

Memorias
LDH
Home Assistant

Jesús Carrascosa Carro
Antonio Membrive Martínez

12 de enero de 2026

Índice general

1. Introducción	2
1.1. HomeAssistant	2
1.1.1. Estructura de HomeAssistant	2
1.2. Arduino	4
1.3. MySensors	4
2. Instalación, Comprobación y Actualización del SO.	6
2.1. Instalación del Sistema Operativo en la MicroSD	7
2.2. Comprobación de la instalación y del acceso por SSH	7
2.2.1. Actualización del SO.	8
3. Instalación de MySensors y HomeAssistant	9
3.1. Instalación de Python y HomeAssistant	9
3.2. Configuración inicial de HomeAssistant	12
3.3. Instalación del Gateway de MySensors	13
3.4. Conexión entre HomeAssistant Y MySensors	14
4. Creación de los Nodos 1 y 2	16
4.1. Nodo 1	16
4.2. Nodo 2	22
4.2.1. Obtención de información de la batería	23
4.3. Construcción del Nodo 2	24
4.3.1. Incorporación de signing	27
4.3.2. Versión final del código	31
A. Creación de un sistema de alertas de glucosa lumínico	35
A.1. Idea principal	35
A.2. Materiales y herramientas necesarias	36
A.3. Construcción	37
A.3.1. Instalación y Configuración de HomeAssistant	37
A.3.2. Configuración de la Bombilla y la Integración Wiz	38
A.3.3. Instalación de HACS	40
A.3.4. Instalación y configuración del elemento personalizado	41
A.3.5. Creación de la automatización	44

Capítulo 1

Introducción

1.1. HomeAssistant

Home Assistant es una plataforma de código abierto y gratuita diseñada para la automatización de hogares, ofreciendo control local sin dependencias de servicios en la nube. Posee más de 1.400 integraciones compatibles, Home Assistant funciona completamente en redes locales, permitiendo reutilizar hardware antiguo (Raspberry Pi, PC, o portátil) como centro de domotización inteligente. Emplea una filosofía basada en los principios del software libre y abierto y su comunidad:

1. Libertad de control, respaldada y aplicada mediante el uso de licencias de software libre y abierto que aunque no poseen una cláusula *copyleft* al emplear la licencia Apache 2.0, permite la compatibilidad con licencias de *copyleft* estricto como la GPLv3.
2. Ausencia de suscripciones mensuales. Se debe matizar que la fundación que respalda el proyecto, ofrece una suscripción mensual a servicio en la nube HomeAssistant cloud que simplifica el acceso remoto a la instancia y las integraciones con diversos asistentes virtuales tales como Alexa y Google. Este servicio es plenamente opcional y las mismas funcionalidades se pueden obtener mediante herramientas y servicios propios, si bien es un procedimiento ligeramente engorroso.
3. Privacidad de datos. Principalmente, porque para la interacción con HomeAssistant no es necesaria una conexión a internet, Sin embargo, determinados dispositivos de terceros (Wiz, Xiaomi etc.), aun siendo compatibles mediante integraciones, pueden necesitar acceso a servicios en la nube para su configuración inicial y/o para algunas funciones avanzadas, lo que introduce dependencias adicionales externas al propio sistema domótico¹.
4. Comunidad activa de desarrollo. Véase el foro oficial, el portal de desarrolladores y red de creadores. También cabe mencionar la integración HACS (HomeAssistant Community Store) que permite la instalación y gestión de elementos personalizados.

1.1.1. Estructura de HomeAssistant

En cuanto a la estructura, los datos y la lógica de HomeAssistant se organizan en torno a dos capas fundamentales que actúan como capas concéntricas de una cebolla:

¹<https://www.home-assistant.io/blog/2016/02/12/classifying-the-internet-of-things/#classifiers>

Capa 1: Modelo de Datos:

La primera capa corresponde al modelo de datos, encargado de representar la información procedente del entorno y de los dispositivos conectados. Esta capa se estructura en varios niveles:

1. **Integraciones:** Conexiones con servicios externos o protocolos (por ejemplo, iluminación inteligente, dispositivos DIY mediante ESPHome, servicios meteorológicos o estándares de comunicación locales como Zigbee o Matter). Una integración puede "traer" múltiples dispositivos.
2. **Dispositivos:** Elementos físicos o lógicos proporcionados por cada integración, como bombillas, sensores, actuadores o pasarelas.
3. **Entidades:** Componentes funcionales concretos de cada dispositivo (luces, interruptores, sensores de temperatura, humedad, movimiento, binarios, cubiertas, climatización, etc.), que exponen estados y atributos.
4. Los **estados**, como su propio nombre indican el estado actual o medición de un sensor. Mientras que los **atributos** son metadatos asociados a los estados de un dispositivo que proporcionan información adicional relevante, como el brillo en el caso de una luz, la unidad de medida en un sensor de temperatura, etc. No representan por sí mismos un estado principal, pero complementan su interpretación y permiten automatizaciones precisas.

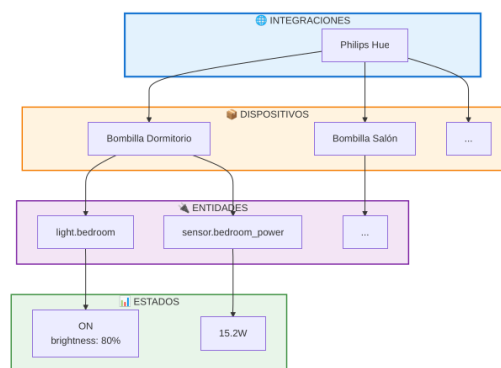


Figura 1.1: Ejemplo de la jerarquía de la capa de datos.

Capa 2: Modelo de Aplicaciones

La segunda capa agrupa las **aplicaciones** que operan sobre el modelo de datos para definir el comportamiento del sistema domótico. En esta capa se incluyen:

- **Automatizaciones, scripts y escenas**, que permiten especificar reglas de comportamiento basadas en disparadores, condiciones y acciones.
 - **Automatizaciones:** Definen reglas completas basadas en el esquema *disparador* $\rightarrow$ *condiciones* $\rightarrow$ *acciones*.
Se ejecutan automáticamente cuando ocurre un evento y permiten que el sistema reaccione sin intervención manual.

- **Scripts:** Agrupan una secuencia de acciones en una unidad reutilizable que puede ser llamada desde automatizaciones, escenas o la propia interfaz.
Resultan adecuadas para encapsular comportamientos procedimentales sin necesidad de repetir la misma lógica en varias automatizaciones.
- **Escenas:** Establecen un conjunto de estados objetivos para varias entidades, sin lógica condicional o flujo complejo.
Al activarse, aplican esos estados predefinidos para crear un ambiente concreto, y suelen combinarse con automatizaciones y scripts para integrarse en comportamientos más elaborados.
- **Interfaces de usuario:** como los paneles de control (dashboards) y las vistas interactivas, así como mecanismos de control por voz o asistentes basados en IA.
- **Herramientas de análisis y comunicación:** Entre ellas destacan el historial de estados, el registro de eventos y los sistemas de notificaciones.

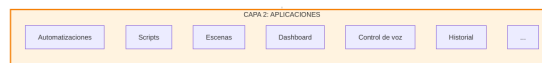


Figura 1.2: Diagrama de la capa de aplicaciones.

1.2. Arduino

Arduino representa una plataforma de hardware y software de código abierto diseñada para facilitar el desarrollo de proyectos electrónicos interactivos, particularmente en el ámbito del prototipado rápido y la educación en electrónica digital. Surgida en 2005 como iniciativa del Interaction Design Institute Ivrea en Italia, Arduino se basa en microcontroladores Atmel AVR y proporciona entornos de desarrollo accesibles que integran compilación, carga de código y depuración en un solo IDE (Entorno de Desarrollo Integrado). Sus placas, como la Arduino Uno basada en el microcontrolador ATmega328P, ofrecen múltiples pines de entrada/salida digitales y analógicos, soporte para comunicación serie, I2C, SPI y PWM, lo que permite la interfaz con sensores, actuadores y módulos de comunicación inalámbrica.

Desde una perspectiva técnica, Arduino opera bajo un modelo de programación basado en el lenguaje Wiring, una simplificación de C/C++ que abstrae complejidades de bajo nivel como la gestión de registros y temporizadores, permitiendo a desarrolladores enfocarse en la lógica de aplicación. Funcionalmente, soporta bibliotecas estandarizadas para periféricos comunes —tales como DHT para sensores de temperatura y humedad, o Servo para control de motores— y un ecosistema de shields expansibles que amplían capacidades como conectividad Ethernet o Bluetooth. Esta arquitectura modular promueve la interoperabilidad y escalabilidad, posicionando a Arduino como herramienta fundamental en aplicaciones de Internet de las Cosas (IoT), robótica y automatización doméstica, donde la integración con protocolos como MQTT facilita el intercambio de datos en redes distribuidas.

1.3. MySensors

MySensors constituye un framework de código abierto orientado a la creación de redes de sensores inalámbricas de bajo costo y bajo consumo energético, optimizado para integración en

sistemas de domótica y monitoreo ambiental. Desarrollado originalmente por Henrik Ekblad, MySensors proporciona una biblioteca que facilita la implementación de redes inalámbricas basadas en plataformas como Arduino, Raspberry Pi y microcontroladores compatibles. El framework emplea principalmente el transceptor nRF24L01+ operando en la banda de frecuencia de 2.4 GHz (específicamente entre 2400 y 2525 MHz distribuidos en hasta 126 canales), aunque también soporta módulos RFM69, WiFi y RRM95(LoRa) como alternativa flexible según requisitos regionales y de construcción.

Técnicamente, MySensors implementa una topología de red mesh autorreparable donde cada nodo puede actuar como repetidor para otros, extendiendo el alcance más allá de la línea de visión directa mediante nodos relé intermedios. La red puede contener hasta 254 nodos independientes, donde uno actúa como gateway hacia controladores domóticos centrales (Home Assistant, Domoticz, o MQTT brokers) y los restantes funcionan como sensores, actuadores o repetidores. Funcionalmente, MySensors define un protocolo de mensajería binaria compacta con tipos estandarizados que abstraen la complejidad de la pila de red, permitiendo a desarrolladores enfocarse en la lógica de aplicación. Soporta un amplio espectro de sensores (temperatura, humedad, movimiento, luminosidad) y actuadores (relés, dimmers), con mecanismos de bajo consumo energético (ciclos sleep/wake) que minimizan el gasto a niveles de microamperios, viabilizando despliegues alimentados por baterías AA durante períodos superiores a un año. Así, MySensors se posiciona como solución escalable para redes IoT residenciales, priorizando fiabilidad, eficiencia energética y compatibilidad con ecosistemas heterogéneos de domótica.

Capítulo 2

Instalación, Comprobación y Actualización del SO.

El objetivo principal de esta sesión de prácticas es instalar y configurar la Raspberry Pi 4, con el sistema operativo Raspberry Pi OS Lite, de tal forma que en las sucesivas prácticas, podamos acceder a ella utilizando SSH y mDNS. Para ello, hemos utilizado las siguientes herramientas/elementos hardware y software:

- **RPi-Imager:** La herramienta de *flasheo* oficial de RaspberryPi¹. Con su interfaz simple nos permite escoger entre una amplia variedad de sistemas operativos(desde variaciones de "propósito general" de Raspbian (y otras distros como ubuntu) hasta derivaciones/variaciones de sistemas operativos ² de "propósito más específico" como libreELec, retroPie o Home Assistant. para toda la familia de SoC de RaspberryPi. Además de permitirnos hacer configuraciones previas a la instalación de usuarios, redes y ssh entre otras características.
- **Raspberry Pi 4 Modelo B:** Un *SBC* (computador monoplaca) que cuenta con **2 GB de RAM DDR4**, un **procesador Cortex A72 de 64 bits**, y conectividad inalámbrica que incluye **WiFi (802.11b/g/n/ac)** y **Bluetooth 5.0**. Además, dispone de **4 puertos USB A** y **2 puertos mini HDMI**. Este microcomputador será el que utilizaremos a lo largo de la asignatura.
- **Raspberry Pi OS Lite:** Anteriormente conocido como Raspbian. Tal y como indican el nombre y su complemento, se trata de la versión liviana (sin entorno de escritorio y con el número de paquetes pre-instalados reducido) del sistema operativo oficial de Raspberry Pi. Está basado en Debian(en nuestro caso Debian 12 *Bookworm*). Fuente³
- **SSH (Secure Shell):** Se denomina así tanto al programa como al protocolo que permite el acceso a un escritorio remoto de forma cifrada. Además permite el intercambio de ficheros,y la gestión de claves RSA para no depender de contraseñas en la conexión. Fuente⁴.

Para conectarse a la Raspberry Pi, se usará la aplicación FLOSS ConnectBot disponible en F-Droid.

¹Es el sucesor espiritual de NOOBS, también de RaspberryPi

²Realmente, el término Sistema Operativo no es el más adecuado, ya que se aproximan más a ser versiones prácticamente *ready-to-go* (incluyendo la propia instalación del SO base) de programas de servidor.

³https://es.wikipedia.org/wiki/Raspberry_Pi_OS

⁴https://es.wikipedia.org/wiki/Secure_Shell

- **mDNS (multicast DNS):** Permite sin necesidad de utilizar un servidor dns convencional pero empleando el modo de funcionamiento, formato e interfaz de DNS, establecer una relación entre el nombre del host (más un `.local`) y su ip local. Es decir nos permitirá conectarnos a la Raspberry sabiendo solamente el usuario, nombre del host y puerto ssh que se haya previamente establecido en la configuración del SO. Por lo general y como ha sucedido a lo largo de estas prácticas, tiene una tasa de fallos bastante baja pero suficientemente significativa como para ser una fuente común de errores.

2.1. Instalación del Sistema Operativo en la MicroSD

Usando la versión de RPi-imager que se encuentra instalada en el ordenador del laboratorio⁵, se ha *flasheado* la tarjeta microsd con la siguiente configuración:

- **General:**
 - **Dispositivo:** Raspberry Pi 4.
 - **SO:** Raspberry Pi OS Lite x64.
 - **Hostname:** raspberrypiLDH (`.local`) .
 - **Usuario y Contraseñas:** practicas practicas .
 - **LAN Inalámbrica:** La correspondiente a uno de nuestros dispositivos móviles. Se ha especificado como país España (ES).
 - **Zona Horaria:** Europa/Madrid. No se ha configurado como servidor horario el *rediris*, de ser necesario se configurará luego⁶.
- **Idioma del teclado:** Español, 105 teclas.
- **Servicios:** SSH activado con autenticación mediante contraseña.
- **Opciones:** Telemetría desactivada.

Tras esto se ha pulsado siguiente y ha comenzado el proceso de instalación y verificación del SO en la tarjeta microSD.

2.2. Comprobación de la instalación y del acceso por SSH

En esta fase de la práctica, se ha instalado la microSD en la Raspberry Pi 4 y se ha conectado a la corriente. Después de esperar a que se estableciera la conexión con la red inalámbrica, se intentó conectarse mediante SSH utilizando la resolución por mDNS (`practicas@raspberrypiLDH.local:22`). Sin embargo, esta conexión falló, por lo que se procedió a averiguar la IP de la Raspberry Pi (10.107.223.138) utilizando la aplicación Vernet⁷, y nos conectamos directamente a través de la dirección IP.

Tras comprobar que la conexión fuese exitosa, se procedió al siguiente paso.

⁵Por indicaciones del profesor no se actualizó a la siguiente versión

⁶Hay que tener en cuenta que exceptuando la RPi5 a la cual se le puede comprar aparte, las RPi no tienen opción a batería para reloj interno, por lo que al desenchufar se pierde el valor de la hora.

⁷También FLOSS y disponible en F-Droid.

2.2.1. Actualización del SO.

Se ejecutaron los siguientes comandos en el interprete de comandos:

```
sudo apt update
sudo apt dist-upgrade
```

La estructura y significado de los comandos es la siguiente:

1. **Sudo:** (mnemotécnico: *Super User DO*) Permite a un usuario autorizado ejecutar un comando como superusuario o como otro usuario, según lo especificado por la política de seguridad. Fuente: MANPAGES.
2. **apt:** Proporciona una interfaz de línea de comandos de alto nivel para el sistema de gestión de paquetes. Comparado con **apt-get** proporciona mayor interactividad al usuario. Fuente: MANPAGES.
3.
 - **update:** Actualiza la información de paquetes de todas las fuentes configuradas. Fuente MANPAGES.
 - **dist-update:** Actualiza los paquetes con resolución inteligente de conflictos, llegando a desinstalar paquetes si fuera necesario (ad diferencia de upgrade) por si solo. Fuente⁸

Estos comandos se pueden combinar de la siguiente forma:

```
sudo apt update && sudo apt dist-upgrade
```

Tras actualizar todos los paquetes, se apagó la RPi remotamente con el auto-explicativo comando:

```
sudo shutdown -h now
```

Como curiosidad, mientras que en una consola normal (no remota) no requiere autenticación mediante sudo, en una consola remota, si lo requiere para evitar apagados accidentales.

⁸<https://askubuntu.com/questions/81585/what-is-dist-upgrade-and-why-does-it-upgrade-more-than-upgrade>

Capítulo 3

Instalación de MySensors y HomeAssistant

3.1. Instalación de Python y HomeAssistant

En este apartado se detallará el proceso para instalar y configurar Home Assistant, el software que utilizaremos para los apartados posteriores, empleando principalmente:

- **APT:** Descrito en la sección 2.
- **Pip:** Gestor de paquetes de Python, utilizado para instalar las dependencias y librerías necesarias del entorno virtual.
- **Git:** Sistema de control de versiones, empleado para clonar repositorios y descargar componentes personalizados o actualizaciones.
- **Wget:** Herramienta de línea de comandos para descargar archivos de la red (de ahí su nombre *Web Get*), necesaria para obtener scripts de instalación y claves de seguridad.

El procedimiento empleado ha sido el siguiente:

1. Descargar el código fuente de la versión requerida de python de la pagina oficial de Python utilizando wget (aunque se podría utilizar también cURL)

```
wget https://www.python.org/ftp/python/3.13.2/Python-3.13.2.tar.xz
tar -xf Python-3.13.2.tar.xz #extraemos
```

2. Instalar todas las dependencias recomendadas, evitando la instalación recursiva de dependencias recomendadas¹ mediante la opción *--no-install-recommends*.

```
sudo apt -y install --no-install-recommends \
build-essential \
gdb \
```

¹Es decir, no instala las dependencias recomendadas de las dependencias recomendadas, peros si las dependencias obligatorias.

```

lcov \
libbz2-dev \
libffi-dev \
libgdbm-dev \
liblzma-dev \
libncurses5-dev \
libreadline-dev \
libsqlite3-dev \
default-libmysqlclient-dev \
libssl-dev \
lzma \
lzma-dev \
tk-dev \
uuid-dev \
xvfb \
zlib1g-dev \
llvm \
xz-utils\
python3-openssl\
git

```

3. Configuramos y compilamos Python dentro de nuestro directorio(para evitar tener que emplear permisos de superusuario) *\$HOME/usuario*²:

```

// Situamos el shell

cd Python-3.13.2

// Creamos y/o Configuramos los archivos de configuración del Makefile
// indicando que active las optimizaciones de compilación y que se
//instale en el directorio del usuario

./configure --prefix=$HOME/python3.13.2-bin --enable-optimizations

// Compilamos el código fuente utilizando 4 hilos de procesamiento,
// generando así los binarios y bibliotecas necesarias para la instalación.

make -j4

// Copia los binarios y archivos resultantes a las ubicaciones
//definidas durante la configuración

make install

```

²Nótese en los siguientes comandos, que *\$HOME* actúa de alias de la dirección del usuario actual en home (e.g. si mi usuario es yisus, el comando `echo $HOME` devuelve `/home/yisus` por lo que convierte rutas –que de otro modo serían relativas– en absolutas).

4. Creamos y activamos el entorno virtual para la instalación de paquetes y dependencias con pip y así evitar conflicto con los paquetes instalados globalmente u otros entornos:

```
//Nos ubicamos en nuestro directorio de usuario

cd $HOME

// creamos el entorno virtual denominado hass-venv

$HOME/python3.13.2-bin/bin/python3 -m venv hass-venv

//Activamos el entorno
cd hass-venv
source bin/activate
```

5. Instalamos dependencias adicionales de HomeAssistant mediante pip y apt, previa actualización de pip:

```
// Actualizamos pip

pip install pip --upgrade

// Instalamos otras dependencias de bases de datos(mysql) y
// compresion(zlib e isal)

pip install zlib_ng isal mysqlclient

// Instalamos otras dependencias de multimedia (torbojpeg y ffmpeg) y
//redes (tcpdump ylibcap)

sudo apt install --no-install-recommends \
libturbojpeg-dev ffmpeg tcpdump libpcap0.8-dev
```

6. Instalamos y arrancamos HomeAssistant en el entorno virtual

```
// Descargamos e instalamos HomeAssistant mediante pip

pip install homeassistant

// Arrancamos HomeAssistant (a modo de prueba)

hass
```

Este proceso, pese a que tomó alrededor de una hora y cuarto en total debido a que por error humano se compiló el código dos veces seguidas, no presentó incidencias, salvo la siguiente advertencia sobre el directorio de instalación que saltó durante la compilación y configuración de Python.

```
PRACTICAS@10.107.223 .138 PRACTICAS@RASPBERRYPI4H .LOCAL
rm -f /home/practicas/python3.13.2-bin/bin/python3-intel64; \
(cd /home/practicas/python3.13.2-bin/bin; ln -s python3.13-intel64 python3-intel64) \
;
rm -f /home/practicas/python3.13.2-bin/share/man/man1/python3.1
cd /home/practicas/python3.13.2-bin/share/man/man1; ln -s python3.13.1 python3.1)
if test "xupgrade" != "xno" ; then \
case upgrade in \
upgrade) ensurepip="--upgrade" ;; \
install*) ensurepip="" ;; \
esac; \
./python -E -m ensurepip \
  --ensurepip --root=/ ; \
;
Looking in links: /tmp/tmpkhhk4_ys
Processing /tmp/tmpkhhk4_ys/pip-24.3.1-py3-none-any.whl
Installing collected packages: pip
WARNING: The scripts pip3 and pip3.13 are installed in '/home/practicas/python3.13.2-bin/bin' which is not on PATH.
Consider adding this directory to PATH or, if you prefer, suppress this warning, use --no-warn-script-location.
Successfully installed pip-24.3.1
practicas@raspberrypi4h:~/Python-3.13.2 $
```

Figura 3.1: Advertencia sobre el directorio de instalación

Esta advertencia surge con frecuencia en instalaciones personalizadas de Python como la nuestra y tras comprobar que no interfería y verificar que la instalación de Python funcionaba correctamente se procedió a ignorarla.

3.2. Configuración inicial de HomeAssistant

Tras lanzar HomeAssistant y acceder mediante un navegador externo a la URL <http://raspberrypiLDH.local:8123>³, aparece la página de bienvenida de HomeAssistant.

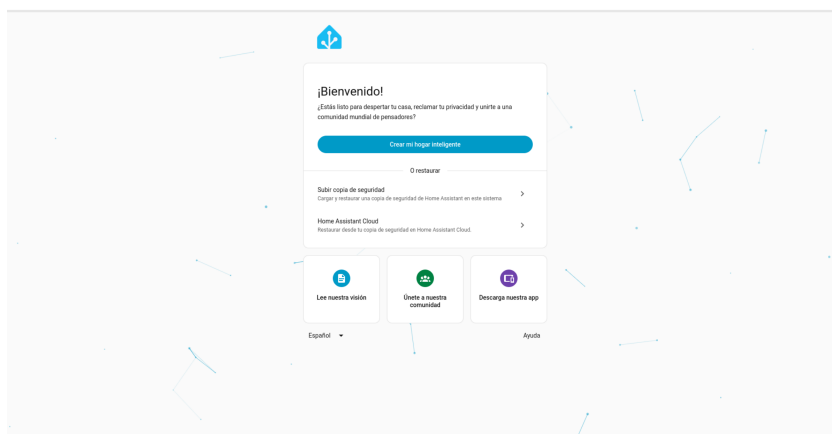


Figura 3.2: Página de bienvenida de HomeAssistant

Se procedió a configurarlo de la siguiente manera:

- **Nombre, usuario y contraseña.** Se puso practicas en todos los campos.
- **Ubicación** Se seleccionó el Instituto de Idiomas por mera comodidad, ya que la ETSII no figuraba como tal en el desplegable⁴.
- **Telemetría** No se ha activado ninguna de las opciones.

³Se matiza el uso de HTTP vs. HTTPS debido a que el cifrado suele dar problemas de acceso, por lo que fue necesario desactivar manualmente la configuración de *Always-on HTTPS* de Firefox.

⁴En instalaciones posteriores sí que se ha conseguido encontrar la ETSII en el desplegable.

Durante la realización de esta sección de la práctica no ha habido incidencias graves, exceptuando fallos menores con la resolución de dominios mediante mDNS por lo que en algunos de los pasos se ha tenido que recurrir a acceder directamente mediante la IP.

3.3. Instalación del Gateway de MySensors

Antes de comenzar con la compilación e instalación parcial de MySensors, se coloca el adaptador que nos permitirá utilizar el protocolo de radiofrecuencia nrf24 de la siguiente manera:

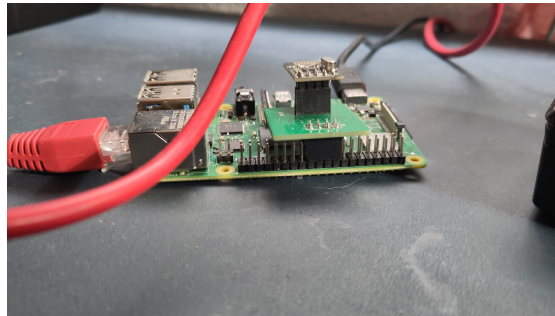


Figura 3.3: RPI4 con adaptador nrf24

Tal y como se puede apreciar en la imagen, el adaptador se coloca en el centro de los puertos GPIO de la Raspberry Pi, dejando ocho pines a cada lado.

Tras esto, en una terminal remota aparte, y con HomeAssistant desconectado, procedemos a la descarga, compilación e instalación parcial⁵ de la versión de desarrollo de MySensors

```
// Descargamos con git la versión de desarrollo de MySensors

git clone https://github.com/MySensors/MySensors.git
--branch development

// Entramos en la carpeta

cd MySensors
```

Y procedemos a configurarlo con las siguientes banderas:

- Conexión vía ETHERNET y puerto 5003:

```
--my-gateway=ethernet --my-port=5003
```

- Seleccionamos como medio de transporte el NRF y el canal que el profesor nos ha asignado (en nuestro caso el canal 15):

⁵Esto se debe a que en la última práctica con, se tendrá que volver a reconfigurar para que emplee signin.

```
--my-transport=rf24
--my-rf24-channel=15
```

- Indicamos la arquitectura y modelo de SoC:

```
--soc=BCM2711
--cpu-flags="-march=armv8-a+crc -mtune=cortex-a72"
```

Por lo que ejecutamos el siguiente comando:

```
./configure --my-gateway=ethernet --my-port=5003
--my-transport=rf24 --my-rf24-channel=15
--cpu-flags="-march=armv8-a+crc -mtune=cortex-a72"
--soc=BCM2711
```

Posteriormente arrancamos comprobamos que el funcionamiento del gateway sea el correcto con el comando:

```
sudo ./bin/mysgw
```

El funcionamiento de este programa es sencillo, simplemente redirige los paquetes recibidos via rf24 y los reenvía a HomeAssistant mediante localhost en el puerto 5003 (y viceversa).

Esta parte de la práctica se ha desarrollado sin incidencias.

3.4. Conexión entre HomeAssistant Y MySensors

Tras reiniciar la Raspberry Pi, se ha procedido a conectarse a ella mediante dos terminales remotos. En el primero se han arrancado MySensors mediante los siguientes comandos:

```
cd MySensors
sudo ./bin/mysgw
```

Mientras que en segundo terminal se lanza HomeAssistant:

```
cd hass-venv
source bin/activate
hass
```

Estos pasos se tendrán que repetir cada vez que se encienda la Raspberry Pi tanto para HomeAssistant como para MySensors.

A continuación se inicia sesión en HA y en la pestaña *Configuración* → *Dispositivos* → *Añadir Integración* → *MySensors* y configuramos una puerta de enlace dejando los parámetros por defecto⁶ y empleando el puerto 5003. Resultando en lo siguiente:

⁶En retrospectiva, habría sido buena idea aumentar la versión a al 2.32 o superior, ya que nos habría ahorrado el problema que tuvimos con el nodo 2.

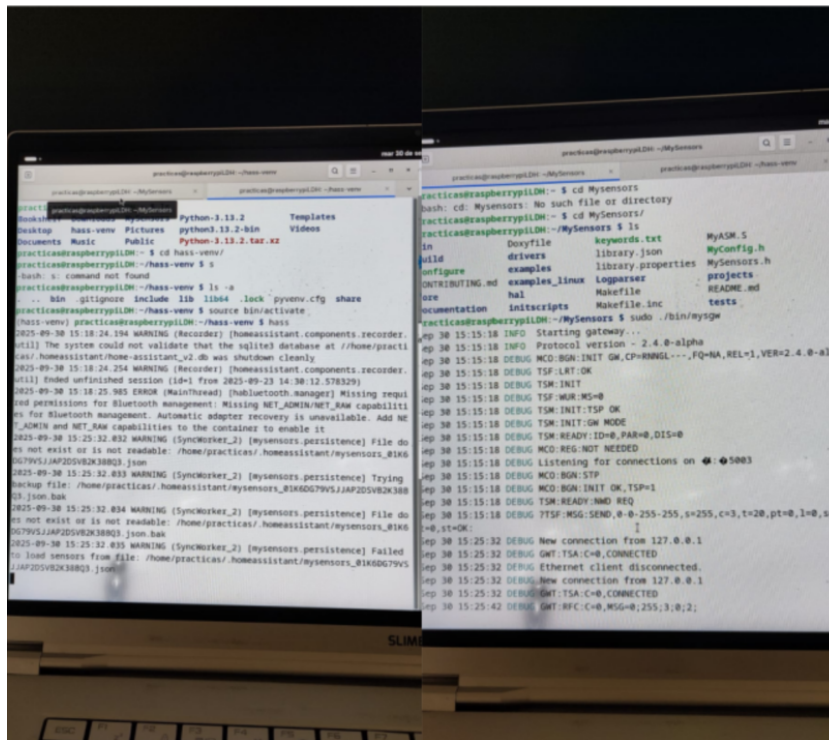


Figura 3.4: Terminal izqdo. ejecutando HomeAssistant y terminal dcho. ejecutando el gateway de MySensors.

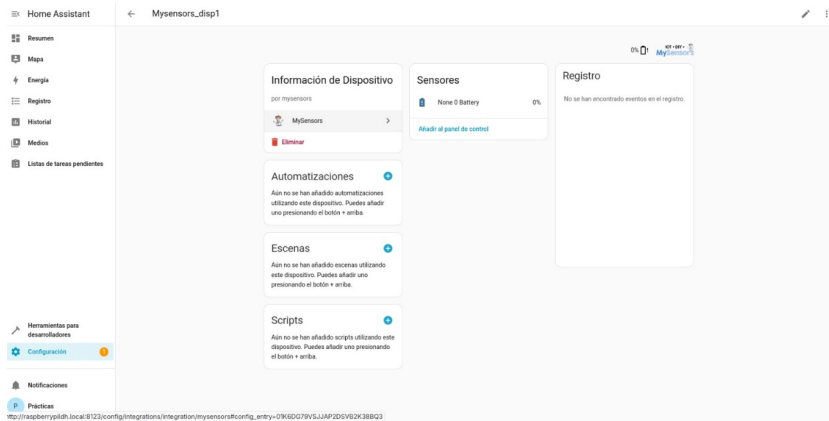


Figura 3.5: Dashboard mostrando el dispositivo MySensors.

Capítulo 4

Creación de los Nodos 1 y 2

En este capítulo se detallará la construcción de dos nodos MySensors que actuarán como dispositivos dentro de nuestra red inalámbrica. El Nodo 1 funcionará como un actuador(relé) controlado por un conmutador e integrado con HomeAssistant, mientras que el Nodo 2 será un sensor portátil alimentado por batería que transmitirá datos de temperatura y humedad a través de la red.

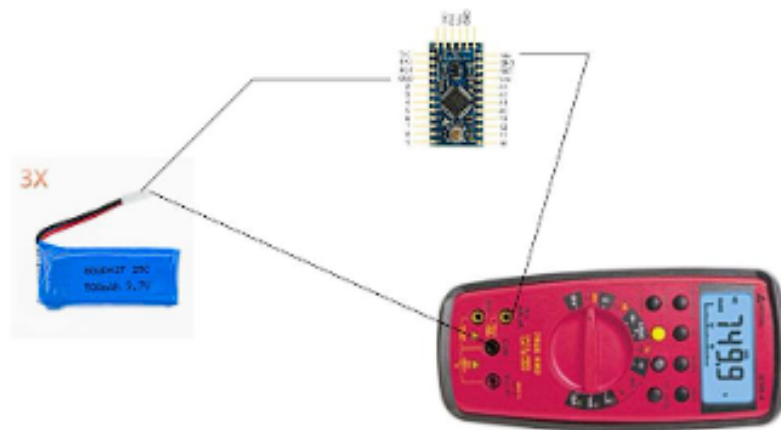


Figura 4.1: Esquemático de la arquitectura IoT a implementar. Fuente: Diapositivas de clase

4.1. Nodo 1

La implementación del Nodo 1 se centra en la construcción de un dispositivo principalmente actuador basado en Arduino Nano con módulo NRF24L01, capaz de gestionar tanto entradas manuales mediante interruptores como comandos remotos desde HomeAssistant.

Implementa las siguientes funcionalidades:

- Cuando uno de los dos interruptores que componen el conmutador cambie de estado, el relé también cambia de estado y se informa a HomeAssistant del nuevo estado actual.
- Desde HomeAssistant se ha de poder cambiar de forma remota el estado del relé y este envíe de nuevo la información sobre el estado actual del relé.

Para la construcción de este nodo se utilizaron los siguientes materiales:

- Arduino Nano: Microcontrolador con módulo NRF24 integrado, encargado de gestionar la lógica del nodo y la comunicación inalámbrica
- Conmutador: Compuesto por 2 interruptores para control manual del sistema
- Relé: Actuador encargado de controlar el encendido y apagado de un LED o dispositivo eléctrico
- Cableado: Conectores hembra-hembra y cables de conexión para realizar las interconexiones necesarias

Interconectándolos de la siguiente manera:

- Una patilla del conmutador al pin digital 3 del arduino y la otra a tierra.
- El pin de señal al pin digital 4 del arduino y el resto de pines del relé(VCC y GND) a sus respectivos homólogos en el arduino nano.

Resultando en el siguiente montaje:

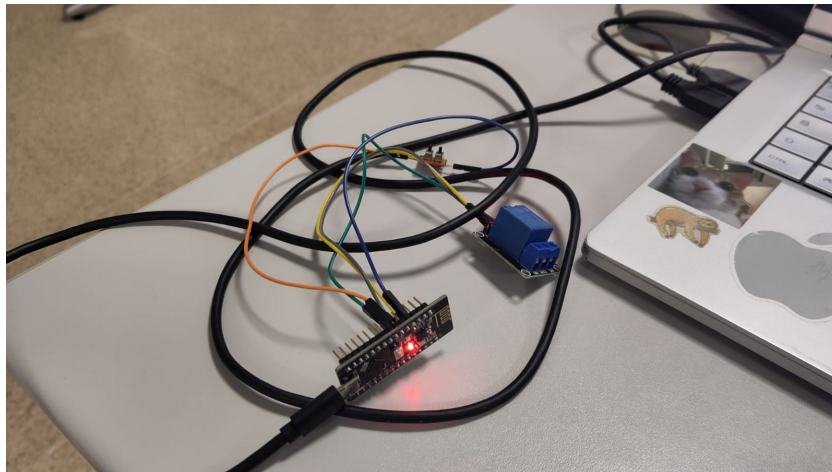


Figura 4.2: Montaje del Nodo 1

En lo que a software respecta se ha elaborado el siguiente programa empleando como base el programa de ejemplo proporcionado por el profesor:

```
/*
 * Documentation: https://www.mysensors.org
 * Support Forum: https://forum.mysensors.org
 *
 * https://www.mysensors.org/build/relay
 */

#define MY_DEBUG
#define MY_RADIO_NRF24
```

```

#define MY_REPEATER_FEATURE
#define MY_NODE_ID 22
#define MY_RF24_CE_PIN 10
#define MY_RF24_CS_PIN 9
#define MY_RF24_CHANNEL 15
#include <SPI.h>
#include <MySensors.h>
#include <Bounce2.h>

#define RELAY_PIN 4
#define CONMUTTER_PIN 3
#define CHILD_ID 45
#define RELAY_ON 1
#define RELAY_OFF 0

Bounce debouncer = Bounce();
bool state = false;
bool initialValueSent = false;

MyMessage msg(CHILD_ID, V_STATUS);
bool prev_state = true;
void setup()
{
    pinMode(CONMUTTER_PIN, INPUT_PULLUP);
    debouncer.attach(CONMUTTER_PIN);
    debouncer.interval(10);

    // Make sure relays are off when starting up
    digitalWrite(RELAY_PIN, RELAY_OFF);
    pinMode(RELAY_PIN, OUTPUT);
}

void presentation() {
    sendSketchInfo("Relay+button", "1.0");
    present(CHILD_ID, S_BINARY);
}

void loop()
{
    if (!initialValueSent) {
        Serial.println("Sending initial value");
        send(msg.set(state?RELAY_ON:RELAY_OFF));
        Serial.println("Requesting initial value from controller");
        request(CHILD_ID, V_STATUS);
        wait(2000, C_SET, V_STATUS);
    }
    if (debouncer.update()) {
        if (debouncer.read() != prev_state) {
            state = !state;

```

```

        prev_state = !prev_state;
        // Send new state and request ack back
        send(msg.set(state?RELAY_ON:RELAY_OFF), true);
    }
}

void receive(const MyMessage &message) {
    if (message.isAck()) {
        Serial.println("This is an ack from gateway");
    }

    if (message.type == V_STATUS) {
        if (!initialValueSent) {
            Serial.println("Receiving initial value from controller");
            initialValueSent = true;
        }
        // Change relay state
        state = (bool)message.getInt();
        digitalWrite(RELAY_PIN, state?RELAY_ON:RELAY_OFF);
        send(msg.set(state?RELAY_ON:RELAY_OFF));
    }
}

```

A continuación se descompone el contenido y funcionamiento del programa:

Configuración del Hardware;

- **Módulo de radio NRF24L01+**: Comunicación inalámbrica mediante el estándar de radiofrecuencia a 2.4 GHz
 - Pin CE (Chip Enable): GPIO 10
 - Pin CS (Chip Select): GPIO 9
 - Canal de operación: 15
- **Identificador de nodo**: 22 (dirección única dentro de la red)
- **Relé**: Conectado al pin digital 4
- **Conmutador**: Conectado al pin digital 3 (con resistencia de pull-up interna activada)

Estructura de datos y variables de estado:

El código mantiene las siguientes variables de control:

Variable	Tipo	Propósito
<code>debouncer</code>	Objeto Bounce2	Filtra ruido de rebotes mecánicos del conmutador
<code>state</code>	bool	Estado actual del relé (verdadero = activado)
<code>initialValueSent</code>	bool	Indicador de sincronización inicial con controlador
<code>msg</code>	MyMessage	Estructura de mensaje para comunicación con pasarela
<code>prev_state</code>	bool	Estado anterior del conmutador para detectar cambios

Flujo de ejecución:

Fase de Inicialización (`setup()`)

Durante el arranque del dispositivo se ejecutan las siguientes operaciones:

1. Configuración del pin del conmutador como entrada con resistencia de pull-up interna
2. Inicialización del objeto debouncer con intervalo de 10 milisegundos para filtrado de ruido
3. Establecimiento del relé en estado desactivado (`RELAY_OFF`)
4. Configuración del pin del relé como salida digital

Esta secuencia garantiza que el relé permanece desactivado durante el proceso de arranque, evitando activaciones indeseadas.

Fase de Presentación (`presentation()`)

El nodo se presenta ante el controlador central mediante:

1. Envío de información del sketch (nombre “Relay+button” y versión “1.0”)
2. Declaración del sensor hijo (`CHILD_ID = 45`) como tipo `S_BINARY` (dispositivo binario)

Esta presentación es esencial para que el controlador reconozca el nodo y establezca la interfaz de usuario correspondiente.

Sincronización Inicial (primera ejecución)

En la primera iteración del bucle principal, el código ejecuta las siguientes operaciones cuando la variable `initialValueSent` es falsa: envía el estado actual del relé al controlador, solicita el estado almacenado en el controlador, espera un máximo de 2 segundos por respuesta (`V_STATUS`), y mantiene el flag `initialValueSent` en falso hasta recibir respuesta. Este mecanismo garantiza que ante un reinicio, el nodo adopta el estado que el controlador tenía registrado, evitando discrepancias de estado.

Lectura y Procesamiento del conmutador

El sistema de detección de pulsaciones funciona según la siguiente lógica: cuando `debouncer.update()` detecta un cambio y la lectura actual difiere de la lectura anterior, el código invierte el estado lógico (toggle), actualiza `prev_state`, envía el nuevo estado al controlador y solicita confirmación (ACK). El debouncer filtra los rebotes mecánicos inherentes a los pulsadores físicos (en nuestro

caso un conmutador), evitando falsas detecciones múltiples. La detección de cambio compara la lectura actual con la anterior para identificar transiciones verdaderas. Aunque los debouncer se aplican principalmente a pulsadores y no suelen ser obligatorios en los conmutadores, sin él, el programa presentaba comportamientos impredecibles por falsas lecturas de estado del conmutador en cuanto surgía un cambio de estado, por lo que se decidió mantenerlo para añadir robustez al sistema.

Confirmación de Entrega (ACK)

El nodo procesa confirmaciones de recepción mediante la evaluación de `message.isAck()`. Cuando se recibe una confirmación de entrega, el código imprime un mensaje de confirmación en el puerto serie, pero no realiza acciones adicionales relacionadas con el control del relé.

Comando de Estado Remoto (V_STATUS)

Para comandos remotos procedentes del controlador, el código implementa el siguiente comportamiento: cuando `message.type == V_STATUS`, verifica si es la primera recepción mediante el flag `initialValueSent`. Si lo es, marca la inicialización como completada e imprime la recepción del valor inicial. Posteriormente, actualiza la variable `state` con el valor recibido, aplica el comando al relé físico mediante la función `digitalWrite()`, y reenvía una confirmación del estado al controlador.

Resultados:

Tras compilar y cargar el código en el Arduino, en el dashboard de HomeAssistant aparece el nodo y finalmente podemos controlar el estado del relé tanto a través de HomeAssistant como manualmente.

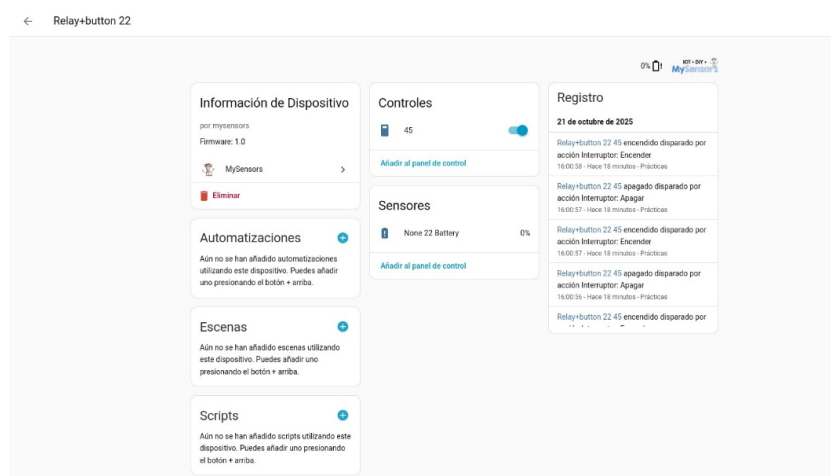


Figura 4.3: El nodo 1 en el dashboard.

Como se puede apreciar en la imagen, el interruptor software aparece con su `CHILD_ID` en lugar de un nombre, ya que no se ha especificado un nombre en la presentación del nodo hijo.

Observaciones

Las únicas incidencias surgidas en esta práctica son dos:

- **No reconocimiento del adaptador:** Se debió a que el adaptador se colocó en la posición del GPIO errónea.
- **Solapamiento de los identificadores de nodo:** Se debe principalmente a que durante el testeo iterativo del programa los identificadores eran los mismos, por lo que el gateway los rechazaba en lugar de actualizarlos. Se solucionó empleando identificadores distintos tanto para los nodos principales e hijos y también eliminando el fichero de persistencia correspondiente en HomeAssistant.

4.2. Nodo 2

En esta sección se describe la evolución del Nodo 2, concebido como sensor portátil alimentado por batería para la medida y envío de temperatura, humedad y nivel de batería dentro de la red MySensors. La implementación se ha estructurado en tres iteraciones sucesivas, con el fin de abordar de forma gradual tanto la construcción hardware como el consumo energético y la seguridad de las comunicaciones, aunque en esta memoria, solo se proporcionará el código de la última iteración y el resto de iteraciones serán comentadas.

1. **Obtención de información de la batería.**
2. **Construcción del Nodo 2 y comparativa de consumo.**
3. **Incorporación del Signing.**

Para ello, se ha empleado los siguientes componentes hardware: Para la implementación del nodo sensor se han utilizado los siguientes componentes hardware:

- **Arduino Pro Mini** basado en el microcontrolador ATmega328P(3,3 Voltios y frecuencia de operación 8 mhz), programado mediante un **adaptador USB-serie** externo.
- **Módulo de radio nRF24** de 2.4 GHz con interfaz SPI, probablemente correspondiente al siguiente modelo: <https://es.aliexpress.com/item/1005006419417915.html>.
- **Sensor digital DHT22** para la medida de temperatura y humedad relativa del ambiente.
- **Batería LiPo de 3.7 V**, con un rango de operación aproximado entre 3.7 y 4.2 V y una capacidad nominal de 380 mAh.

Se ha montado siguiendo el esquemático proporcionado por el profesor:

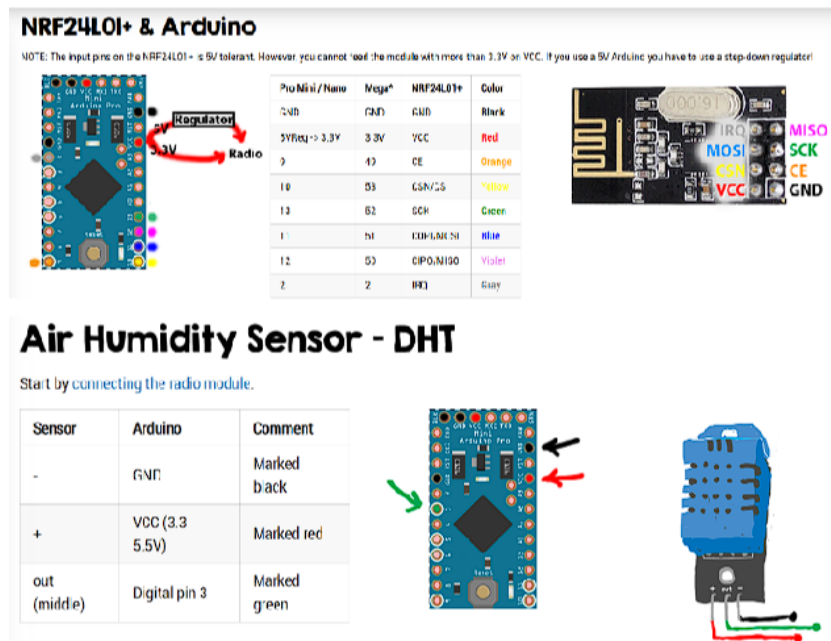


Figura 4.4: Esquemático proporcionado por el profesor

Matizando que el pin de datos del DHT se ha conectado al pin 3 del Arduino Pro Mini.

4.2.1. Obtención de información de la batería

El objetivo de esta primera iteración era simplemente averiguar el cómo podíamos acceder a la información de la batería. Usando como referencia el código de ejemplo sobre mediciones internas de MySensors. A continuación, se describe los elementos principales del código obviando la parte de comunicación de MySensors:

- **Voltaje de fuente de la CPU** Mediante la función `hwCPUVoltage()` obtenemos el valor en `long` del voltaje que posee actualmente la batería. Como se puede observar, el valor que realmente se obtiene es el valor del voltaje que ha sido suministrado a la CPU por parte de la fuente de alimentación (la batería).
- **Conversión a porcentaje** Los valores que de voltaje que emite una batería de polímero de litio de un valor nominal de 3.7 Voltios con una oscilación dependiente de su estado de carga entre 3 y 4.2 Voltios. Esto nos permite estimar la carga restante de la batería como porcentaje mediante la siguiente ecuación hiperbólica¹

$$y(x) = 83,33 \cdot \frac{x}{1000} - 250 \quad (4.1)$$

$$x \in [3000, 4200]$$

Donde y es el porcentaje y x se corresponde con el valor en miliVoltios obtenidos por la función `hwCPUVoltage()`

¹Esta ecuación es solo una estimación y no reemplaza a un controlador de carga real, que considera factores como la autodescarga, el deterioro de la batería, la intensidad de corriente, la temperatura y las condiciones de operación.

- **Lógica de envío** Para el envío se ha empleado un envío condicionado de la siguiente manera:

Si el valor medido actual es distinto al enviado anteriormente, se envía. Por el contrario, no se envía

Cabe destacar que para el envío se emplea una función específica: `sendBatteryLevel(batteryPercent)`

4.3. Construcción del Nodo 2

En cuanto al diseño software, se ha utilizado como base el ejemplo sobre DHT proporcionado por mysensors(<https://github.com/mysensors/MySensorsArduinoExamples/blob/master/examples/DhtTemperatureAndHumiditySensor/DhtTemperatureAndHumiditySensor.ino>). Para poder utilizar este código, al requerir bibliotecas externas personalizadas, ha sido necesario descargarlas y ubicarlas en la misma carpeta donde se encuentra nuestro *sketch* principal de Arduino. En cuanto a funcionamiento interno, el sketch presenta la siguiente estructura:

Configuración inicial

El código define parámetros fundamentales del nodo y la radio inalámbrica:

- ID del nodo: 90 (identificador único en la red)
- Canal de radio: 15 (frecuencia de comunicación)
- Pines: CE (9) y CS (10) para control del módulo RF24
- Protocolo: RF24 (radiofrecuencia 2.4 GHz de bajo consumo)

Los sensores registrados son: humedad (ID 91), temperatura (ID 92) y voltaje (ID 93), que actúan como dispositivos secundarios dentro del nodo principal.

Inicialización de sensores

En la función `setup()`, el código configura el sensor DHT (temperatura/humedad) en el pin 3 y verifica que el intervalo de actualización sea compatible con el sensor. Aplica un tiempo de espera inicial de 2000 ms para estabilizar la electrónica antes de las primeras lecturas.

Ciclo de medición y transmisión

El bucle `loop()` ejecuta las siguientes operaciones cada 6 segundos:

Lectura de datos

- Obtiene temperatura y humedad del sensor DHT
- Mide voltaje de la batería mediante `hwCPUVoltage()`
- Calcula el porcentaje de batería usando una fórmula lineal: $(83,33 \times \text{voltaje}/1000) - 250$

Lógica de envío selectivo

- Temperatura: se envía solo si cambió respecto a la última medición o tras 10 lecturas sin cambios (fuerza actualización cada 60 segundos)
- Humedad: aplica la misma condición de envío selectivo
- Voltaje: se transmite en cada ciclo con 2 decimales de precisión
- Nivel de batería: se actualiza únicamente si el porcentaje cambia

Manejo de errores

El código es capaz de detectar lecturas inválidas (valores NaN) del sensor DHT y registrar estos fallos a través del puerto serie sin afectar a la ejecución.

Consumo energético:

Tras verificar el funcionamiento correcto del nodo 2 se estableció una comparativa de consumo entre una versión del Arduino Pro Mini con led contra una versión sin led, obteniendo como valores pico(correspondientes a la transmisión de datos):

- **Con LED:** Se obtuvo un valor pico en torno a unos 8 miliamperios.
- **Sin LED:** Se obtuvo un valor pico en torno a unos 0.015 miliamperios, bastante inferior al consumo que se obtiene la versión con led.

El esquema de montaje a seguir ha sido proporcionado por el profesor

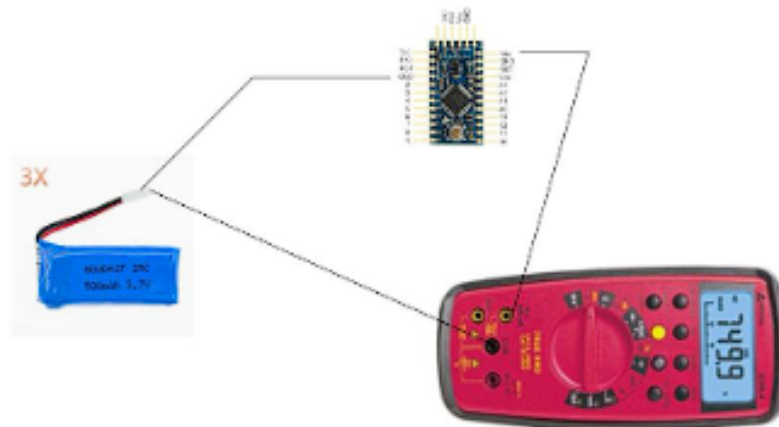


Figura 4.5: Montaje para la medición de corriente. Nótese que para la medición, se ha preferido usar los pines VCC del programador. en lugar de VDC

La fórmula aproximada para averiguar la duración de la batería sería:

$$Tiempo_{horas} = \frac{miliAmperios_{hora} * Factor_{corrección}}{miliamperios_{promedio}} \quad (4.2)$$

y asumiendo un factor de corrección² de 1 y que el consumo promedio se corresponde con el 1 % del consumo en pico³, obtendríamos los siguientes valores:

$$Tiempo_{horas} = \frac{380}{8 * 0,01} = 4750 \text{ Horas} = 198 \text{ Días} \quad (4.3)$$

■ **Sin LED:**

$$Tiempo_{horas} = \frac{380}{0,015 * 0,01} = 2533333,33 \text{ Horas} = 1055555 \text{ Días} \quad (4.4)$$

Con LED:

Como se puede observar por el resultado de la versión sin led, las mediciones obtenidas deben de ser erróneas por lo que a continuación se detallará el consumo individual teórico⁴ de los componentes (Arduino pro mini) y módulo nrf24 y se hará una estimación:

- **Arduino pro mini** según la tabla encontrada en la siguiente página web <https://www.iot-experiments.com/arduino-pro-mini-power-consumption/> el Arduino pro mini sin led en activo consume en torno a unos 3.62 miliAmperios versus 33.3 microAmperios en reposo. Mientras que la versión con led consume alrededor de 5 miliAmperios en activo y 1.152 miliAmperios en reposo

The simple @ 8MHz

STATE	INPUT VOLTAGE	POWER LED	REGULATOR	CONSUMPTION
Active	3.3V on Raw pin	On	With	4.29mA
Active	3.3V on Vcc pin	On	With	5mA
Power down	3.3V on Raw pin	On	With	1.152mA
Active	3.3V on Vcc pin	Removed	With	3.62mA
Active	3.3V on Raw pin	Removed	With	3.21mA
Power down	3.3V on Vcc pin	Removed	With	33.3µA
Power down	3.3V on Raw pin	Removed	With	31.7µA
Active	==3.3V== on Vcc pin	Removed	Removed	==3.72mA==
Power down	3.3V on Vcc pin	Removed	Removed	==4.2µA==

Figura 4.6: Tabla con los valores de corriente asociados a los estados de actividad.

- **Módulo nrf24** según la tabla encontrada en la página web oficial de MySensors (<https://www.mysensors.org/build/battery>) observamos que el módulo que probablemente se corresponda con el utilizado presenta 11.7 miliAmperios máximos para el envío y 12.2 para la recepción, mientras que en reposo consume en torno a 0.98 microAmperios. <https://www.mysensors.org/build/ba>

²El factor de corrección es una unidad inventada par atener en cuenta cosas como la autodescarga, variabilidad de la capacidad dependiente de la condiciones de operación entre otras cosas.

³Es decir a lo largo de un ciclo de 6 minutos(en el código se ha establecido una frecuencia mayor para poder hacer comprobaciones más rápidas), el promedio total de intensidad incluyendo un envío (se ha asumido que un envío tarda en torno a 100 milisegundos) se corresponde aproximadamente con un 1 % de la intensidad en pico(correspondiente a un envío)

⁴Mas bien empíricamente medido por otros

Radio Power consumption

Measured using nRF24 DevKit v1.2 (ref <https://learn.adafruit.com/nrf24-devkit>), nRF24 DevKit and gateway located next to each other.

Radio	Module	To Min/Max [mA]	To [mA]	Sleep [mA]	Link
nRF24L01+	ETape E01-M010	6.5 / 11.7	11.8	0.75	https://www.adafruit.com/item/2262374474.html
nRF24L01+	Nordic nRF24L01+ (REFM00) (PCB antenna/NCP80 crystal)	7.0 / 11.7	12.2	0.98	end of life
nRF24L01+	No name, No ack ck	6.8 / 11.7	12.0	0.94	unknown
nRF24L01+	No name, No ack inverted	11.6 / 21.5	13.7	1.85	unknown
nRF24L01+	No name, SMD castellated	11.7 / 25.9	13.7	2.02	https://www.adafruit.com/item/2542004443.html (Bakki)
nRF24L01+ PA+LNA	ETape E01-M010DPS	32.7 / 135.0	19.3	0.75	https://www.adafruit.com/item/2262374474.html
nRF24L01+ PA+LNA	No name	37.1 / 177.1	13.7	1.16	unknown

Figura 4.7: Tabla comparativa del consumo de los diferentes módulos nrf24.

Asumiendo que cuando el módulo está activo presenta una proporción 80 % del tiempo enviando y 20 % recibiendo, obtendríamos que de promedio el consumo activo es de 11.8 miliAmperios.

Módulo	Reposo	Activo
Arduino (LED)	1.152 mA	5 mA
Arduino	33.3 μA	3.62 mA
nrf24	0.98 μA	11.8 mA
Ambos(LED)	1.152mA	16.62mA
Ambos(sin LED)	34.28 μA	15.42mA

Actualizando también la intensidad promedio en términos de la intensidad pico, obtenemos que la intensidad promedio se corresponde aproximadamente con el 0.25 % de la intensidad pico en el caso sin LED y 6.96 % en el caso con LED, dando lugar a una duración estimada más razonable de:

■ **Con LED:**

$$Tiempo_{horas} = \frac{380}{16,62 * 0,0696} = 328,5 \text{ Horas} = 13,68 \text{ Días} \quad (4.5)$$

■ **Sin LED:**

$$Tiempo_{horas} = \frac{380}{15,42 * 0,25 * 10^{-2}} = 9857,32 \text{ Horas} = 410,72 \text{ Días} \quad (4.6)$$

Como curiosidad, a diferencia del código del Nodo 1 que emplea `wait()` para los periodos de espera entre mediciones, este programa emplea la función `sleep()` que coloca al microcontrolador en un estado de bajo consumo energético.

4.3.1. Incorporación de signing

El mecanismo de signing (firma en español) en MySensors es un protocolo de seguridad diseñado para garantizar la autenticidad del nodo emisor y la integridad de los datos transmitidos, previniendo ataques de retransmisión (*replay*) y manipulación por terceros (*Man-in-the-Middle*). A diferencia del cifrado, este sistema no oculta la información, sino que valida su legitimidad firmando la mediante el algoritmo HMAC-SHA256 en un esquema de desafío-respuesta estructurado de la siguiente manera:

1. **Uso de Nonce (Salt):** Para evitar la predictibilidad, el receptor genera un número aleatorio (*nonce*) que envía al emisor. Este *nonce* se incorpora en el cálculo de la firma.

2. **Clave Pre-Compartida (PSK):** Ambos nodos (sensor y gateway) almacenan una clave secreta idéntica que nunca se transmite por aire. La firma se deriva de la combinación del mensaje, el *nonce* y la PSK.
3. **Verificación Temporal:** Para contrarrestar ataques de replay retardados, el receptor impone una ventana de tiempo estricta entre la emisión del *nonce* y la recepción del mensaje firmado.

El método se puede implementar mediante hardware (empleando lecturas de resistencias o módulos criptográficos especializados). Ambas implementaciones tienen la desventaja común de reducir el tamaño del *payload*.

En el caso de esta práctica se ha optado por implementar la versión software en el Nodo 2 haciendo los cambios pertinentes que se detallan a continuación:

Habilitación del signing en el Gateway

MySensors Para ello ha sido necesario recompilar el gateway de MySensors (véase 3.3) añadiendo las siguientes banderas de configuración:

- **-my-signing=software:** Define el backend de firma de mensajes. Esta flag especifica el tipo de mecanismo de seguridad para firmar (autenticar) los mensajes en la red. Con software, se utiliza un backend de firma basado en software. otras de las opciones posibles son:
- **-my-signing-request-signatures:** Instruye al gateway para exigir que **todos** los nodos firmen sus mensajes y viceversa.
- **-my-signing-debug:** Activa los mensajes de depuración verbosos relacionados con la firma de mensajes, añadiendo información sobre:
 - Solicitudes de nonce (números aleatorios para la firma).
 - Validación de firmas.
 - Errores en el proceso de firmado.

Resultando en el siguiente comando:

```
./configure --soc=BCM2711--cpu-flags="-march=armv8-a+crc -mtune=cortex-a72"
--my-rf24-channel=22 --my-port=5003
--my-signing-debug
--my-signing=software
--my-signing-request-signatures
```

Posteriormente, se generaron las claves software con el siguiente comando:

```
sudo ./bin/mysgw --gen-soft-hmac-key --gen-soft-serial-key >> claves.txt
```

Resultando en un fichero denominado **claves.txt**⁵ que contiene las claves de serie y hmac generadas por el gateway.

⁵Las claves utilizadas posteriormente en el código no coinciden con las mostradas en la imagen, ya que estas últimas se han generado únicamente con fines ilustrativos.


```

~/
// SIGNIgn
#define MY_SIGNING_SOFT // Activa la firma software
#define MY_SIGNING_REQUEST_SIGNATURES //Requiere firmas por parte del gateway
// Enable debug prints
#define MY_DEBUG
#define MY_NODE_ID 99
#define MY_RF24_CC_PIN 9
#define MY_RF24_CS_PIN 10
#define MY_RF24_CHANNEL 15

// Enable and select radio type attached

```

Figura 4.11: Cabeceras de `dht_siging.ino`

Tras haberse cargado en el Arduino, dio como resultado lo siguiente en el dashboard de HomeAssistant pese a que el código es prácticamente idéntico al de la versión sin firma(exceptuado las cabeceras):

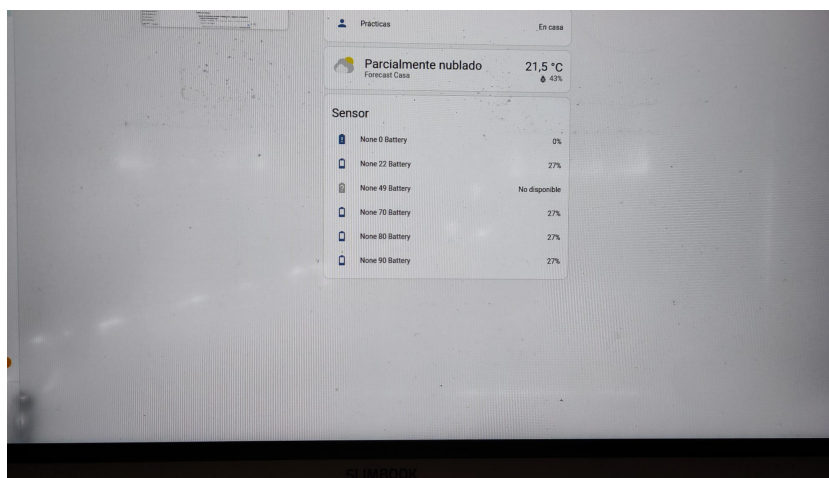


Figura 4.12: No se reconoce el nuevo sensor en el dashboard de HomeAssistant.

Sin embargo, pese a esto, el gateway de MySensors sí que reconoce y verifica el nodo tal y como se puede apreciar en la siguiente figura:

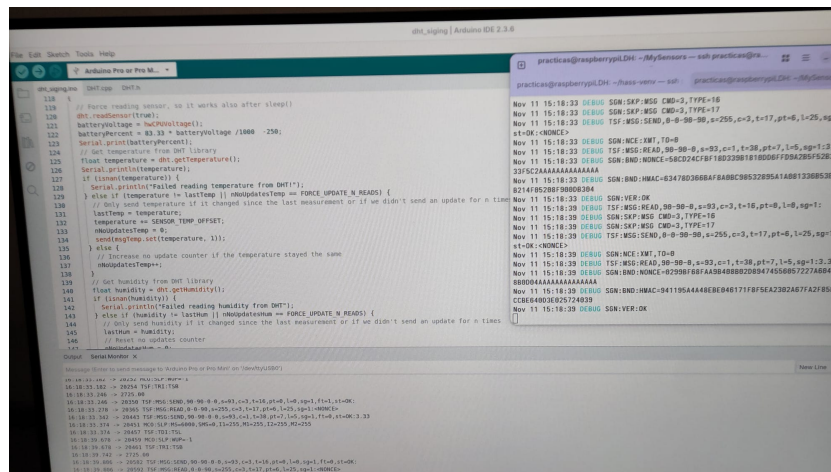


Figura 4.13: El terminal de MySensors emitiendo señales de verificación correcta junto al IDE de arduino.

Dificultades encontradas

A continuación se enumeran las dificultades encontradas a lo largo de esta parte de la práctica:

- **Configuración incorrecta de la placa:** Se debió a que se seleccionó erróneamente la configuración de frecuencia y voltaje de operación de la placa Arduino Pro Mini, lo que resultaba en mojibake⁶ en el puerto serie.
- **Versión obsoleta de MySensors:** Causó problemas tanto en el signing como en el envío de información de la batería. Se solucionó actualizando la versión de la integración de MySensors.
- **Problemas de acceso al USB:** Se debió a que, tras un formateo, el IDE de Arduino perdió el acceso del usuario al grupo de USB, impidiendo el acceso al puerto serie de la placa. Se solucionó añadiendo manualmente el usuario al grupo.

4.3.2. Versión final del código

```
// SIGNIgn
#define MY_SIGNING_SOFT // Activa la firma software
#define MY_SIGNING_REQUEST_SIGNATURES //Requiere firmas por parte del gateway
// Enable debug prints
#define MY_DEBUG
#define MY_NODE_ID 90
#define MY_RF24_CE_PIN 9
#define MY_RF24_CS_PIN 10
#define MY_RF24_CHANNEL 15

// Enable and select radio type attached
#define MY_RADIO_RF24
```

⁶<https://es.wikipedia.org/wiki/Mojibake>

```

// #define MY_RADIO_RFM69
// #define MY_RS485

#include <SPI.h>
#include <MySensors.h>
#include "DHT.h"

// Set this to the pin you connected the DHT's data pin to
#define DHT_DATA_PIN 3

// Set this offset if the sensor has a permanent small offset to the real temperatures.
// In Celsius degrees (as measured by the device)
#define SENSOR_TEMP_OFFSET 0

// Sleep time between sensor updates (in milliseconds)
// Must be >1000ms for DHT22 and >2000ms for DHT11
static const uint64_t UPDATE_INTERVAL = 6000;

// Force sending an update of the temperature after n sensor reads, so a controller showing the
// timestamp of the last update doesn't show something like 3 hours in the unlikely case, that
// the value didn't change since;
// i.e. the sensor would force sending an update every UPDATE_INTERVAL*FORCE_UPDATE_N_READS [ms]
static const uint8_t FORCE_UPDATE_N_READS = 10;

#define CHILD_ID_HUM 91
#define CHILD_ID_TEMP 92
#define CHILD_ID_VOLT 93

float lastTemp;
float lastHum;
uint8_t nNoUpdatesTemp;
uint8_t nNoUpdatesHum;
bool metric = true;

MyMessage msgHum(CHILD_ID_HUM, V_HUM);
MyMessage msgTemp(CHILD_ID_TEMP, V_TEMP);
MyMessage msgVolt(CHILD_ID_VOLT, V_VOLTAGE);
DHT dht;
int batteryVoltage;
int batteryPercent;
int oldBatteryPercent;

void presentation()
{
    wait(200);
    // Send the sketch version information to the gateway
    sendSketchInfo("Temperatura y humedad seguro", "2.0");
}

```

```

    // Register all sensors to gw (they will be created as child devices)
    present(CHILD_ID_HUM, S_HUM, "Humedad");
    present(CHILD_ID_TEMP, S_TEMP, "Temperatura");
    present(CHILD_ID_VOLT, S_MULTIMETER, "Voltaje");

    metric = getControllerConfig().isMetric;
}

void setup()
{
    dht.setup(DHT_DATA_PIN); // set data pin of DHT sensor
    if (UPDATE_INTERVAL <= dht.getMinimumSamplingPeriod()) {
        Serial.println("Warning: UPDATE_INTERVAL is smaller than supported by the sensor!");
    }
    // Sleep for the time of the minimum sampling period
    //to give the sensor time to power up
    // (otherwise, timeout errors might occure for the first reading)
    wait(2000);
    sleep(dht.getMinimumSamplingPeriod());
}

void loop()
{
    // Force reading sensor, so it works also after sleep()
    dht.readSensor(true);
    batteryVoltage = hwCPUVoltage();
    batteryPercent = 83.33 * batteryVoltage /1000 -250;
    Serial.print(batteryPercent);
    // Get temperature from DHT library
    float temperature = dht.getTemperature();
    Serial.println(temperature);
    if (isnan(temperature)) {
        Serial.println("Failed reading temperature from DHT!");
    } else if (temperature != lastTemp || nNoUpdatesTemp == FORCE_UPDATE_N_READS) {
        // Only send temperature if it changed since the last measurement
        //or if we didn't send an update for n times
        lastTemp = temperature;
        temperature += SENSOR_TEMP_OFFSET;
        nNoUpdatesTemp = 0;
        send(msgTemp.set(temperature, 1));
    } else {
        // Increase no update counter if the temperature stayed the same
        nNoUpdatesTemp++;
    }
    // Get humidity from DHT library
    float humidity = dht.getHumidity();

```

```

if (isnan(humidity)) {
    Serial.println("Failed reading humidity from DHT");
} else if (humidity != lastHum || nNoUpdatesHum == FORCE_UPDATE_N_READS) {
    // Only send humidity if it changed since the last measurement
    //or if we didn't send an update for n times
    lastHum = humidity;
    // Reset no updates counter
    nNoUpdatesHum = 0;
    Serial.println("Enviado los datos");
    send(msgHum.set(humidity, 1));
} else {
    // Increase no update counter if the humidity stayed the same
    nNoUpdatesHum++;
}

// voltage y bateria
send(msgVolt.set(batteryVoltage/1000.0 , 2));
if (oldBaterlyPercent != batteryPercent) {
    sendBatteryLevel(batteryPercent);
    oldBaterlyPercent = batteryPercent;
}

// Sleep for a while to save energy
sleep(UPDATE_INTERVAL);
}

```

Apéndice A

Creación de un sistema de alertas de glucosa lumínico

A.1. Idea principal

El objetivo de este proyecto es la implementación de un sistema de alertas lumínicas para monitoreo de glucosa integrado en la plataforma Home Assistant. El sistema proporciona notificación visual inmediata ante valores de glucosa fuera de rango mediante una bombilla inteligente WiZ, eliminando la dependencia de notificaciones móviles o alarmas sonoras <https://www.home-assistant.io/blog/2016/02/12/classifying-the-internet-of-things/classifiers/>.

La motivación clínica surge de que la persona usuaria presenta una propensión fuerte a hipoglucemias severas, requiriendo monitorización constante por parte de cuidadores. El sistema traduce los valores de glucosa en alertas lumínicas dinámicas, permitiendo que los cuidadores detecten rápidamente bajadas críticas sin necesidad de acceso directo a aplicaciones móviles, mejorando así la capacidad de respuesta ante episodios de riesgo.

Sin embargo, se ha de matizar que tras la adopción posterior de un sistema de circuito cerrado por parte de la persona usuaria ha reducido significativamente la incidencia de hipoglucemias severas, disminuyendo así la criticidad inmediata del sistema de alertas. A pesar de esto, el sistema mantiene valor como herramienta complementaria de monitorización visual.

La solución se estructura en tres capas funcionales:

1. Adquisición continua de datos desde CareLink mediante integración en Home Assistant.
2. Procesamiento y automatización mediante reglas condicionales basadas en umbrales glucémicos.
3. Control de la bombilla inteligente WiZ como elemento de notificación visual.

El nivel de glucosa en sangre de la persona usuaria se puede discretizar en 3 estados:

1. **Por encima de rango:** Cuando los niveles se encuentran por encima de 180 mg/dL. Es un estado **CRÍTICO** que requiere medidas correctoras si es prolongado en el tiempo, pero **NO URGENTE**, ya que no provoca inconsciencia.
2. **En rango:** Cuando los niveles de glucosa se encuentran comprendidos entre 70 y 180 mg/dL. Es el estado **ÓPTIMO** y **NO URGENTE**.

3. **Por debajo de rango:** Cuando los niveles de glucosa están por debajo de 70 mg/dL. Es un estado **CRÍTICO** y **URGENTE**.

Por lo tanto, la lógica que el sistema ha de emplear atendiendo a los anteriormente dichos estados es la siguiente:

- *Si es de noche (22:30 a 6:30):*
 - *Si el estado es **NO URGENTE**(en rango o por encima de rango):* La bombilla se apaga, para facilitar la conciliación del sueño.
 - *Si el estado es **URGENTE**:* La bombilla se enciende en color rojo con el brillo al máximo.
- *Si es de día(entre las 6:30 y 22:30):*
 - ***Por encima de rango:*** La bombilla se torna amarilla.
 - ***En rango:*** La bombilla se torna verde.
 - ***Por debajo de rango:*** La bombilla se torna roja.

A.2. Materiales y herramientas necesarias

Para la construcción de este sistema se han empleado las siguientes herramientas software y hardware:

- **HomeAssistant(versión OS):** Descrito en detalle en la sección 1.1. En este caso se usará la versión de Sistema Operativo, debido al que la versión de "programa" empleada a lo largo de las prácticas no permite la instalación de complementos
- **RPi 3B+:** Se ha utilizado la Raspberry Pi 3B+ como plataforma computacional. Al igual que la Rpi 4 trata de un microordenador de bajo coste que al ser una versión anterior cuenta con procesador ARM Cortex-A53 de cuatro núcleos a 1.4 GHz, 1 GB de RAM, conectividad Gigabit Ethernet, WiFi 802.11ac y Bluetooth 5.0. Aunque durante las prácticas de la asignatura se empleó principalmente la Raspberry Pi 4 del profesor, la Raspberry Pi 3B+ ofrece compatibilidad plena y es suficiente para ejecutar Home Assistant OS de forma continua.
- **HACS** (<https://www.hacs.xyz/>) Es una Integración personalizada que proporciona una interfaz de usuario para gestionar elementos personalizados en HomeAssistant, permitiendo:
 - Descubrir y descargar nuevos elementos.
 - Administrar elementos personalizados
 - Publicar elementos personalizados.

Esta integración se utilizará para descargar el componente Carelink desarrollada por el usuario de github yo-han.

- **Carelink Integration** Proveniente del siguiente repositorio <https://github.com/yo-han/Home-Assistant-Carelink> y descargada y configurada mediante HACS. Esta componente permite la interacción entre HomeAssistant y la plataforma Carelink de Medtronic de donde provienen los datos de glucosa.

- **Bombilla multicolor WiZ** Se trata de una bombilla inteligente multicolor comercial de consumo general (es decir, no es un producto *Do-It-Yourself*). Presenta compatibilidad nativa con HomeAssistant a través de su integración local push, por lo que la interacción se realiza localmente y el dispositivo avisa a HomeAssistant de los cambios en el estado que se producen. Si bien esto no quita que sea necesaria una configuración inicial mediante la app de WiZ.

A.3. Construcción

A continuación se detallan los pasos seguidos para la construcción e implementación del sistema:

A.3.1. Instalación y Configuración de HomeAssistant

Se ha seguido un procedimiento similar al descrito en las secciones 2.1 y 1.1 salvo con las diferencias que se detallan a continuación(omitiendo datos personales):

- **Instalación del SO:** En Rpi-imager aparte de cambiar de dispositivo, se ha seleccionado HomeAssistant OS (*other specific purpose OS* → *Home Automation* → *HomeAssistant OS 16.3*). A diferencia de otros SO, no ha sido necesaria establecer la configuración previa (usuario, SSH, wifi, ...)

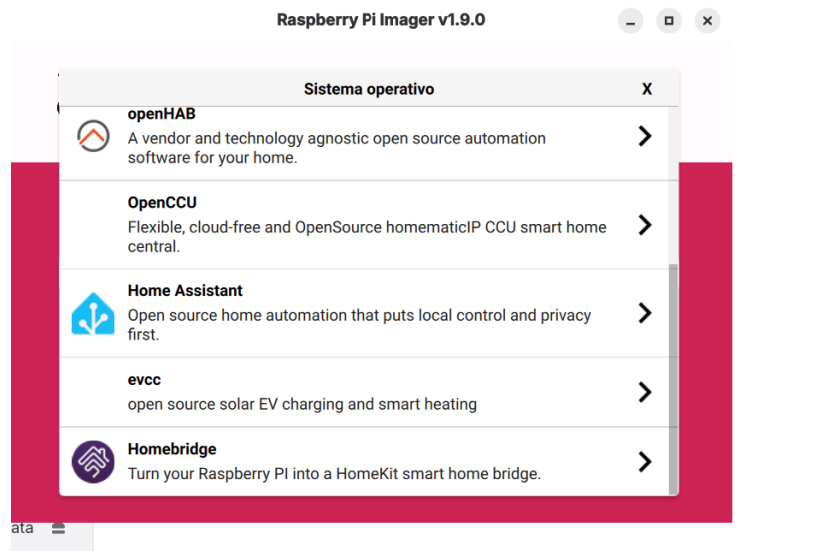


Figura A.1: Captura de Rpi-imager.

- **Configuración Inicial de HomeAssistant:** a través de la interfaz web, el procedimiento es el mismo que el descrito en la sección 1.1 con la salvedad de que se han empleado datos distintos. resultando en el siguiente dashboard:

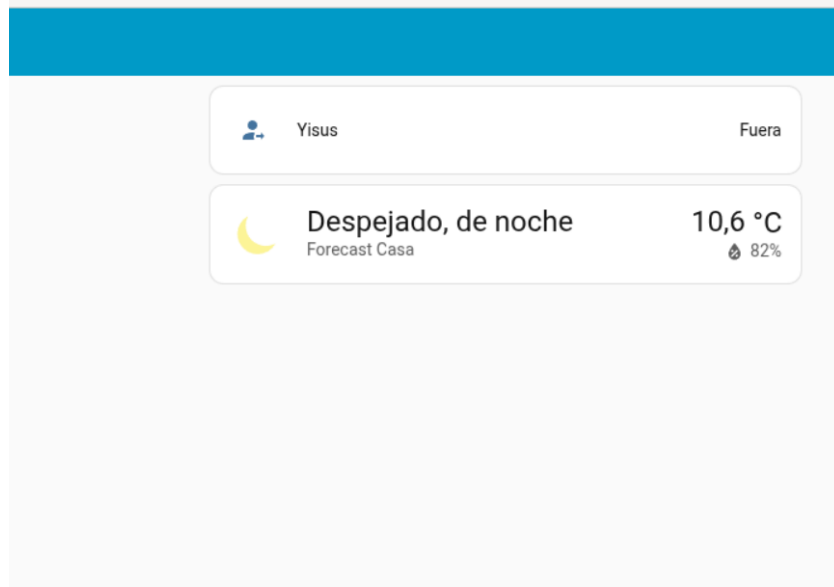


Figura A.2: Dashboard inicial.

A.3.2. Configuración de la Bombilla y la Integración Wiz

Tras esto se ha configurado la bombilla mediante la aplicación WiZ conectándola a la red local para poder así acceder a ella mediante la integración de Wiz.

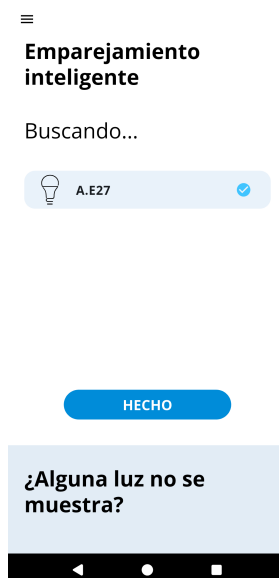
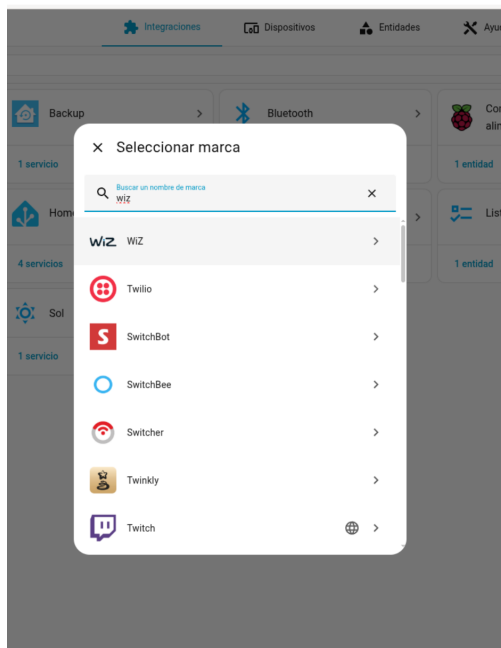


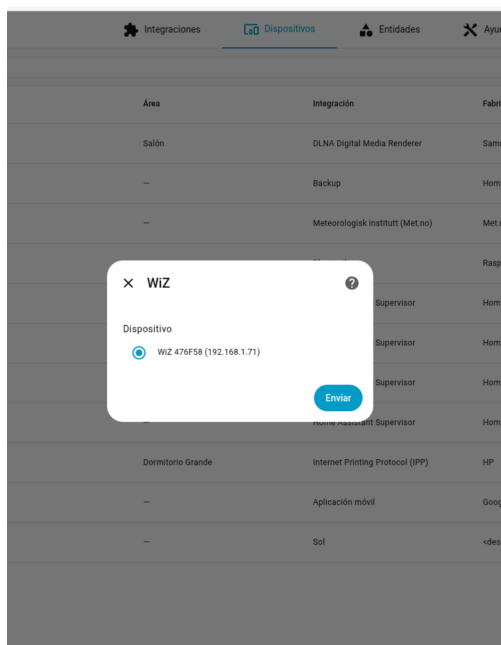
Figura A.3: Captura de pantalla de la configuración de la bombilla en la app de WiZ.

Tras agregarla a la red local, se ha proseguido en HomeAssistant con la adición de la bombilla, ejecutando los siguientes pasos:

1. Click en el menú de 3 puntos → Añadir a HomeAssistant → Añadir dispositivo → WiZ



2. Selecciona y configura la bombilla



3. Tras esto en la el dashboard aparecerá la bombilla

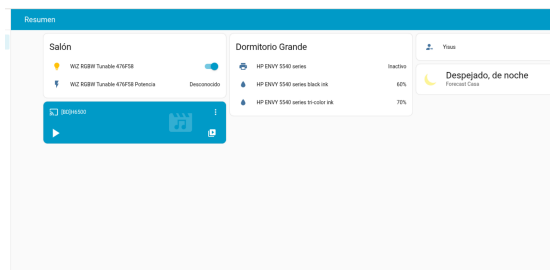


Figura A.4: Dashboard en el que aparece la recién configurada bombilla.

A.3.3. Instalación de HACS

Siguiendo los pasos de la web de HACS (<https://www.hacs.xyz/docs/use/>) instalamos HomeAssistant Community Store :

1. Instalamos el complemento que facilita la instalación, abriendo el enlace en el mismo navegador en el que hemos iniciado sesión en HomeAssistant.

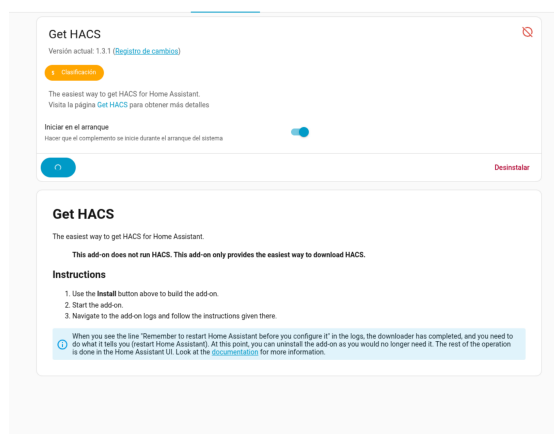


Figura A.5: Página de instalación del complemento getHACS

2. Tras esto reiniciamos HomeAssistant y limpiamos la caché del ordenador.

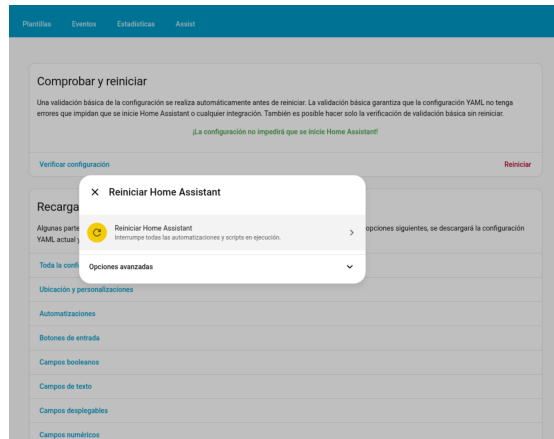


Figura A.6: Reiniciamos HomeAssistant

3. Clic en Añadir integración y luego seleccionamos la integración HACS.
4. Aceptamos las "condiciones" y vinculamos una cuenta en GitHub.

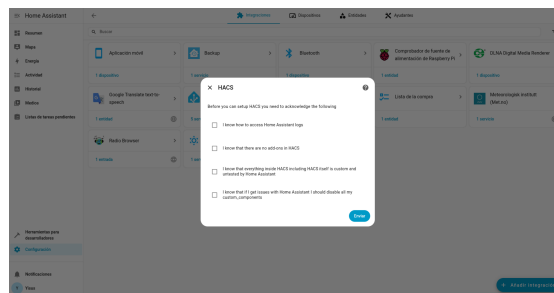


Figura A.7: Aceptamos las "condiciones".

5. Tras esto HACS se habrá configurado correctamente y aparecerá en el menú lateral del dashboard.

A.3.4. Instalación y configuración del elemento personalizado

La instalación del complemento *Carelink* se ha estructurado en dos fases diferenciadas. La primera se centra en la obtención de las credenciales de una cuenta de cuidador en Carelink previamente vinculada al paciente. La segunda aborda la configuración e instalación del componente *Carelink* en Home Assistant.

Obtención de las credenciales de cuidador

Para este proceso se ha utilizado el script desarrollado por el usuario de GitHub **plamarci**, el cual permite obtener los *tokens* de inicio de sesión de la aplicación CareLink. La ejecución se realiza empleando los siguientes comandos:

```

//Creamos una carpeta
mkdir test && cd test

// descargamos las dependencias
pip install requests
pip install pyOpenSSL
pip install selenium-wire
pip install curlify
pip install blinker==1.7.0

// descargamos el script
curl https://raw.githubusercontent.com/yo-han/Home-Assistant-
Carelink/refs/heads/develop/utils/carelink_carepartner_api_login.py
-o script.py

// ejecutamos el script
python script.py

```

Una vez ejecutado, se abrirá una ventana de Firefox solicitando el inicio de sesión. Tras completar la autenticación, se generará un archivo denominado `logindata.json` que contiene los *tokens* de acceso necesarios para la integración con la aplicación CareLink.

Instalación y configuración de CareLink

Una vez obtenidos los *tokens* de acceso mediante el procedimiento descrito en la sección anterior, se procede a la instalación y configuración del complemento *CareLink* en Home Assistant. Este componente permite integrar en la plataforma los datos proporcionados por la cuenta de cuidador, ofreciendo sensores que representan el estado del paciente, valores de glucosa, tendencias y otros parámetros operativos del sistema de monitorización.

El procedimiento de instalación a través de la UI es el siguiente:

1. En el menú lateral de la interfaz web de Home Assistant, seleccionar la opción **HACS**.
2. En el campo de búsqueda, introducir *carelink*; aparecerá una única integración correspondiente al complemento, la cual debe instalarse.

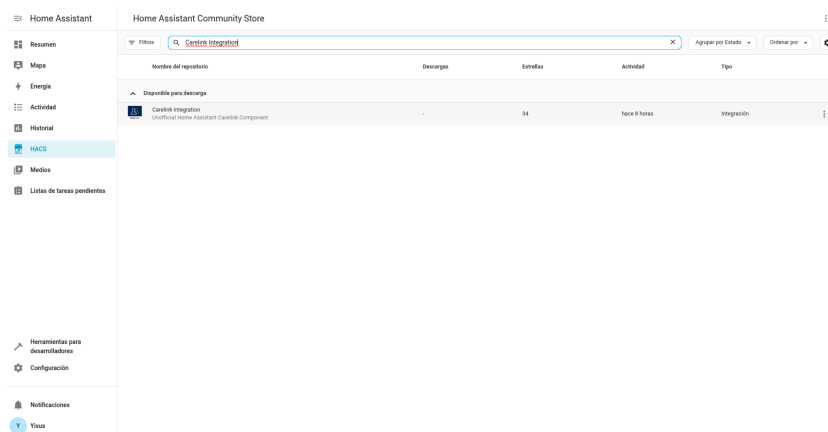


Figura A.8: Resultado de buscar Carelink en HACS

3. Antes de añadir el dispositivo, utilizar el complemento *File Editor* para cargar el archivo de credenciales obtenido previamente (`logindata.json`) en la carpeta `custom_components/carelink`.
4. A continuación, configurar el nuevo dispositivo desde el asistente de la integración. Todos los campos deben dejarse en blanco(exceptiando el intervalo de escaneo que se deja en 60), ya que el archivo con los datos de inicio de sesión ya ha sido proporcionado en el paso anterior. Tras completar este proceso, los datos del paciente asociado estarán disponibles en el panel principal de Home Assistant.

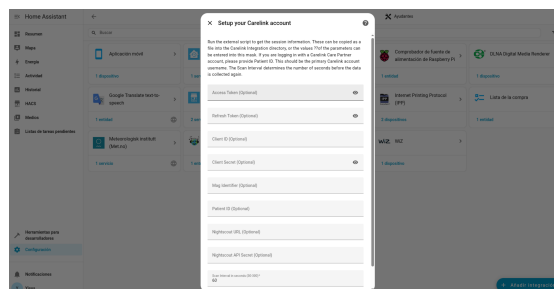


Figura A.9: Setup en blanco de la cuenta Carelink en HomeAssistant

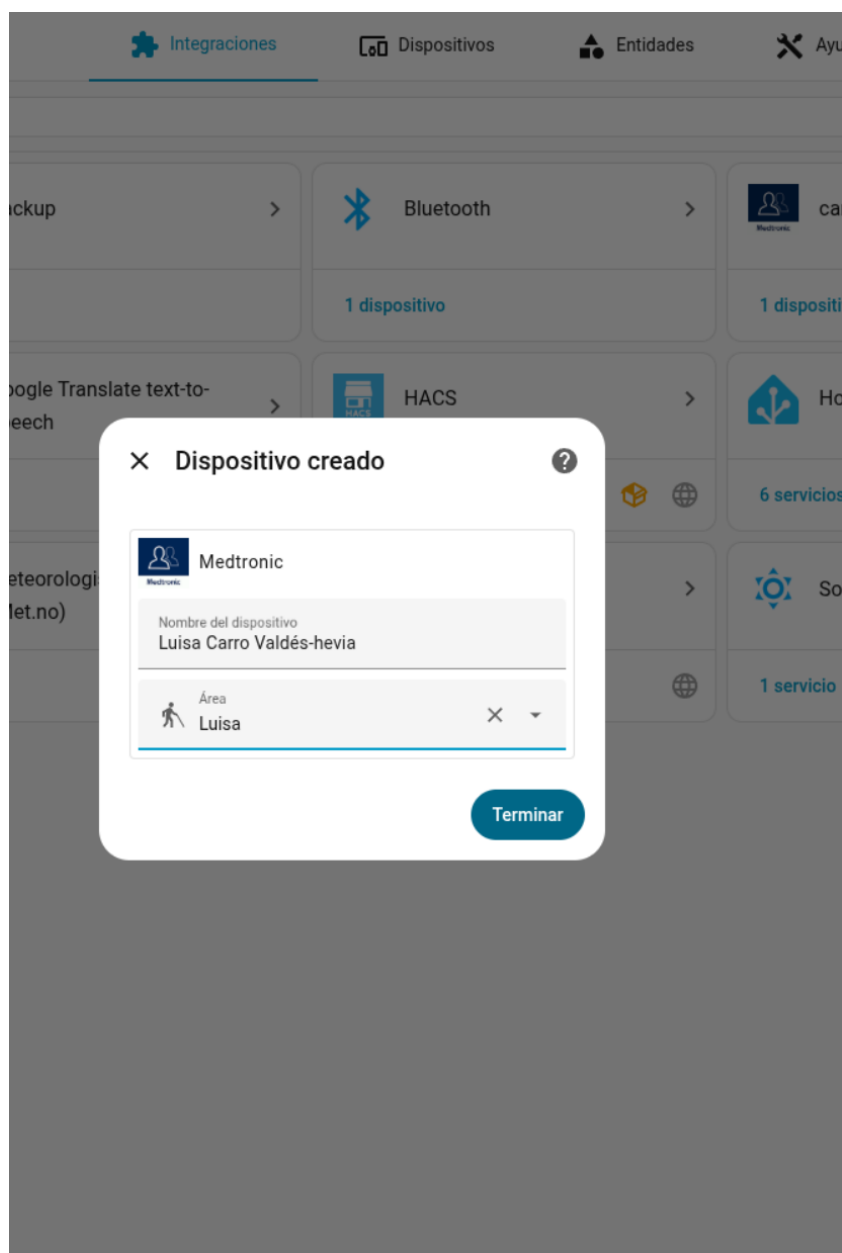


Figura A.10: Resultado de la configuración del dispositivo

A.3.5. Creación de la automatización

Una vez integrados los datos del paciente en Home Assistant, se procede a la creación de la automatización responsable de evaluar el atributo *last glucose level mg/dL*. Con este valor se implementa la lógica condicional descrita en la sección A.1, lo que permite generar acciones en función del nivel de glucosa recibido desde CareLink.

Para ello, se ha desarrollado un script que aplica dicha lógica de forma estructurada. El resultado puede apreciarse en la siguiente figura:

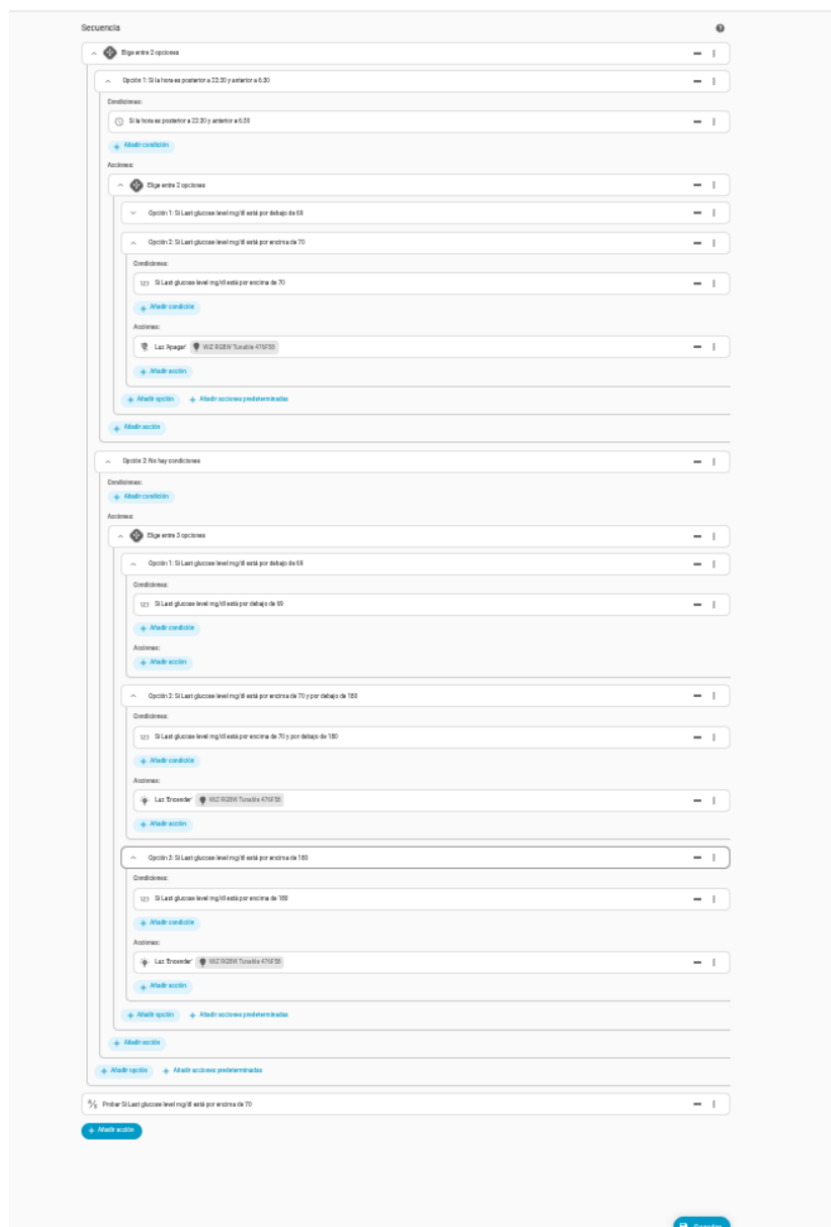


Figura A.11: Script desarrollado para la lógica condicional de los niveles de glucosa

Que se corresponde al siguiente código YAML:

```
sequence:
- choose:
```

```

- conditions:
  - condition: time
    after: "22:30:00"
    before: "06:30:00"
    weekday:
      - mon
      - tue
      - thu
      - wed
      - fri
      - sat
      - sun
sequence:
  - choose:
    - conditions:
      - condition: numeric_state
        entity_id: sensor.last_glucose_level_mg_dl
        below: 69
      sequence:
        - action: light.turn_on
          metadata: {}
          target:
            entity_id: light.wiz_rgbw_tunable_476f58
          data:
            rgb_color:
              - 239
              - 1
              - 1
            brightness_pct: 100
    - conditions:
      - condition: numeric_state
        entity_id: sensor.last_glucose_level_mg_dl
        above: 70
      sequence:
        - action: light.turn_off
          metadata: {}
          target:
            entity_id: light.wiz_rgbw_tunable_476f58
          data: {}
- conditions:
  - condition: time
    after: "06:31:00"
    before: "22:29:00"
    weekday:
      - mon
      - tue
      - wed
      - thu
      - fri

```

```

- sat
- sun
sequence:
- choose:
- conditions:
- condition: numeric_state
  entity_id: sensor.last_glucose_level_mg_dl
  below: 69
  sequence: []
- conditions:
- condition: numeric_state
  entity_id: sensor.last_glucose_level_mg_dl
  above: 70
  below: 180
  sequence:
- action: light.turn_on
  metadata: {}
  target:
    entity_id: light.wiz_rgbw_tunable_476f58
  data:
    rgb_color:
      - 23
      - 217
      - 45
    color_temp_kelvin: 5100
    brightness_pct: 97
- conditions:
- condition: numeric_state
  entity_id: sensor.last_glucose_level_mg_dl
  above: 180
  sequence:
- action: light.turn_on
  metadata: {}
  target:
    entity_id: light.wiz_rgbw_tunable_476f58
  data:
    rgb_color:
      - 251
      - 213
      - 19
    color_temp_kelvin: 6500
alias: Script Bombilla
description: Script para la bombilla de la luisa

```

Este script será invocado cada 30 segundos mediante una automatización. El objetivo de esta periodicidad es garantizar que, en caso de que el color de la bombilla se modifique accidentalmente, dicho estado incorrecto no permanezca activo durante más de 30 segundos.

Si se optara por ejecutar el script cada vez que la bombilla cambiase de color, existiría el riesgo de generar un bucle no deseado: el propio cambio de color provocado por el script podría volver a activar la automatización, produciendo una secuencia repetitiva y no controlada. La ejecución periódica evita este escenario y garantiza un comportamiento estable

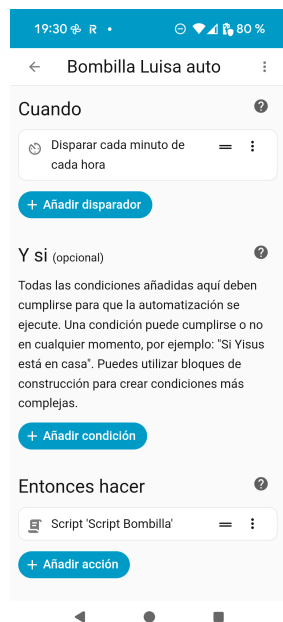


Figura A.12: Captura de pantalla de la automatización