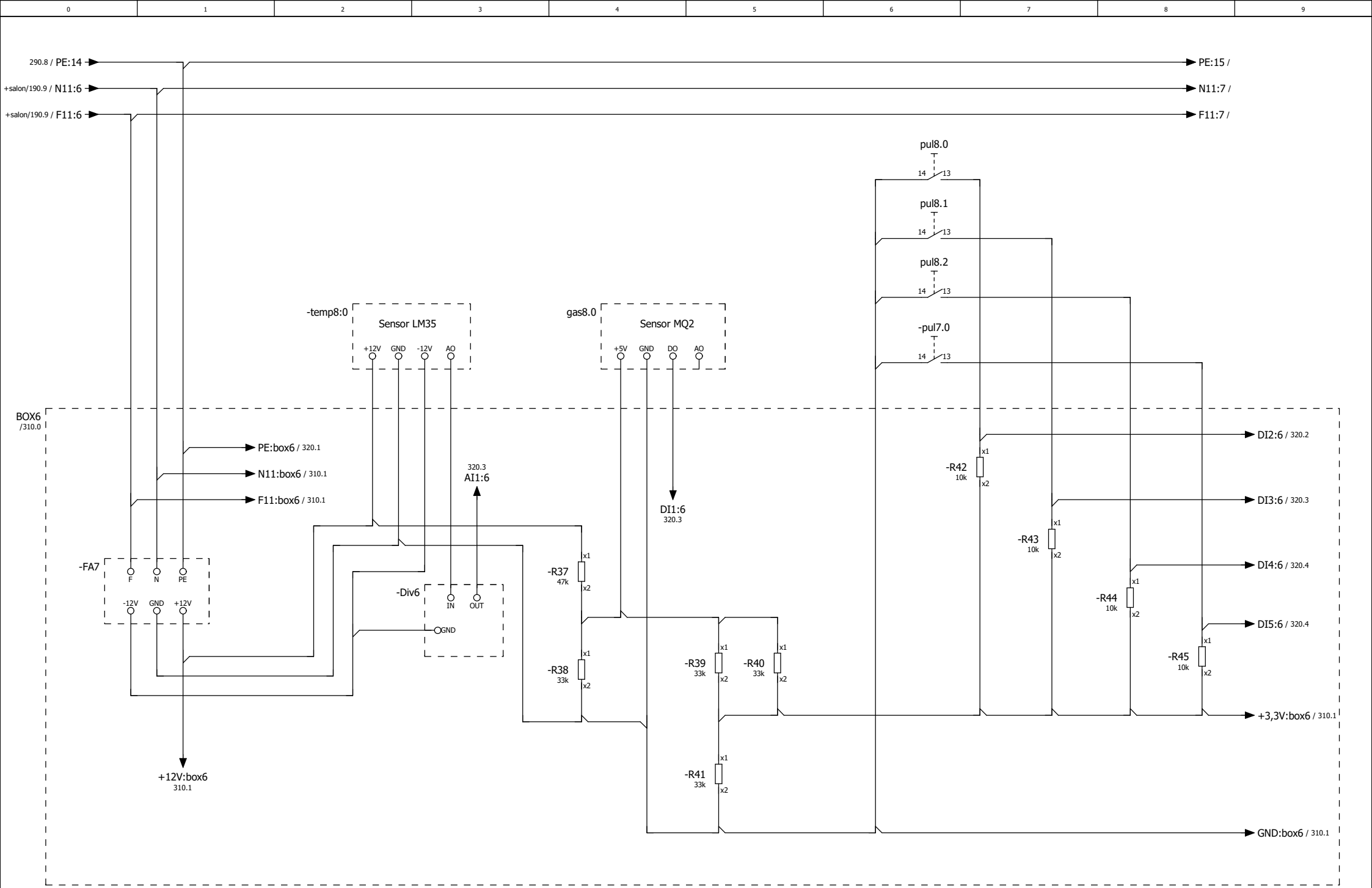
300

			Fecha	17/06/2020	EPLAN		EPLAN Software & Service GmbH & Co. KG	
			Resp.	SERBO				
			Probado		Instalación domótica con Raspberry Pi y ESP32 mediante MQTT			
Cambio	Fecha	Nombre	Original		Sustitución por	Sustituido por		

IEC_tpl003

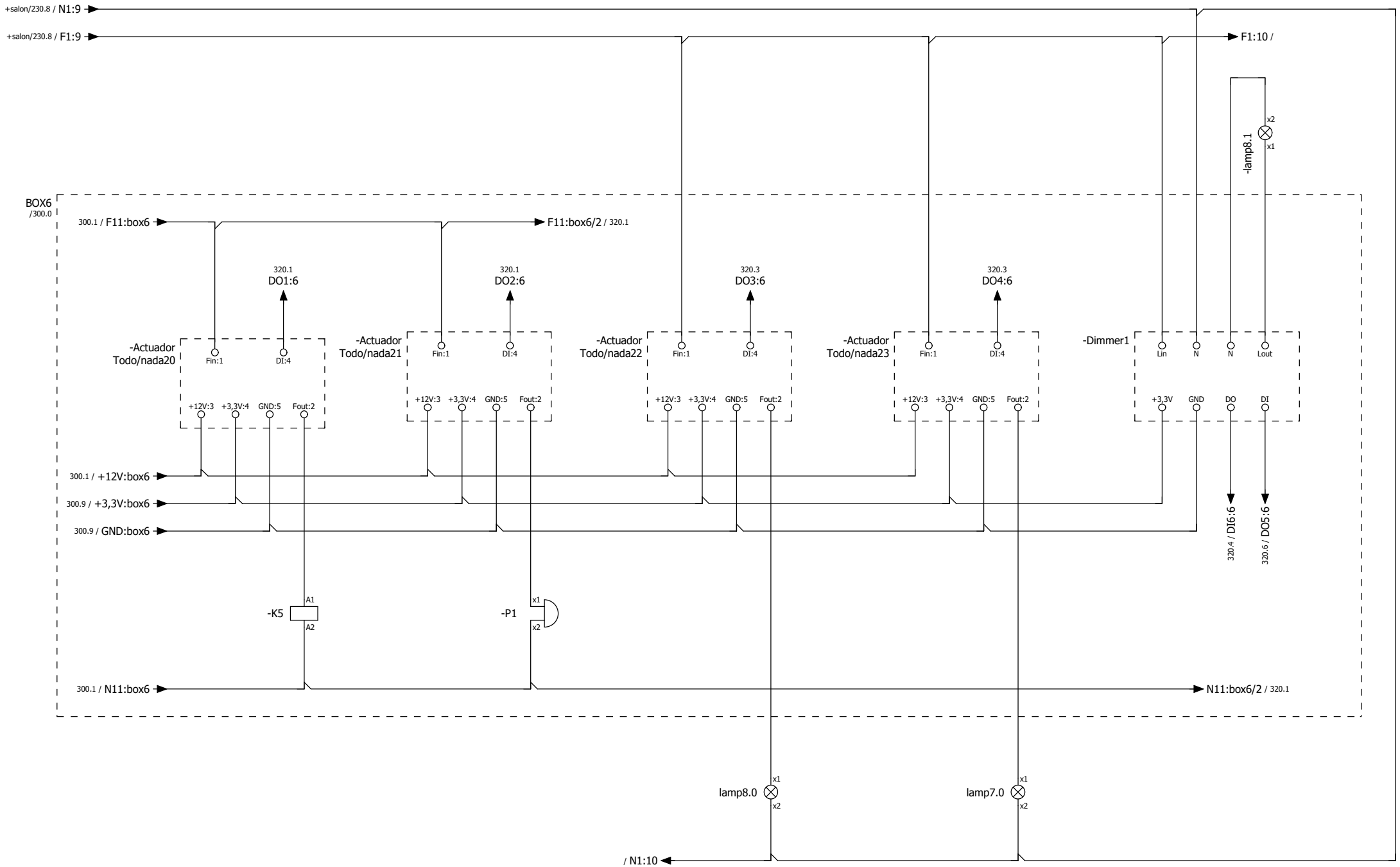
Hoja 290

Página 33 / 36



290 310

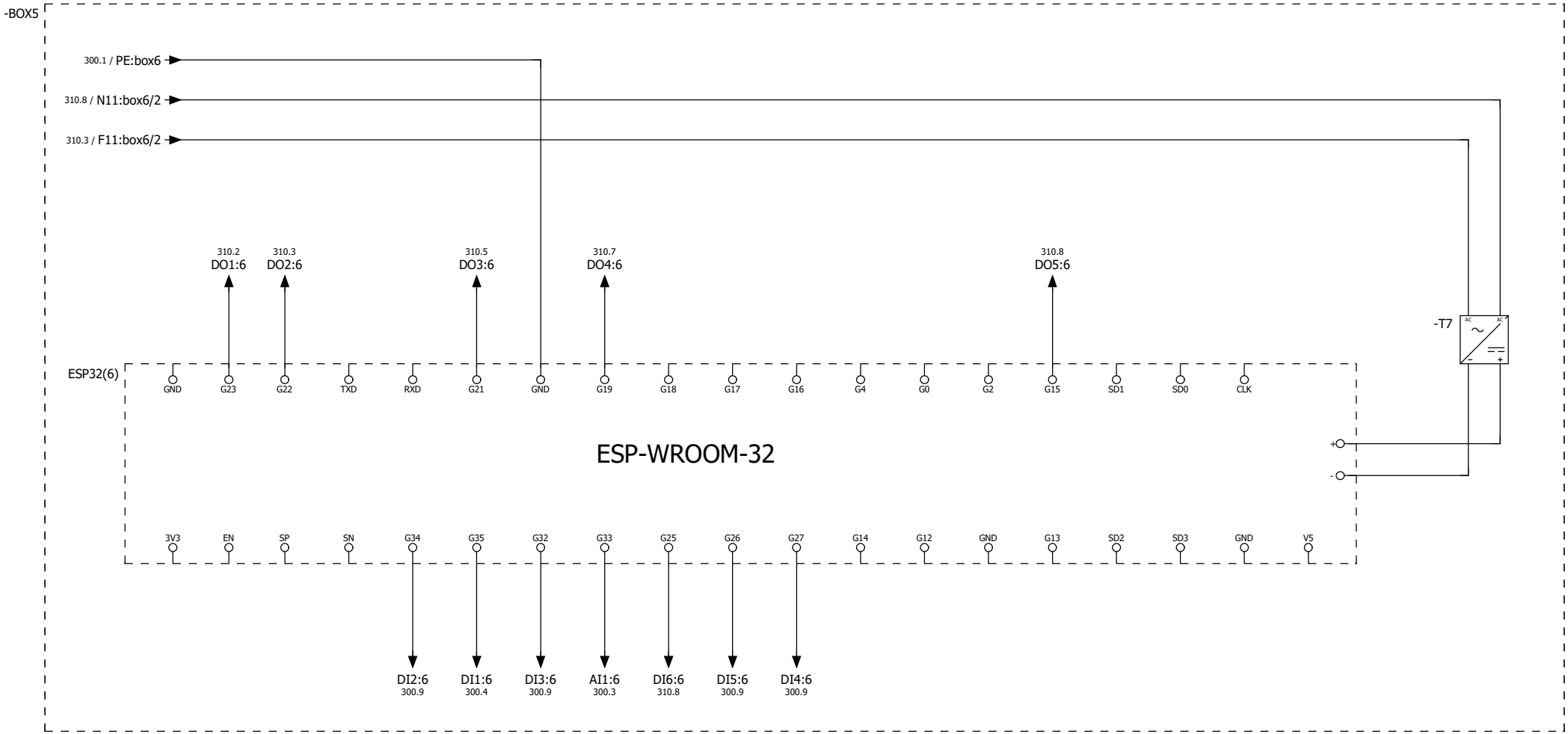
			Fecha	28/07/2020	EPLAN	EPLAN Software & Service GmbH & Co. KG		
			Resp.	SERBO				
			Probado		Instalación domótica con Raspberry Pi y ESP32 mediante MQTT		IEC_tpl003	Hoja 300
Cambio	Fecha	Nombre	Original		Sustitución por	Sustituido por		Página 34 / 36



300

320

			Fecha	17/06/2020	EPLAN	EPLAN Software & Service GmbH & Co. KG		
			Resp.	SERBO				
			Probado		Instalación domótica con Raspberry Pi y ESP32 mediante MQTT		IEC_tpl003	Hoja 310
Cambio	Fecha	Nombre	Original		Sustitución por	Sustituido por		Página 35 / 36



310

			Fecha	28/07/2020	EPLAN	EPLAN Software & Service GmbH & Co. KG		
			Resp.	SERBO				
			Probado		Instalación domótica con Raspberry Pi y ESP32 mediante MQTT		IEC_tpl003	Hoja 320
Cambio	Fecha	Nombre	Original		Sustitución por	Sustituido por		Página 36 / 36