Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Integración de un sistema de vigilancia mediante Telegram en Home Assistant

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Héctor Calatayud Sánchez

Tutor: Carlos David Martínez Hinarejos

2020-2021

Agradecimientos

En primer lugar, agradecer a mi tutor Carlos David Martínez, quien me ha ofrecido la posibilidad de realizar este proyecto y por el esfuerzo extra que ha aportado para su revisión. Agradecerle su disposición en todo momento para guiarme en la dirección adecuada, con el objetivo de llevar a cabo un proyecto de buena calidad.

En segundo lugar, agradecer a Adrián Zengotitabengoa, quien me ha motivado a introducirme en el mundo de la domótica. Gracias a su apoyo he podido llevar a cabo este proyecto tan importante para mí, en el cual he disfrutado aprendiendo y me ha hecho crecer tanto personalmente como profesionalmente.

Por último, agradecer a mis familiares, amigos y seres queridos, quienes me han apoyado incondicionalmente y han estado a mi lado desde el primer momento, ya que el ánimo y la motivación recibidos han sido de vital importancia para no rendirme y presentar el trabajo.

Resumen

En este trabajo se pretende construir un sistema de vigilancia domótico completamente automatizado capaz de enviar notificaciones gracias al protocolo MQTT a una aplicación de mensajería, en este caso Telegram. Con este fin se empleará equipo especializado de monitorización, como son las cámaras IP que funcionan mediante conexión inalámbrica y sensores de movimiento PIR (*Passive Infrared*), que serán administrados por un microcontrolador. Para ello será necesaria una plataforma como Home Assistant, de código abierto y que permite establecer una comunicación entre los dispositivos y los sensores para llevar a cabo una serie de acciones según los datos de entrada que hayan sido recibidos por éstos. El sistema de control será instalado en un miniordenador o SBC (*Single Board Computer*) como Raspberry Pi, con los recursos suficientes para poder gestionar todo el flujo de comunicación y datos, incluyendo una base de datos en SQLite que irá almacenando información de cada uno de los accesos producidos por el sistema de vigilancia.

Palabras clave: Telegram, Home Assistant, MQTT, sistema domótico, vigilancia.

Resum

En aquest treball es pretén construir un sistema de vigilància domòtic completament automatitzat capaç d'enviar notificacions gràcies al protocol MQTT a una aplicació de missatgeria, en aquest cas Telegram. Amb aquesta finalitat, emprarà equip especialitzat de monitoratge com són les càmeres IP que funcionen mitjançant connexió sense fils i sensors de moviment PIR (*Passive Infrared*), que seran administrats per un microcontrolador. Per a això serà necessària una plataforma com Home Assistant, de codi obert i que permet establir una comunicació entre els dispositius i els sensors per dur a terme un seguit d'accions segons les dades d'entrada que hagen sigut rebudes per aquests. El sistema de control serà instal·lat en un miniordinador o SBC (*Single Board Computer*) com Raspberry Pi, amb els recursos suficients per poder gestionar tot el flux de comunicació i dades, incloent-ne una base de dades en SQLite que anirà emmagatzemant informació de cadascun dels accessos produïts pel sistema de vigilància.

Paraules clau: Telegram, Home Assistant, MQTT, sistema domòtic, vigilància.

Abstract

This work aims to build a fully automated home automation surveillance system capable of sending notifications thanks to the MQTT protocol to a messaging application, in this case Telegram. This system would use specialized monitoring equipment such as IP cameras that work via wireless connection and PIR (*Passive Infrared*) motion sensors, which will be managed by a microcontroller. This will require a platform such as Home Assistant, open source and that allows communication between the devices and the sensors to carry out a series of actions according to the input data that has been received by them. The control system will be installed on a minicomputer or SBC (*Single Board Computer*) such as Raspberry Pi, with sufficient resources to be able to manage all the communication and data flow, including a SQLite database that will store information on each of the accesses produced by the surveillance system.

Keywords: Telegram, Home Assistant, MQTT, home automation system, surveillance

Tabla de contenidos

1. Introducción 13
   1.1 Motivación 14
   1.2 Objetivos 14
   1.3 Impacto esperado 15
   1.4 Metodología 16
   1.5 Estructura 18
   1.6 Convenciones 19
2. Estado del arte 21
   2.1 Qué es la domótica 21
   2.2 Aportaciones de la domótica 21
   2.3 Elementos característicos de una instalación domótica 23
   2.4 Protocolos de comunicación comunes 25
   2.5 Propuesta 26
3. Análisis del problema 27
   3.1 Identificación y análisis de soluciones posibles 28
      3.1.1 Home Assistant 29
      3.1.2 Domoticz 30
      3.1.3 openHAB 31
      3.1.4 Comparativa entre las diferentes plataformas 32
      3.1.5 Otras opciones 34
   3.2 Solución propuesta 36
   3.3 Presupuesto 37
4. Diseño de la solución 39
   4.1 Arquitectura del sistema 39
   4.2 Diseño detallado 41
      4.2.1 Conexionado eléctrico 41
      4.2.2 Diseño de la base de datos 42
      4.2.3 Arquitectura del controlador domótico 45
5. Tecnología utilizada 47
   5.1 Hardware 47
      5.1.1 Raspberry Pi 3 Model B 47
      5.1.2 Arduino UNO REV3 48

5.1.3 Velleman VMA314 49
5.1.4 EZVIZ C6N 50
5.2 Software 50
5.2.1 Home Assistant Lovelace UI 51
5.2.2 Telegram 52
5.2.3 MQTT 52
5.2.4 YAML 53
5.2.5 SQLite 53
6. Desarrollo 55
6.1 Primeros pasos en Arduino IDE 55
6.2 Pruebas en Arduino IDE 57
6.3 Instalación de Raspbian en Raspberry Pi 58
6.4 Instalación de Home Assistant 59
6.5 Primeros pasos en Home Assistant 60
6.6 Fase 1. Integración del sensor PIR mediante puerto serie 63
6.7 Fase 2. Integración de la mensajería MQTT 68
6.8 Fase 3. Integración de la cámara IP de vigilancia 69
6.9 Fase 4. Integración de Telegram. Configuración del *bot* 72
6.10 Fase 5. Integración de la base de datos. *Shell Command*. 75
6.11 Automatización y detección de movimiento 78
7. Pruebas 83
8. Conclusiones 89
8.1 Relación del trabajo desarrollado con los estudios cursados 89
9. Trabajos futuros 91
10. Bibliografía 93
11. Anexos 97

Índice de figuras

Figura 1. Ciclo completo de un sprint en Scrum 17
Figura 2. Diferentes tipos de sensores 23
Figura 3. Diferentes tipos de actuadores 24
Figura 4. Esquema de una arquitectura domótica centralizada 24
Figura 5. Matriz DAFO 28
Figura 6. Diagrama de la arquitectura del sistema 39
Figura 7. Esquema eléctrico de conexión con la placa Arduino 41
Figura 8. Conexión entre la protoboard y Arduino 42
Figura 9. Diagrama UML de la clase Access 44
Figura 10. Arquitectura de Home Assistant 45
Figura 11. Arquitectura del núcleo de Home Assistant 46
Figura 12. Raspberry Pi 3 Model B 47
Figura 13. Arduino UNO REV3 48
Figura 14. Velleman VMA314 49
Figura 15. EZVIZ C6N 50
Figura 16. Home Assistant Lovelace UI 51
Figura 17. Ventana principal de Arduino IDE 55
Figura 18. Constantes del programa de Arduino 56
Figura 19. Cuerpo de la función setup en Arduino 56
Figura 20. Cuerpo de la función loop en Arduino 57
Figura 21. Monitor serie de Arduino 58
Figura 22. Creación de una cuenta en Home Assistant 60
Figura 23. Creación de una casa en Home Assistant 61
Figura 24. Menú principal de la interfaz Lovelace 62
Figura 25. Controles del servidor de Home Assistant 63
Figura 26. Panel de administración del servidor 63
Figura 27. Activación del puerto serie en la Raspberry Pi 64
Figura 28. Configuración del sensor de puerto serie 65
Figura 29. Editar panel de control en la interfaz Lovelace 65
Figura 30. Creación de una vista en el panel de control 66
Figura 31. Menú de entidades no utilizadas 66

Figura 32. Adición del sensor de movimiento a la vista principal 67
Figura 33. Tarjeta asociada al sensor PIR con sensor desactivado 67
Figura 34. Tarjeta asociada al sensor PIR con presencia detectada 67
Figura 35. Integración MQTT broker 68
Figura 36. Integración sensor MQTT 68
Figura 37. Integración FFmpeg 69
Figura 38. Integraciones FFmpeg Motion y FFmpeg Noise 70
Figura 39. Tarjetas asociadas a los sensores de ruido y movimiento sin detección 70
Figura 40. Tarjetas asociadas a los sensores de ruido y movimiento con detección 71
Figura 41. Whitelist de directorios de Home Assistant 71
Figura 42. Inicio de BotFather 73
Figura 43. Creación de un bot en Telegram 74
Figura 44. Integración de Telegram 74
Figura 45. Integración Shell Command 76
Figura 46. Automatización para almacenar acceso por presencia 79
Figura 47. Activación del sensor de movimiento PIR 83
Figura 48. Presencia detectada en interfaz Lovelace 83
Figura 49. Notificación de presencia detectada en Telegram 84
Figura 50. Obtener acceso e imagen por identificador 84
Figura 51. Obtener accesos por rango de fecha 85
Figura 52. Eliminar un acceso por identificador 85
Figura 53. Movimiento PTZ de una cámara en Telegram 86
Figura 54. Obtener streaming de la cámara IP 87

Índice de tablas

Tabla 1. Comparativa entre las principales plataformas domóticas 33
Tabla 2. Presupuesto del proyecto 37

1. Introducción

A lo largo de los años se nos han planteado diversos problemas que con la ayuda de la tecnología se han podido resolver de una manera mucho más sencilla, cómoda y con un menor coste de personal gracias a la automatización. Es un gran avance que ha permitido que la sociedad moderna haya avanzado hasta donde se encuentra en la actualidad.

Así pues, la seguridad y la privacidad son uno de los campos donde la tecnología ofrece la capacidad de mantener un entorno específico controlado bajo ciertas circunstancias. Cuando se cumple una condición determinada, es posible notificar al usuario para que éste actúe en consecuencia.

La aplicación que se va a desarrollar emplea un conjunto de *hardware* y sensores que se comunican entre sí mediante un protocolo de mensajería y que permite gestionar un sistema domótico de vigilancia en tiempo real sin la necesidad de un intermediario, con la ventaja que ello supone y siendo aplicable en cualquier entorno particular y profesional.

Hoy en día, la mayoría de hogares e instalaciones disponen de algún dispositivo inteligente enfocado en el control local y la privacidad. Estos dispositivos pueden ser controlados mediante órdenes de voz, aplicaciones o interfaces web. El proceso llevado a cabo en este proyecto se apoya en un *software* de automatización comprendido por diferentes módulos que permiten controlar estos dispositivos inteligentes, tomando los parámetros de entrada que reciba del entorno para ejecutar una serie de acciones especificadas por el usuario. En otras palabras, el funcionamiento de este sistema estará basado en eventos.

1.1 Motivación

La principal motivación es la posibilidad de desarrollar un sistema completo automatizado con todos los componentes que requiere desde cero, desde obtener una señal eléctrica producida por un sensor a procesarla y mostrar información al usuario final de manera que sea práctico y útil. Utilizando lenguajes de programación y el equipo adecuado es posible controlar dispositivos y sistemas electrónicos y las capacidades que esto ofrece son de gran abundancia. Cada vez es mayor el número de dispositivos inteligentes (y no inteligentes), sobre todo en el hogar, que pueden conectarse entre sí para llevar a cabo una tarea descrita por el usuario.

Por tanto, resulta de interés aprender el funcionamiento de un microcontrolador, que es el que se encarga de interpretar la información obtenida por los sensores. En este caso se trata de Arduino UNO [1], un microcontrolador re-programable basado en una placa electrónica de *hardware* libre que permite establecer conexiones con diferentes sensores y actuadores de una manera sencilla, principalmente con cables DuPont. La plataforma Arduino dispone de un potente IDE con el que se pueden realizar proyectos con los componentes anteriormente mencionados y, por lo tanto, es muy flexible y fomenta el aprendizaje de cualquier persona interesada en la electrónica y la programación.

Otra razón es la motivación por adquirir conocimientos sobre sistemas domóticos debido al auge que éstos tienen en la actualidad. Resulta muy práctico conocer una plataforma *open source* como es Home Assistant y adquirir experiencia para poder elaborar cualquier tipo de proyecto en el hogar o en cualquier otro entorno. Así mismo, no implica un coste elevado y es posible integrar la mayoría de dispositivos en un espacio de tiempo muy corto, modificando únicamente unos parámetros generales de configuración que influenciarán el comportamiento de éstos.

1.2 Objetivos

Los objetivos principales del proyecto son los siguientes:

- Diseñar un sistema de vigilancia domótico para control de accesos completamente automatizado capaz de enviar notificaciones a Telegram cuando se detecte movimiento en una ubicación específica. El sistema debe ser lo suficientemente escalable para admitir un mayor número de dispositivos en multiples ubicaciones y permitir que toda la información converja en una única base de datos desde la que se pueda consultar o eliminar la información que el usuario considere conveniente.

- Integrar el conjunto de dispositivos y sensores necesarios para el correcto funcionamiento del sistema utilizando un microcontrolador Arduino UNO, tanto para la detección de movimiento (utilizando un sensor de movimiento PIR o *Passive Infrared*) como para la grabación de video (usando una cámara IP) que posteriormente será enviada al usuario para su comprobación. Adicionalmente se empleará un LED vinculado con el sensor, que se iluminará para comprobar rápidamente la correcta detección por parte de éste, y un zumbador o altavoz con el mismo objetivo, para obtener un *feedback* sonoro cuando esto suceda.

Aparte de los objetivos principales ya mencionados, existen unos objetivos secundarios que también son importantes:

- Ayudar al lector o a la lectora a introducirse en los sistemas domóticos y a discernir los componentes que pueda necesitar según el proyecto que quiera realizar, para así sacar provecho de todas las utilidades que nos brinda la tecnología a día de hoy.
- Conocer los protocolos que pueden intervenir en el control y la comunicación de dispositivos comúnmente utilizados en la domótica, como MQTT [2], ampliamente utilizado como el estándar para mensajería IoT (*Internet of Things)* [3].
- Plantear el uso de una plataforma de código abierto como Home Assistant [4], útil para ahorrar mucho tiempo de desarrollo y evitar errores en la implementación, y que es flexible y fácilmente escalable con nuevos dispositivos.
- Desarrollar soluciones que se complementen con la plataforma, como *scripts*, como alternativa cuando no exista una integración ya preparada con la funcionalidad exacta que se espera.
- Maximizar el empeño y la efectividad del sistema domótico con el menor coste económico posible y sin perder rendimiento mediante el uso de un mini ordenador como Raspberry Pi.

1.3 Impacto esperado

Las expectativas acerca de este proyecto se centran en mejorar el estilo de vida de la persona usuaria, reduciendo el esfuerzo necesario que tendrá que enfocar en cuanto a lo que seguridad se refiere. Obtendrá un control de accesos completamente automatizado que le notificará cuando se detecte movimiento, mientras el usuario disponga de conexión a Internet y sin tener que comprobar manualmente el estado de la ubicación que desee monitorizar. Adicionalmente, podrá consultar a qué hora se realizaron estos accesos y la grabación, para tener un mayor control de la estancia, y

será capaz de controlar el posicionamiento PTZ (*Pan-Tilt-Zoom*) de las cámaras IP utilizadas (si lo admiten) desde la propia aplicación de mensajería.

La persona usuaria adquirirá conocimientos de domótica suficientes para poder controlar cualquier dispositivo que requiera tomar una acción si se cumplen ciertas condiciones esperadas, únicamente con un aprendizaje mínimo. Entenderá la sintaxis YAML utilizada por Home Assistant, la cual es utilizada por cada una de las integraciones que la componen, con el fin de modificar los parámetros de configuración de un dispositivo concreto. Después de realizar un estudio del proyecto podrá escalar a otros tipos de proyectos dentro de la misma plataforma, pues el funcionamiento es muy similar para todas las integraciones de la plataforma.

1.4 Metodología

Durante este proyecto se podría definir que la metodología utilizada es una adaptación de aquella que se conoce como **Scrum** [5]. Aunque esta forma de trabajar es ampliamente utilizada por equipos de desarrollo, ha sido posible aplicar la misma en un entorno donde únicamente hay un solo miembro.

Este tipo de metodología funciona de manera que puede empezarse un proyecto sin tener mucho conocimiento en el mismo y se basa en el aprendizaje continuo y la adaptación a los factores fluctuantes. En primer lugar, deben plasmarse y enumerarse todas las necesidades del proyecto en lo que se conoce como *product backlog*, el cual reunirá las funcionalidades que debe cumplir el trabajo para que sea finalizado con éxito (y que podrá ser dinámico). Una vez tenemos este documento, arranca el ciclo de **sprint** y se realiza una planificación sobre qué tareas van a ser abordadas por el equipo de desarrollo, por lo que se require otro documento conocido como *sprint backlog*, que contendrá esta información. Este documento dividirá las tareas en los módulos o subtareas que sean necesarias y que deberán ser realizadas en un espacio de tiempo corto, normalmente entre una y cuatro semanas como máximo.

Una vez realizados los pasos anteriores se tiene como objetivo realizar un seguimiento diario o *daily scrum*. Aquí tendremos en cuenta qué tareas se realizaron el día anterior, cuáles se van a realizar el presente día y cuáles se llevarán a cabo en el futuro, recopilando todos aquellos problemas que hayan sucedido durante el transcurso. Es muy común la utilización de un tablero **Kanban** [5] para definir el orden y la planificación de las tareas, cuáles están en proceso y cuáles ya han sido finalizadas, tal y como se ha mencionado en el punto anterior. El siguiente paso será el *sprint review*, en el que se revisará el trabajo realizado para garantizar que la funcionalidad es la esperada y que se están cumpliendo las metas u objetivos establecidos. Por último, se lleva a cabo la fase final del ciclo, denominada *sprint retrospective*, en la que se analiza el *sprint* anterior con respecto al actual para encontrar problemas en el

proceso e introducir posibles mejoras que puedan ser aplicadas en la siguiente iteración. La figura 1 muestra gráficamente las etapas Scrum descritas.

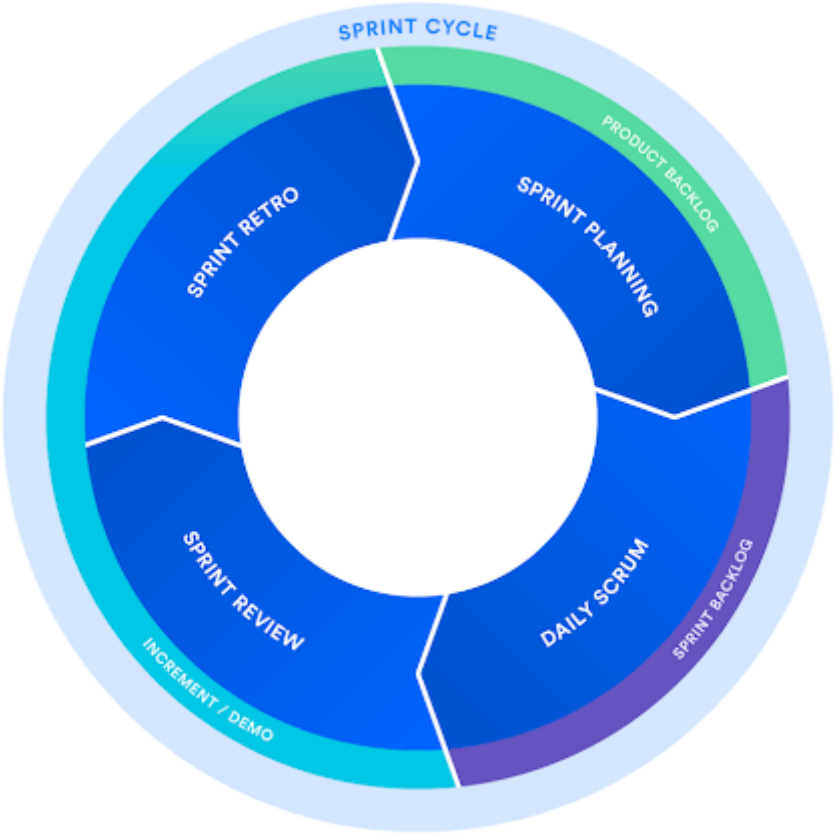

Figura 1. Ciclo completo de un sprint en Scrum

En el caso de este proyecto, hemos seguido estas pautas, fijando unas tareas generales que definen la funcionalidad que debería tener el sistema una vez esté completo y las subtareas que deben ser realizadas, pudiendo introducir detalles más técnicos en su descripción. Se lleva a cabo una subtarea al mismo tiempo y a diario se comprueba su relación con las tareas anteriores y qué elementos son necesarios para completarla, así como los problemas que presenta y cómo solventarlos. Una vez revisamos la tarea y comprobamos su correcto funcionamiento mediante pruebas con el equipo real, analizamos cómo este cambio afecta al conjunto del sistema, comprobamos si son necesarios ajustes adicionales y procedemos posteriormente a la siguiente tarea. De esta manera tenemos la ventaja de seguir un orden estructurado en el que el funcionamiento es el esperado y tenemos un mayor control de todos los elementos que intervienen en el proyecto.

1.5 Estructura

Este proyecto está estructurado en diez capítulos y un anexo:

- Capítulo 1 (Introducción): se justifica la realización de este proyecto y se detallan los objetivos que se pretenden alcanzar y sus expectativas, además de describir la metodología utilizada, la estructuración de cada uno de los capítulos y las pautas seguidas para distinguir los conceptos clave.
- Capítulo 2 (Estado del arte): se introduce el concepto de la domótica y cuáles son las características que ofrece, indicando los diferentes tipos de sensores y actuadores, además de nombrar los protocolos de comunicación más utilizados en este ámbito y de presentar la propuesta del proyecto a ser desarrollado.
- Capítulo 3 (Análisis del problema): se realiza un análisis de mercado y de las soluciones domóticas más relevantes, detallando cada una de sus funcionalidades y realizando una comparación entre ellas. Además, se justifica la elección de la plataforma domótica utilizada en este proyecto y el presupuesto de éste.
- Capítulo 4 (Diseño de la solución): se ofrece una descripción general de la arquitectura del proyecto y de la plataforma domótica, el circuito eléctrico empleado y el diseño de la base de datos utilizada en detalle.
- Capítulo 5 (Tecnología utilizada): aquí se nombran cada uno de los componentes *hardware* y *software* que intervienen en el desarrollo de este proyecto.
- Capítulo 6 (Desarrollo): se describe en detalle cada uno de los pasos realizados para llevar a cabo la configuración y la integración de todos los componentes y alcanzar el resultado final.
- Capítulo 7 (Pruebas): aquí se realizan las pruebas finales para verificar el correcto funcionamiento de la solución propuesta.
- Capítulo 8 (Conclusiones): se extraen las conclusiones al finalizar el trabajo y se relaciona éste con los estudios cursados en el grado.
- Capítulo 9 (Trabajos futuros): se presenta un posible trabajo similar a la solución propuesta con funcionalidad añadida, dirigido a los estudiantes interesados en el ámbito de la domótica y la automatización.
- Capítulo 10 (Bibliografía): se indican las referencias utilizadas a lo largo de este trabajo.
- Anexo: aquí se adjunta todo el código implementado para el correcto funcionamiento de la solución.

1.6 Convenciones

A lo largo de este trabajo se respetará el siguiente modelo para distinguir los diferentes términos que aparezcan en el transcurso del mismo:

- Las palabras extranjeras o tecnicismos y los títulos de las tablas e imágenes serán marcados en cursiva.
- Los términos importantes o de relevancia en el marco aparecerán marcados en negrita.
- Las citas a otros autores serán remarcadas con comillas dobles.
- Las referencias a fuentes externas estarán indicadas mediante corchetes.
- Se emplearán los paréntesis para aquellos conceptos que requieran información adicional o alguna aclaración.

2. Estado del arte

2.1 Qué es la domótica

En palabras de la Asociación Española de Domótica e Inmótica (CEDOM), nacida en 1992 por iniciativa de un grupo de empresas que apostaron por el sector de la domótica, e inicialmente creada como el Comité Español de la Domótica [6]:

"La domótica es el conjunto de tecnologías aplicadas al control y la automatización inteligente de la vivienda, que permite una gestión eficiente del uso de la energía, que aporta seguridad y confort, además de comunicación entre el usuario y el sistema."

El campo de la domótica pretende facilitar la vida de los humanos y mejorar el diseño de los hogares, dando respuesta a los cambios sociales y las tendencias de nuestra forma de vivir. Añade flexibilidad, polifuncionalidad y dota a nuestra vivienda de un toque más personal.

Un sistema domótico es aquel que recibe una entrada, ya sea mediante información de sensores u otros dispositivos, la procesa y ejecuta un conjunto de acciones a unas salidas o actuadores. Este sistema podría acceder a redes externas si así fuera necesario para recabar datos.

Con el tiempo, el sector de la domótica ha ido evolucionando de forma considerable, más aún en los últimos años. Hoy en día, se aportan soluciones domóticas para todo tipo de viviendas, a un coste bajo y de una buena calidad, fácilmente instalables y con gran variedad de productos. Cualquier persona podría aprender fácilmente cómo sacarle partido a cada una de las posibilidades que esta tecnología nos ofrece.

2.2 Aportaciones de la domótica

De acuerdo con el libro *Instalaciones Automatizadas en Viviendas y Edificios* [7] de José Moreno Gil, existen cuatro aplicaciones principales en viviendas domóticas, de las cuales al menos una debería satisfacerse para una instalación correcta:

- **Control y gestión de la energía**: son servicios que se encargan de racionalizar los consumos en base a diferentes criterios, como potencia contratada o tarifas. Ejemplos de uso:

  - **Desconexión selectiva de cargas eléctricas**, dando prioridad a determinados receptores, consiguiendo desconectar otros ante la conexión de los prioritarios.
  - **Programación de la puesta en marcha de receptores** cuando se aplica la tarifa más barata (doble tarifa), mediante programadores horarios, para poner en funcionamiento los receptores de más potencia en el horario con la energía eléctrica más barata.
  - **Calefacción y aire acondicionado por zonas**, delimitando zonas de calefacción y aire acondicionado para un mayor confort y aprovechamiento óptimo de los recursos.
  - **Alumbrado exterior en función de la luminosidad y la presencia**, evitando de esta manera la conexión de éste cuando haya luz natural o no se detecte presencia.
  - **Lectura remota de contadores**, evitando así las visitas domiciliarias.
  - **Información de consumos, costes y horarios de tarifas**, entre otros.
  - **Utilización de fuentes alternativas de energía**, como la solar o la eólica.
- **Seguridad**: la gestión de la seguridad tiene como objetivo proteger tanto a las personas como a los bienes. Ejemplos de uso:
  - **Detectores de gas**, capaces de cerrar la electroválvula de paso de gas al inmueble y así evitar posibles explosiones en caso de producirse escapes.
  - **Detectores de agua**, capaces de cerrar la electroválvula de paso de agua al inmueble y así evitar inundaciones y desperfectos, en caso de producirse.
  - **Detectores de humo y fuego**, conectados a centros de recepción de alarmas.
  - **Detectores de presencia**, alarmas acústicas, rotura de cristales o conexión con centros de seguridad, para prevenir intrusiones.
  - **Simulación de presencia aleatoria**, encendiendo y apagando las luces, la televisión y otros dispositivos, así como subiendo y bajando las persianas, para evitar posibles robos.
  - **Llamada telefónica al usuario ante cualquier alarma**, en caso de ausencia, y actuación de elementos de alarma (como señalizaciones ópticas o sirenas) y de elementos de seguridad (corte de electroválvulas).
- **Automatización de sistemas e instalaciones domésticas**: este grupo es el más extenso y recoge un gran número de aplicaciones. Ejemplos de uso:
  - **Monitorización del funcionamiento de los sistemas**.
  - **Control mediante órdenes** para situaciones concretas, como el apagado centralizado o la activación de los sistemas de alarma.
  - **Accionamiento automático de persianas y toldos** para el máximo aprovechamiento de la luz solar, con la posibilidad de accionamiento manual por mando a distancia.

    - **Iluminación por detectores de movimiento por infrarrojos (PIR)**.
    - **Red de aspiración centralizada** con tomas distribuidas por la vivienda.
    - **Riego exterior automático**, tomando en cuenta el terreno, la humedad, la lluvia o el viento.
    - **Distribución de las señales de audio y vídeo** por el inmueble.
    - **Videoportero automático mediante las señales de audio y vídeo**, ofreciendo la posibilidad de operar con televisión y telefonía privada.
- **Comunicaciones**: estas aplicaciones contemplan el intercambio de mensajes entre personas y dispositivos, tanto fuera como dentro del hogar. Ejemplos de uso:
    - **Envío de alarmas desde la vivienda al teléfono móvil** de la persona usuaria.
    - **Diagnóstico remoto** de la vivienda.
    - **Actuación de los receptores o sistemas a distancia**.

2.3 Elementos característicos de una instalación domótica

Una instalación domótica requiere percibir señales provenientes del exterior, como la presión, la temperatura o la presencia de personas, entre muchos otros. A nivel genérico, podría decirse que una instalación domótica se compone por los siguientes elementos [8]:

- **Sensores**: son dispositivos capaces de enviar señales al sistema domótico, recogiendo información de los diferentes parámetros que controlan, como la presión o la temperatura, para que éste actúe en consecuencia. Su función principal es transformar un estado físico del entorno en señales eléctricas. Entre los diferentes tipos de sensores se encuentran los interruptores, los detectores de presencia, los termostatos, los sensores de viento y los sensores de lluvia (figura 2).

Figura 2. Diferentes tipos de sensores

- **Controladores**: son dispositivos que reciben información del entorno, para posteriormente procesarla y llevar a cabo las acciones pertinentes mediante los actuadores.

- **Actuadores**: son dispositivos que reciben una señal de un controlador y ejecutan acciones que repercuten sobre la variable controlada, como motores, relés o pistones (figura 3).

Figura 3. Diferentes tipos de actuadores

Por lo tanto, se entiende que los sensores captan información del exterior, la transforman en señales eléctricas y la envían a los controladores para su procesamiento. Una vez procesados los datos, se llevan a cabo unas determinadas acciones mediante el uso de actuadores. El esquema de la figura 4 muestra de forma gráfica cómo es una arquitectura domótica centralizada.

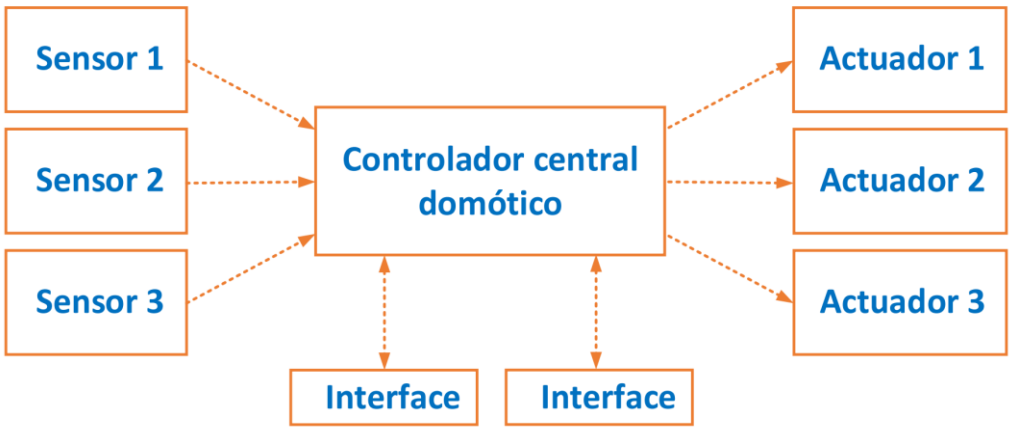

Figura 4. Esquema de una arquitectura domótica centralizada

2.4 Protocolos de comunicación comunes

Las empresas de mayor tamaño colaboraron para definir una serie de protocolos comunes, aumentando con ello la seguridad de las instalaciones domóticas. Con el tiempo, estas soluciones han ido obteniendo cuota de mercado. A continuación se van a explicar aquellos sistemas domóticos más empleados, recogidos en el libro *Instalaciones domóticas* [9], de Luis Miguel Cerdá Filiu.

Se define un protocolo como el lenguaje de comunicación que transmite instrucciones entre dispositivos, ya sea con cable o de forma inalámbrica. Los dispositivos conectados deben utilizar un estándar con el fin de poder intercambiar mensajes.

El protocolo **X10**, de origen europeo y diseñado a finales de los años setenta, se basa en la transmisión de comunicaciones por corrientes portadoras. Este sistema consiste en usar la misma red de cableado eléctrico para transmitir señales superpuestas a la corriente eléctrica, incorporando también comunicación inalámbrica. Se trata de un sistema descentralizado y de fácil instalación, pero las comunicaciones son débiles y sensibles a interferencias.

El protocolo **KNX** se basa en una conexión mediante bus. Como ventajas, es un sistema abierto e independiente del fabricante, con soporte para un gran número de dispositivos. Soporta múltiples medios de transmisión y tiene la capacidad de acoplarse a otros sistemas.

En cuanto al protocolo **LonWorks**, también conocido como LON (*Local Operating Network)*, se basa en un sistema abierto y descentralizado. Es robusto, permitiendo la comunicación entre dispositivos de una red, sin la necesidad de un controlador. En comparación con otros sistemas trabaja a altas velocidades.

Por otro lado, **Z-Wave** es un protocolo de comunicaciones inalámbricas por radiofrecuencia de baja potencia, operando en la banda de 1 GHz, lo que supone estar menos expuesto a interferencias. Existe una versión mejorada del protocolo, Z-Wave Plus, con un alcance mayor y un menor consumo energético. Actualmente es el protocolo más utilizado en la domótica inalámbrica.

Por último, **Zigbee** es un estándar para comunicaciones inalámbricas diseñado para dispositivos que requieren consumir poca energía, con menor ancho de banda y comunicación bidireccional segura. Opera en la banda de los 2,4 GHz (al igual que Wi-Fi y Bluetooth), pero apenas sufre interferencias en comparación con otros protocolos.

2.5 Propuesta

Con este trabajo se pretende reducir el impacto económico de una instalación domótica convencional, con los recursos humanos que ello conlleva. Se elaborará un sistema domótico centralizado de vigilancia con el conocimiento de la persona usuaria y sin la necesidad de compañías de seguridad. Esto reducirá el número de posibles fallos y tendremos la capacidad de controlar todos los dispositivos de nuestro hogar.

En el sistema domótico que vamos a implementar, todos los dispositivos son intercambiables y reemplazables, sin importar el fabricante, siempre y cuando utilicen los protocolos de comunicación establecidos. Es una solución rápida, efectiva y de un práctico uso cotidiano, además de ser escalable con un mayor número de dispositivos.

3. Análisis del problema

A día de hoy, cuando una persona desea controlar el acceso a su vivienda o alguna de sus localizaciones, normalmente debe comprobar cada cierto tiempo que el entorno se encuentra en condiciones correctas, bien sea de forma presencial o adquiriendo equipo de monitorización. Cuando abandona la estancia, debe inspeccionar cada cierto periodo de tiempo mediante una aplicación móvil y verificar que todo está en orden. Esto causa numerosas distracciones e impide la concentración del usuario, a la vez que implica un esfuerzo extra y la posibilidad de interrumpir cualquier tarea que esté realizando.

La finalidad de elaborar un sistema domótico es que será completamente autonómo y reactivo a un estímulo que el usuario considere conveniente, en este caso movimiento por parte de otra persona o entidad.

El sistema de vigilancia concretado en este proyecto podría clasificarse como DIY (*Do It Yourself*), por lo que no existen otros productos oficiales en el mercado con exactamente las mismas características y con el mismo proceso. Esto es porque intervienen distintos componentes o módulos independientes que se entrelazan entre sí para ofrecer un sistema personalizado con una función muy específica y a gusto del usuario.

Con el fin de definir el modelo de negocio para esta solución, se va a emplear la matriz **DAFO** [10] como herramienta de análisis estratégico y de gran utilidad para elaborar un estudio de mercado. Los factores que la componen son las Debilidades, Amenazas, Oportunidades y Fortalezas. Esta matriz ofrece un claro diagnóstico para poder tomar las mejores decisiones estratégicas en el futuro y de esta forma mejorar el producto. La figura 5 muestra el análisis DAFO para el sistema propuesto.

	Análisis Interno	Análisis Externo
Negativo	**Debilidades** ■ Diseño poco intuitivo. ■ Requiere un mínimo de aprendizaje de la plataforma. ■ Necesidad de cooperar continuamente con el cliente.	**Amenazas** ■ Aparición de compañías con productos similares. ■ Poca dificultad para replicar soluciones alternativas.
Positivo	**Fortalezas** ■ Bajo coste del sistema. ■ Desarrollo sencillo. ■ Adaptable a todas las necesidades del cliente. ■ Escalable y ampliable con multitud de dispositivos.	**Oportunidades** ■ Muchos usuarios buscando soluciones personalizadas. ■ Funcionalidades muy prácticas para el día a día. ■ Mejora cualquier sistema ya implantado. ■ Poca competencia en el mercado actual.

Figura 5. Matriz DAFO

3.1 Identificación y análisis de soluciones posibles

En este apartado vamos a analizar en profundidad cada una de las características que nos ofrecen cada una de las plataformas de domótica más importantes en la actualidad. Se explicarán las ventajas y desventajas de cada una y el motivo por el que se ha seleccionado la que ha sido utilizada en este proyecto. Finalmente, se realizará una tabla comparativa entre las plataformas mencionadas para obtener una visión más general de sus características y se ofrecerán alternativas a estas soluciones.

3.1.1 Home Assistant

Home Assistant [4] es una plataforma de código abierto escrita en el lenguaje de programación Python para la automatización del hogar, diseñada para actuar como un sistema de control centralizado (también llamado *hub*) para los dispositivos inteligentes y enfocada en el control local y la privacidad. Ofrece una interfaz de usuario para controlar estos dispositivos y es posible utilizar órdenes de voz mediante un asistente virtual como Alexa o Google Assistant.

Ofrece una serie de integraciones para soportar multitud de dispositivos IoT, así como una serie de servicios. Incluye integraciones nativas para protocolos de conexión como MQTT, Bluetooth, Zigbee o Z-Wave, además de soportar interfaces de terceros mediante el acceso a una API pública.

Introduce su propio sistema operativo, llamado Hass.io, para facilitar su uso en miniordenadores como Raspberry Pi, permitiendo controlar la instalación y extender la funcionalidad del *software* con *plugins*. Aun así, es opcional y sería suficiente con instalar la plataforma en un entorno como Linux, siendo el más recomendado, admitiendo también otros sistemas operativos como Windows o macOS.

Al realizar la instalación y ejecutarlo, abre un servidor web (puerto 8123) desde el que podemos visualizar cada uno de los dispositivos que tengamos configurados. Es posible realizar una búsqueda automática de estos dispositivos si se encuentran en la misma red o añadirlos manualmente usando la sintaxis YAML, la cual es sencilla de aprender y muy similar a XML o JSON.

La seguridad es uno de los factores más importantes a tener en cuenta y Home Assistant está enfocado en ello, teniendo en mente la privacidad y que es una aplicación *open source*, incluso superando a otras plataformas clasificadas como *closed source*. El acceso remoto está deshabilitado por defecto y los datos se almacenan únicamente en el propio equipo. Las cuentas de usuario pueden asegurarse mediante una autenticación en dos pasos para prevenir el acceso de posibles atacantes.

Como posibles desventajas, puede resultar algo difícil para usuarios poco experimentados en el tema de la domótica o sin unos conocimientos mínimos de programación o sintaxis en lenguajes de marcado.

3.1.2 Domoticz

Domoticz [11] es una plataforma de automatización del hogar muy ligera escrita en el lenguaje de programación C++ que permite configurar y monitorizar multitud de dispositivos, incluyendo luces, interruptores y sensores de todo tipo (temperatura, humedad, radiación ultravioleta, etc), el uso de electricidad y el consumo de agua, entre muchos otros. Es posible enviar alertas o notificaciones a cualquier dispositivo móvil.

El manejo de eventos es programable por el usuario, en los que se puede utilizar **Blockly** (para escribir código de manera visual utilizando gráficos entrelazados) o Lua. La primera opción es la más recomendable para empezar, al ser la más fácil y la más entendible. En cambio, con Lua podrán realizarse automatizaciones más complejas. Lua [12] es un rico lenguaje de *scripting* ideal para soluciones embebidas, potente, ligero y eficiente, que ha sido utilizado en muchas aplicaciones industriales.

La aplicación se ejecuta en segundo plano y tiene una interfaz de usuario basada en web. El puerto por defecto de la interfaz web es el 8080. Una vez realizado este paso se puede añadir *hardware*, dispositivos y cambiar la configuración. La sección de *hardware* se encarga de conectar dispositivos que se comunican directamente con Domoticz mediante protocolos como Z-Wave, Zigbee, Wi-Fi, Bluetooth o 433Mhz. Por otro lado, la sección de dispositivos ofrece una visión general que listará todos aquellos artilugios que sean detectados automáticamente, y los cuales será posible importar para su futuro uso en la plataforma. La configuración permite cambiar parámetros generales de inicio de sesión y la ubicación actual, lo que da la posibilidad de que los sensores reaccionen al clima y a la nueva zona horaria.

Respecto a la seguridad, no es necesario un nombre de usuario o contraseña para acceder a la interfaz web, por lo que si se desea proteger Domoticz (únicamente si se ha compartido el puerto al mundo exterior) es posible especificar estos datos de acceso para ingresar al sistema. Esto podría suponer una desventaja para ciertos usuarios inexpertos. Además, dispone de un panel de seguridad en el que se puede especificar si el usuario se encuentra en el hogar o está ausente, lo cual puede ser usado como sistema de alarma. Si el usuario desea compartir sus sensores a otros usuarios puede indicar el puerto remoto en el que Domoticz escuchará para conexiones remotas.

3.1.3 openHAB

openHAB (por sus siglas, *open Home Automation Bus*) [13] es una plataforma domótica de código abierto, escrita en el lenguaje de programación Java, cuyo objetivo es actuar como una herramienta centralizada para la automatización del hogar. Ofrece la posibilidad de integrar multitud de dispositivos y otros sistemas en una solución única, y proporciona una interfaz de usuario uniforme y un enfoque a reglas de automatización a través de todo el sistema sin importar el número de fabricantes o subsistemas involucrados. Es decir, es una herramienta muy flexible que se enfoca principalmente en la interoperabilidad.

Para su correcta instalación se requiere una JVM (*Java Virtual Machine*) y puede ser desplegado en múltiples sistemas operativos, como Windows, macOS o Linux, siendo recomendable utilizar una instancia dedicada de Raspberry Pi. Al igual que la mayoría de plataformas domóticas, posee su propio servidor web que es arrancado una vez es ejecutado, escuchando en el puerto 8080. Es posible realizar una búsqueda automática de dispositivos escaneando la red local para posteriormente ser utilizados en futuras soluciones.

La plataforma se comunica electrónicamente con dispositivos, tanto inteligentes como no inteligentes, que ejecutan las acciones definidas por el usuario, y es capaz de interactuar con todos los dispositivos con herramientas creadas por el usuario. Para lograr esta función, openHAB segmenta ciertas funciones y operaciones como las que son explicadas a continuación:

- **Cosas** (*Things*): son las entidades que pueden ser añadidas físicamente al sistema y que pueden proporcionar múltiples funcionalidades. Pueden ser tanto dispositivos como servicios y sólo son relevantes para el proceso de configuración. Contienen parámetros de ajuste como podrían ser una dirección de IP o un *token* de acceso.
- **Uniones** (*Bindings*): son los paquetes de *software* que son instalados por el usuario en la plataforma y cuyo objetivo es establecer una comunicación entre un dispositivo y una Cosa. Se encargan de traducir las órdenes en sentido bidireccional entre el dispositivo y la Cosa. Forman parte de los *add-ons* y gracias a ellos existe soporte para un gran número de dispositivos.
- **Canales** (*Channels*): las Cosas proporcionan Canales, que se encargan de representar cada una de las funciones que proporciona una Cosa. Por ejemplo, si una Cosa es el clima local, ésta podría tener varios Canales como son la temperatura, la presión y la humedad que la componen. Los Canales están enlazados con los Ítems, los cuales permiten que las Cosas reaccionen a eventos enviados para un ítem que está enlazado con uno de sus Canales.
- **Ítems** (*Items*): representan la capa virtual de un dispositivo, mientras que las Cosas representan la capa física. Los Ítems manifiestan la funcionalidad que es

usada por la aplicación, principalmente la interfaz de usuario o la lógica de automatización. Contienen un estado y se utilizan mediante eventos.

- **Reglas** (*Rules*): son utilizadas para automatizar tareas, pudiendo ser invocadas y a su vez ejecutar una serie de acciones cuando se han cumplido ciertas condiciones, como son los *triggers*. El motor de Reglas es potente y ligero y permite sacar partido a todas las funcionalidades que la plataforma ofrece.

Existen múltiples interfaces web que pueden ser elegidas por el usuario y el contenido variará dependiendo de la que haya sido elegida. Esto podría resultar bastante confuso para usuarios novatos, ya que dependiendo de la configuración habrá ciertas opciones que no serán mostradas, como las automatizaciones. Éstas pueden ser añadidas mediante la interfaz, definiendo el *trigger* que la lanzará y las acciones que serán ejecutadas posteriormente.

Resulta sencillo trabajar con cualquier tipo de dispositivo dado que integra más de 2000 *add-ons* en su plataforma en constante crecimiento, permitiendo trabajar con cualquier tipo de tecnología o servicio.

En cuanto a la seguridad de openHAB, también permite acceder a la interfaz web mediante HTTPS, usando el puerto 8443 establecido por defecto. Al comienzo se genera un certificado SSL autofirmado de 256 bits ECC y esto garantiza que cada instalación tiene su propio certificado individual para evitar que el servidor sea replicado. Aun así, no es posible restringir el acceso a ciertos usuarios, por lo que no se recomienda exponer el servidor a Internet. Si se desea acceder de manera remota es recomendable establecer una conexión VPN o usar el servicio openHAB Cloud, que actuará como un túnel para dirigir todas las peticiones a través de él.

3.1.4 Comparativa entre las diferentes plataformas

En la tabla 1 se muestran las características y funcionalidades de las principales plataformas domóticas, siendo todas ellas comparadas con el resto en cada uno de los aspectos.

	Home Assistant	Domoticz	openHAB
Instalación	Fácil y rápida instalación en una Raspberry Pi, pudiendo funcionar las 24 horas del día.	Es poco intuitiva y requiere una Raspberry Pi con conexión a Internet.	Utilizando OpenHabian es muy sencilla y toma entre 20 y 40 minutos, de otra manera es más laboriosa.
Configuración	Descubrimiento automático de todos los dispositivos. Para proyectos más personalizados requiere editar archivos y conocimiento de YAML.	La mayor parte de la configuración puede realizarse mediante la interfaz web, aunque es poco intuitiva.	Con la interfaz web Paper UI es posible realizar la mayor parte de las configuraciones sin editar archivos, aunque no admite todas las funciones, por lo que sigue siendo necesario editar algunos archivos.
Flexibilidad	Es bastante flexible y cubre las necesidades de la mayoría de los usuarios.	Es suficientemente estable, pero está algo limitado en términos de dispositivos y configuraciones compatibles.	Es muy flexible, pero también es un sistema muy complejo y puede resultar confuso para el usuario promedio.
Comunidad	Creciente comunidad con un gran número de usuarios dispuestos a prestar ayuda y con una documentación clara y de buena calidad.	Comunidad con un bajo número de usuarios. La documentación se encuentra desactualizada y resolver problemas concretos puede ser difícil.	Destaca por la comunidad que posee con muchos usuarios experimentados en la domótica. La documentación es muy precisa y se adapta a todo tipo de público.
Desarrollo	Nuevas actualizaciones de *gadgets* cada semana y compatibilidad con la mayoría de dispositivos.	Los dispositivos más recientes se quedan atrás en lo que se refiere a la compatibilidad, especialmente aquellos que provienen de marcas propietarias o con protocolos poco conocidos.	Desarrollo bastante lento, con un riguroso proceso de aprobación, por lo que las actualizaciones tardan en aparecer. Compatibilidad muy extendida para la mayoría de dispositivos.
Automatización	Requiere el aprendizaje de YAML para definir reglas de automatización, la cual es sencilla y es similar a XML o JSON. Puede utilizarse Python para desarrollos más complejos.	Requiere el aprendizaje de Xbase, el cual no es fácil de manejar, pero pueden crearse comportamientos complejos.	Utiliza el lenguaje de *scripting* Lua, el cual es limpio y poderoso, y no es difícil de aprender.
Seguridad	Autenticación en dos pasos (2FA). El acceso remoto está deshabilitado por defecto.	Usuario y contraseña de acceso no establecidos por defecto.	Genera un certificado SSL para evitar que el servidor sea replicado. No es posible restringir el acceso a ciertos usuarios.

Tabla 1. Comparativa entre las principales plataformas domóticas

3.1.5 Otras opciones

Existen multitud de plataformas de domótica que comparten ciertas similitudes con las principales soluciones ya mencionadas y que pueden resultar de interés para iniciar un nuevo proyecto, según el objetivo que la persona usuaria pretenda alcanzar. Se muestran a continuación una serie de alternativas y una breve descripción de cada una de ellas.

- **ioBroker**: Solución de *software* para integrar varios sistemas domésticos inteligentes en un sistema general. Se podría definir como una plataforma de integración de IoT [14].

  Tiene una estructura modular y una variedad de adaptadores que permiten la comunicación con más de 400 plataformas diferentes, desde Alexa a Z-Wave. Es multiplataforma, escalable y programable.

  Esta plataforma proporciona **VIS**, una poderosa herramienta con la que es posible presentar gráficamente los valores actuales de los sensores y el *streaming* de las cámaras, entre muchos otras posibilidades.

- **HomeGenie**: Plataforma con servidor específicamente diseñado para integrar tecnologías de automatización en una única solución [15].

  La instalación incluye compatabilidad para los protocolos más comunes como X10, Z-Wave o Philips Hue. Además contiene un editor de programas y un editor de *widgets*, por lo que es posible operar directamente en la interfaz de usuario de la web.

  Pueden analizarse los datos de los sensores y servicios externos mediante estadísticas. Estos datos pueden ser la iluminación, la temperatura, la humedad, el viento, la lluvia, el consumo de energía u otros parámetros.

  Para los usuarios más avanzados incluye una API extensible para añadir funcionalidad a la ya existente, siempre que se deseen agregar nuevas características.

- **Jeedom**: *Software* de código abierto compatible con cualquier sistema Linux [16]. Sus principales características son las siguientes:
  - Gestión de escenarios e interacción con la instalación domótica mediante texto o voz.
  - Visualización de historias y generación de gráficos y curvas.
  - Enlace de todos los dispositivos conectados.
  - Personalización de la interfaz de usuario.

- Compatible con protocolos como Z-Wave, EnOcean, KNX, Legrand Bus, RFXcom, RTS, Chacon o Edisio.

No requiere una conexión externa a su servidor dado que toda la instalación se gestiona de forma local, aumentando la seguridad de la plataforma frente a intrusiones. La persona usuaria es capaz de crear sus propias automatizaciones utilizando *widgets*, vistas y diseños.

Para administrar los dispositivos se crean **Objetos**, los cuales representan diferentes estancias del hogar. A su vez, cada Objeto puede tener un padre en la jerarquía y éste será mostrado en el panel de control.

Con el fin de interactuar con la automatización del hogar son necesarias las órdenes de información, que almacenan los datos de los sensores, y las órdenes de acción, que permiten controlar los actuadores.

- **FHEM**: Servidor perl con licencia GPL para la automatización del hogar. Se utiliza para automatizar tareas comunes como encender o apagar interruptores y registrar eventos como la temperatura, el consumo de energía o la humedad. El servidor puede ser controlado vía web, telnet o TCP/IP [17].

  Es compatible con multitud de dispositivos y protocolos como KNX, Z-Wave, EnOcean, Philips Hue, FRITZ!DECT, Intertechno, HomeEasy o X10.

  Las características más importantes son las que se indican a continuación:
  - Permite crear dispositivos o registros (pueden ser almacenados en archivos o una base de datos) al recibir datos de un nuevo dispositivo.
  - Notifica a programas externos o scripts al recibir ciertos eventos.
  - Contiene órdenes basadas en el tiempo, es decir, serán ejecutadas en un momento determinado.
  - Soporte para multitud de interfaces como texto plano, JSON o XML, cada una de ellas sobre TCP/IP, SSL o HTTP.

  Posee una arquitectura modular con más de 400 módulos en la actualidad y que pueden ser utilizados con un gran número de dispositivos, incluso con los más inusuales. Como dato adicional, es posible modificar la interfaz a gusto de la persona usuaria.

- **LinuxMCE**: Plataforma de domótica basada en Linux que permite la gestión de medios, seguridad, telecomunicaciones, redes y otros sistemas para el hogar. Es posible organizar todo el contenido mediante etiquetas de metadatos y reconoce cualquier dispositivo automáticamente, incluso si éste no se encuentra en la misma red [18].

Incluye la posibilidad de monitorizar sensores, cámaras de seguridad y desencadenar eventos. Es compatible con el protocolo VoIP con una configuración mínima.

Puede sustituir a un router estándar, contiene un *firewall* avanzado y un QoS (Quality of Service) de nivel alto para garantizar que el ancho de banda se usa efectivamente. El clima y la iluminación del hogar también pueden ser inspeccionados y automatizados dado que admite protocolos como X10, Z-Wave e Insteon, entre muchos otros.

- **Switchur**: Extensión libre de IFTTT, Zapier y Microsoft Power Automate que permite crear rutinas de automatización para acceder a múltiples fuentes de datos, sensores, eventos y disparadores. Esto es gracias a un constructor de lógica y un motor de expresiones que proporciona un control total a la persona usuaria [19].

  Existe la opción de crear ítems Switchboard y automatizaciones. Los ítems Switchboard son conexiones en tiempo real con los sensores inteligentes y servicios por medio de IFTTT, Zapier y Microsoft Power Automate. Es una herramienta muy flexible que permite automatizar una gran variedad de escenarios.

3.2 Solución propuesta

Home Assistant se presenta como la solución elegida para este proyecto debido a la facilidad de instalación de su servidor, además de ser una herramienta muy potente y ligera, con unos requisitos poco demandantes en *hardware*. La documentación que contiene es bastante precisa y permite a cualquier persona usuaria introducirse con mucha sencillez en el mundo de la domótica y sacar partido a todas las integraciones de la plataforma con un mínimo conocimiento de programación. Presenta compatibilidad con la mayoría de protocolos existentes en la actualidad, es muy seguro, escalable y ampliable. Las automatizaciones pueden crearse desde la interfaz o un archivo de configuración, tal como ocurre con las integraciones que la componen. La comunidad que la conforma es la más grande en comparación con el resto de plataformas, por lo que si hubiera algún problema podría recibirse ayuda para completar cualquier proyecto que se esté llevando a cabo.

Domoticz, en cambio, ofrece la posibilidad de crear nuestras propias automatizaciones mediante gráficos, o bien mediante un lenguaje de *scripting* para tareas más avanzadas. Sin embargo, no es la opción más segura y su uso puede resultar algo complejo para un usuario nuevo, además de que la comunidad que posee tiene un bajo número de usuarios, lo que podría aumentar la dificultad para finalizar un proyecto si se desconoce la solución a un determinado problema.

Por otro lado, openHAB es una solución completa que contiene más de 2000 *add-ons* y una comunidad con un gran número de usuarios, pero la complejidad de instalación y creación de automatizaciones, así como de la administración de la interfaz de usuario, provoca que no sea la opción más idónea para el usuario promedio. Es necesario un conocimiento básico de programación y la interfaz es poco intuitiva.

Todas las opciones presentadas en este apartado funcionan mediante un servidor desde el cual es posible realizar un control total desde la interfaz web y son aptas para emprender un proyecto de automatización para el hogar, dado que cuentan con una gran compatibilidad de protocolos para conectar prácticamente cualquier dispositivo existente en la actualidad, inteligente o no inteligente, y dotarlos de un comportamiento que será programado por la persona usuaria con la ayuda de sensores externos y actuadores.

3.3 Presupuesto

Para llevar a cabo el proyecto ha sido necesaria la adquisición de distintos componentes electrónicos, incluyendo el SBC Raspberry Pi, la placa Arduino, herramientas para el conexionado y otros materiales auxiliares para el funcionamiento del sistema. En la tabla 2 se refleja que el coste del proyecto resulta económico sin la necesidad de adquirir equipamiento de alta gama.

Producto	Unidades	Precio
Arduino UNO REV3	1	20,00€
Raspberry Pi 3 Model B 1GB RAM	1	44,89€
Protoboard	1	4,99€
Kit básico de componentes (incluye resistencias, LEDs y zumbadores)	1	7,99€
Piezas de cable DuPont	1	6,99€
Fuente de alimentación micro USB 5V 3A 15W	1	9,99€
Cámara de vigilancia IP EZVIZ C6N 1080p PTZ	1	39,99€
Detector de movimiento PIR Velleman VMA314	1	7,25€
Teclado USB compacto	1	21,96€
Pantalla táctil LCD 7”	1	72,99€
TOTAL		**237,04€**

Tabla 2. Presupuesto del proyecto

De los productos mencionados en la tabla anterior, y si se deseara abaratar el coste general, podrían eliminarse la pantalla táctil y el teclado compacto. La pantalla sirve como herramienta auxiliar para Raspberry Pi con el objetivo de no depender de una

pantalla externa en todo momento, lo que agiliza el desarrollo del proyecto. Por otro lado, el teclado cumple el mismo objetivo y es necesario si no se dispone de una unidad. Ambos son prescindibles si se configura un acceso remoto (por ejemplo, mediante xRDP) al mini ordenador, por lo que el precio total del proyecto descendería a 142,09€.

4. Diseño de la solución

4.1 Arquitectura del sistema

Para este proyecto necesitaremos una tensión eléctrica de 5V para alimentar tanto el SBC Raspberry Pi como la placa Arduino UNO, que estará conectada mediante USB a éste. El diseño estará dividido en dos módulos, en el que uno se comunicará con el otro para transmitir los datos que serán recibidos por el entorno, para así poder ser procesados y ejecutar una serie de acciones en consecuencia. El diagrama de la figura 6 se muestra cómo se dividen los bloques y qué relación tienen entre ellos.

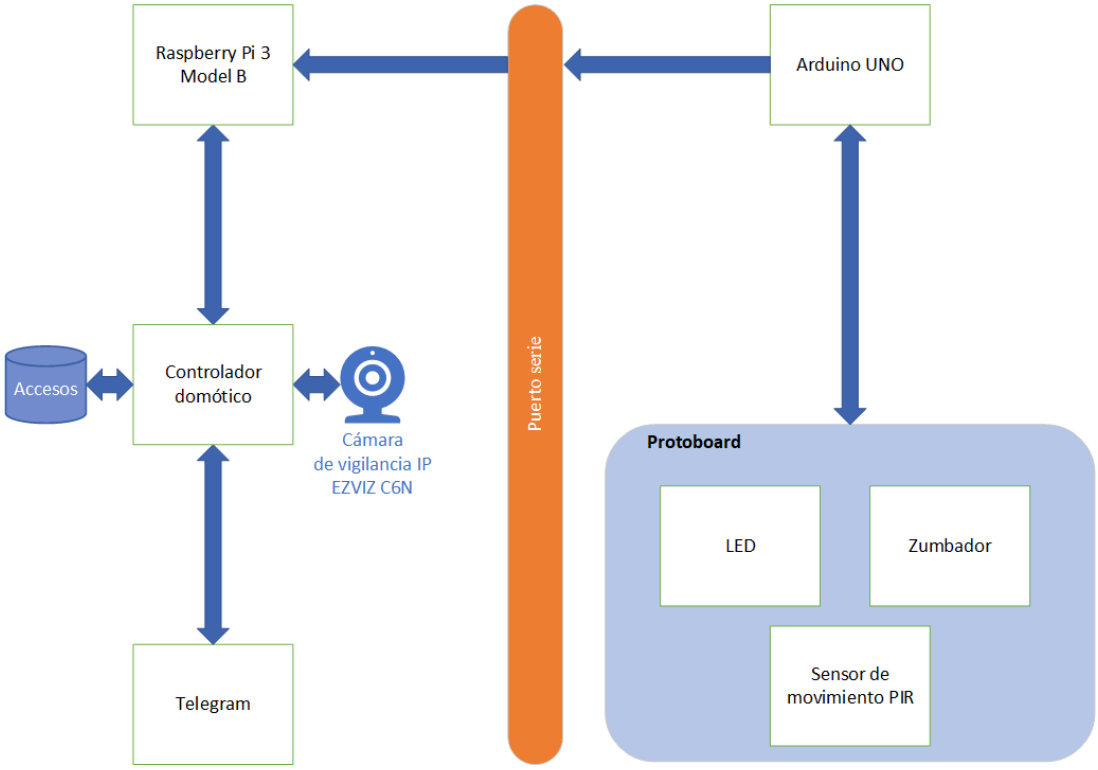

Figura 6. Diagrama de la arquitectura del sistema

En la figura 6 podemos observar que existen dos módulos separados por el puerto serie, que actúa como conexión entre ambos. El módulo situado en la parte derecha contiene los componentes eléctricos y en el segundo módulo situado en la parte izquierda se ubican los componentes lógicos que determinarán las acciones a realizar dependiendo de los datos provenientes del módulo anterior.

En primer lugar, la *protoboard* conforma un bloque en el que se encuentran los elementos eléctricos. El sensor de movimiento PIR detecta el movimiento presente en

el entorno siempre que el objeto se encuentre en su campo de acción, y este evento es controlado por el microcontrolador Arduino UNO. Cuando esto sucede, el microcontrolador se encarga de encender la luz LED para indicar de forma visual que se ha producido una detección de movimiento (de ahora en adelante lo denominaremos **acceso**), así como de emitir un sonido agudo mediante el zumbador. Finalmente, este evento es enviado a través del puerto serie del microcontrolador utilizando la cadena de texto "Sensor ON" en sentido **unidireccional** para posteriormente ser manejado por el mini ordenador. En cualquier otro caso se enviará de forma periódica la cadena "Sensor OFF", que es ignorada y no recibirá ningún tipo de procesamiento.

Por el otro lado, disponemos del mini ordenador Raspberry Pi, que es el elemento principal del módulo lógico y el cual está conectado al microcontrolador por el puerto serie mediante un cable USB. Este componente integrará un controlador domótico, en este caso Home Assistant, cuyo objetivo principal es el de manejar el evento recibido por el sensor de presencia y ejecutar una serie de acciones descritas por la persona usuaria. Este evento será procesado cuando se reciba la cadena "Sensor ON" desde el puerto serie del microcontrolador, y no se realizará ninguna acción en caso contrario. El controlador se encargará de conectar con todo el equipo necesario y de interactuar con la base de datos, que contiene un registro de todos los accesos producidos. Sólo será necesario emplear como dispositivo externo una cámara IP de vigilancia, cuya funcionalidad será la de obtener el *streaming* de éste una vez se haya recibido el evento por parte del sensor.

El objetivo principal de esta arquitectura es que el usuario final sea capaz de recibir una notificación cuando el evento anteriormente mencionado se produzca. Para ello, es necesaria una aplicación de mensajería como Telegram [20], la cual dispone de una API pública mediante *bots*, para obtener la información que es enviada por el controlador domótico. La persona usuaria recibirá estos datos y actuará según su criterio. Cabe destacar que tanto la aplicación de mensajería como la base de datos y la cámara IP actúan en sentido **bidireccional**. Es posible que la persona usuaria interactúe con Telegram mediante órdenes para solicitar datos a Home Assistant y que éstos sean enviados de vuelta a la aplicación de mensajería para su visionado. La base de datos que contiene los accesos podrá devolver la información solicitada de éstos al controlador (lectura), así como de insertar nuevos accesos (escritura). Por último, Home Assistant enviará la orden de grabación a la cámara y ésta devolverá el *streaming* solicitado de vuelta al mini ordenador para su escritura en disco.

Para el correcto funcionamiento de este diseño se requiere una conexión inalámbrica de alta velocidad, dado que se transmitirán continuamente datos provenientes de la cámara en formato de vídeo e imagen desde Home Assistant a la aplicación de mensajería. Además, ésta deberá ser persistente y ser accesible desde el exterior del hogar mediante Internet. Con el fin de realizar una simulación se podría utilizar un cable Ethernet entre la cámara y la Raspberry Pi para mayor velocidad de descarga.

4.2 Diseño detallado

4.2.1 Conexionado eléctrico

En el esquema eléctrico de la figura 7 se muestra la conexión eléctrica entre la placa Arduino y la *protoboard*, incluyendo los componentes eléctricos utilizados durante el proceso.

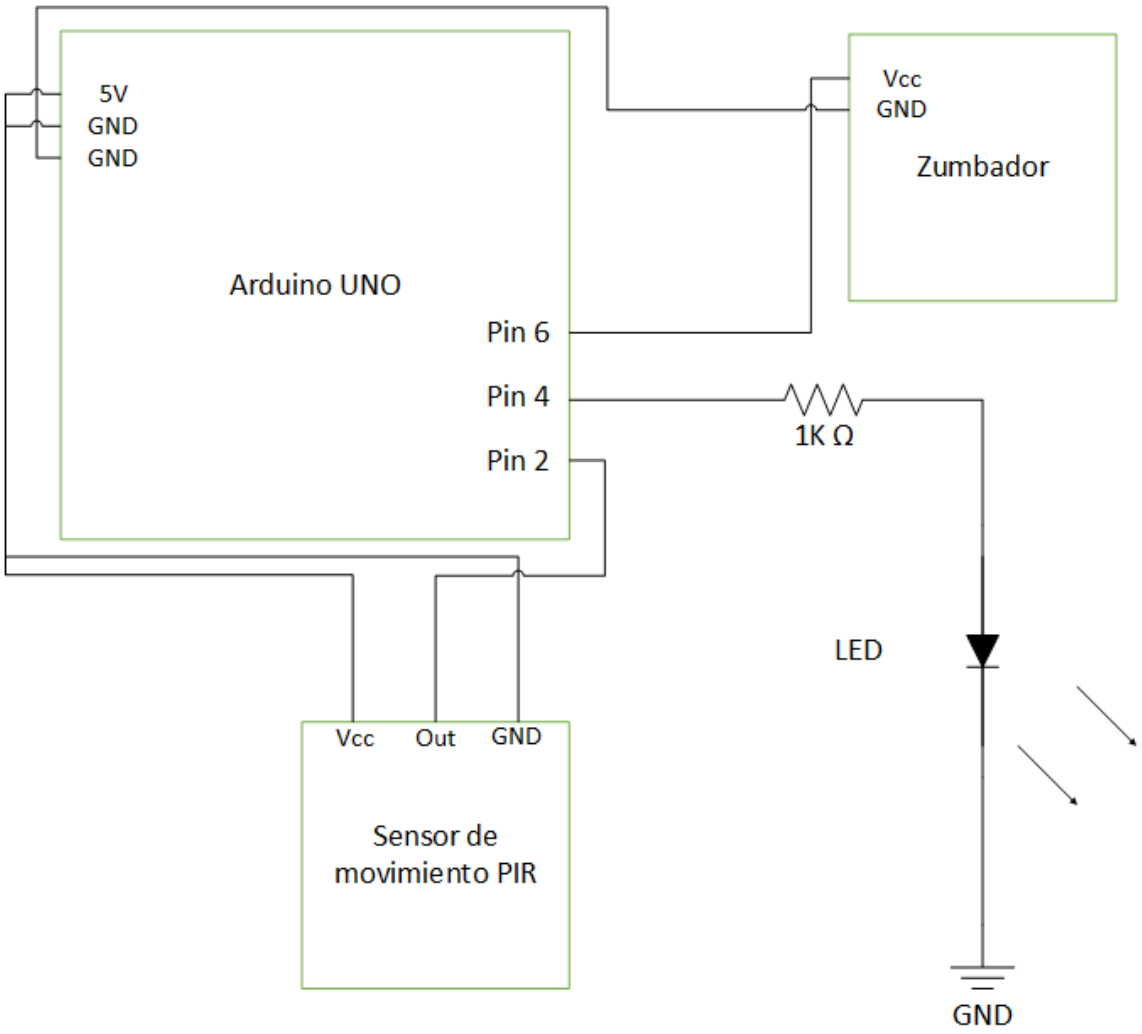

Figura 7. Esquema eléctrico de conexión con la placa Arduino

En el esquema anterior se pueden distinguir los principales componentes eléctricos, como son el sensor de movimiento PIR, el diodo LED y el zumbador, todos conectados a uno de los pines de la placa Arduino UNO, la cual actuará de microcontrolador. El sensor empleará una señal de salida conectada al pin 2 del microcontrolador que se activará cuando éste reciba cambios en la radiación infrarroja y será de tipo ON/OFF. El diodo LED requiere una resistencia de 1000 Ohm y será

encendido a través del pin 4 cuando el evento anterior se produzca, al igual que ocurrirá con el zumbador, que emitirá un sonido y será manejado mediante el pin 6. Es importante mencionar que la conexión entre el zumbador y la placa se realiza utilizando un pin de tipo PWM (*Pulse-Width Modulation*), que permite controlar el voltaje que sale del pin y con ello variar el sonido producido por el zumbador.

El resultado de mapear el circuito en la protoboard sería el mostrado en la figura 8:

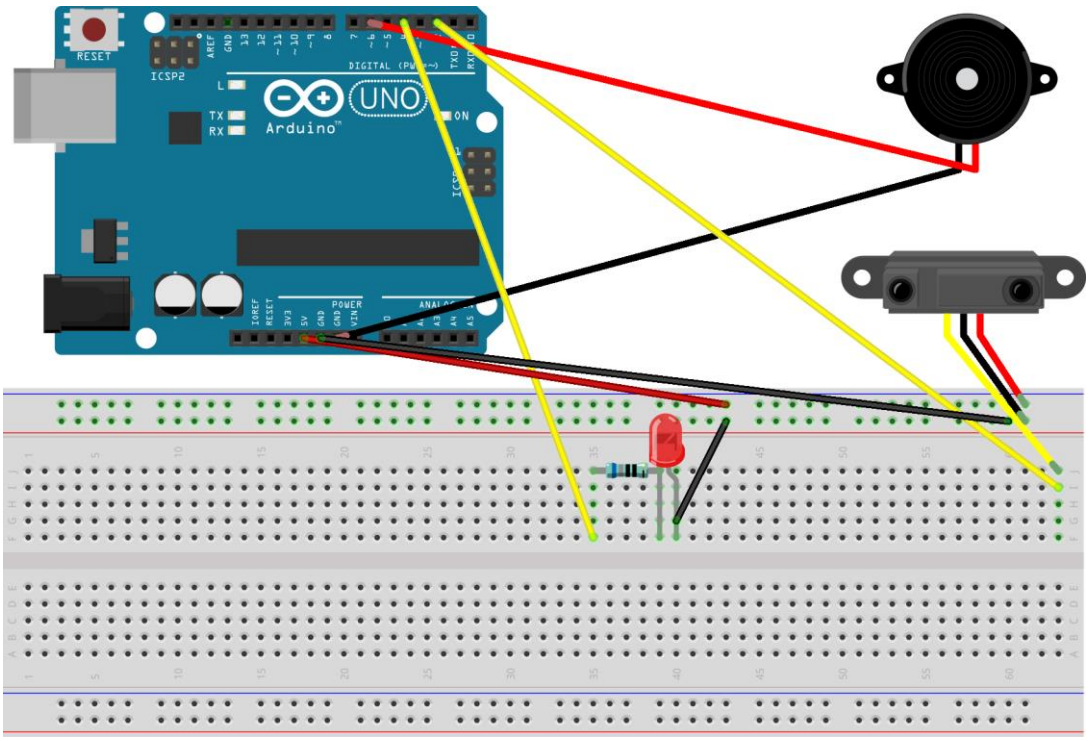

Figura 8. Conexión entre la protoboard y Arduino

Una vez realizadas las conexiones entre la protoboard y Arduino, el siguiente paso será conectar la placa a la Raspberry Pi mediante un cable USB tipo B, que actuará como un puerto serie. Esto sería suficiente para dotar de alimentación a la placa sin la necesidad de cableado adicional.

4.2.2 Diseño de la base de datos

Para registrar cada una de las detecciones de movimiento captadas por el sensor PIR se va a proceder a crear una base de datos de accesos, que contendrá toda la información necesaria y de interés para la persona usuaria relativa a éstos. Estará compuesta únicamente por una tabla denominada **Access**, que contendrá las siguientes columnas:

- **id** (Identificación): identificador único de cada uno de los registros de la tabla. Es un número de tipo entero autoincrementable.
- **code** (Código): número de serie único compuesto por el prefijo "AC", seguido de la fecha y hora de creación del acceso en formato americano y sin espacios. Por ejemplo, el código AC-20210618183354 indica que el acceso se ha producido el día 18 de junio de 2021 a las 18:33:54. Es un identificador alternativo que permite consultar un registro si se desconoce el identificador principal.
- **created_date** (Fecha de creación): fecha y hora donde se ha producido la detección de movimiento por parte del sensor. Es el dato más importante al momento de filtrar y consultar los registros. Es obtenido automáticamente por el sistema.
- **location** (Ubicación): estancia del hogar donde ha tenido lugar el acceso. Es una cadena de texto que se encuentra en la configuración de la plataforma domótica y que va asociada a una cámara de vigilancia.
- **snapshot_path** (Ruta de imagen): nombre de archivo que incluye el directorio donde se encontrará la imagen obtenida por la cámara cuando se produzca un acceso. Este directorio se encuentra en */config/photos* y el nombre del archivo será único y estará compuesto por el prefijo "snapshot" seguido de un *timestamp*. Un ejemplo de nombre de fichero sería "snapshot_2021-05-28T11.46.00".
- **record_path** (Ruta de grabación): nombre de archivo que incluye el directorio donde se encontrará la grabación obtenida por la cámara cuando se produzca un acceso. Este directorio se encuentra en */config/videos* y el nombre del archivo será único y estará compuesto por el prefijo "record" seguido de un *timestamp*. Un ejemplo de nombre de fichero sería "record_2021-05-28T11.46.00".
- **automatic** (Automático): indica si el acceso actual se ha generado de forma automática por detección del sensor PIR, o si por el contrario se ha creado manualmente cuando la persona usuaria ha solicitado a través de la plataforma de mensajería obtener el estado de una ubicación concreta del hogar.

La base de datos que se presenta está diseñada con el SGBD (Sistema de Gestión de Bases de Datos) conocido como SQLite [21], pensado para ser muy ligero y eficiente, y contendrá la información más relevante e imprescindible con la que la persona usuaria desea interactuar. Ésta se ubica en la ruta */home/homeassistant/.homeassistant* con el nombre de archivo *access-control.db*.

La clase que representa un acceso se denomina también **Access** y está escrita en el lenguaje de programación Python. Esta clase se utilizará para insertar nuevos accesos, así como para recuperarlos o eliminarlos. Esto es posible gracias al ORM (*Object Relational Mapper*) llamado SQLAlchemy, que trabaja a alto nivel con el motor SQL

para construir automáticamente el objeto deseado. Se ubica en el archivo *models.py* en la ruta */home/homeassistant/.homeassistant/scripts*. En la figura 9 se detalla el diagrama UML que representa la clase mencionada y que contiene los métodos a ser utilizados.

Figura 9. Diagrama UML de la clase Access

Los métodos contenidos en esta clase son utilizados al interactuar en la plataforma de mensajería mediante órdenes. Según el tipo de orden se ejecutara un método u otro y devolverá el resultado esperado con los argumentos dados.

4.2.3 Arquitectura del controlador domótico

Home Assistant proporciona una plataforma para el control domótico y la automatización del hogar. Funciona como un sistema embebido con una configuración y actualización sencillas mediante la interfaz de usuario. La arquitectura de este controlador está dividida en tres capas, desde la más interna a la más externa: núcleo, supervisor y sistema operativo. La figura 10 muestra esta división de una manera más gráfica.

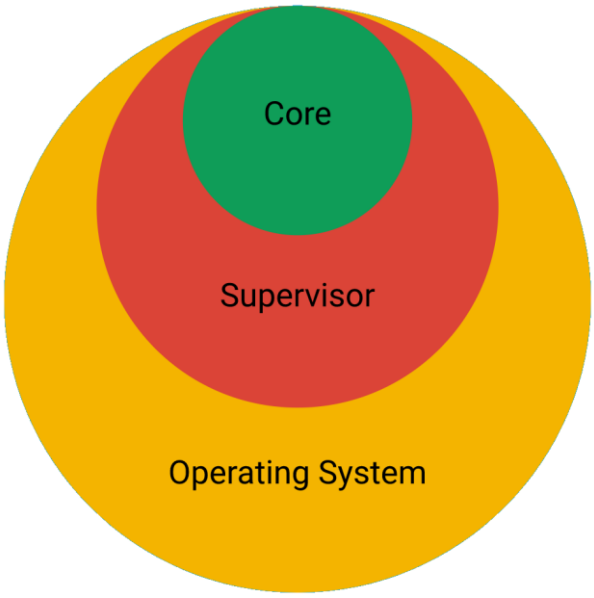

Figura 10. Arquitectura de Home Assistant

Como se observa, la capa más externa de Home Assistant pertenece al sistema operativo, el cual proporciona un entorno Linux mínimo para ejecutar el supervisor y el núcleo. Las capas más exteriores dependen de la capa inmediatamente inferior, por lo que el comportamiento de la aplicación está definido en el núcleo.

El **núcleo** interactúa con la persona usuaria, el supervisor y los dispositivos y servicios IoT. Contiene numerosas clases de apoyo para tratar con escenarios comunes, como proporcionar una entidad o controlar la ubicación. A su vez, se compone de cuatro partes fundamentales:

- **Máquina de estados** (*State Machine*), la cual hace un seguimiento del estado de los componentes y dispara un evento de cambio de estado cuando esto sucede. Realiza comprobaciones continuamente para llevar una traza de cada uno de los estados en el tiempo.
- **Bus de eventos** (*Event Bus*), que facilita el disparo y la escucha de eventos. Actúa como intermediario entre la máquina de estados y el registro de servicios.

Gracias a éste las automatizaciones son posibles en la plataforma, lo que la dota de un comportamiento reactivo.

- **Temporizador** (*Timer*), cuya función es enviar un evento de cambio de tiempo cada segundo al bus de eventos. De esta manera, el sistema proporciona un control sincronizado del tiempo.
- **Registro de servicios** (*Service Registry*), el cual escucha los eventos de llamada a servicio provenientes del bus de eventos y permite otro código para registrar servicios.

El esquema de la figura 11 señala gráficamente las partes anteriormente mencionadas.

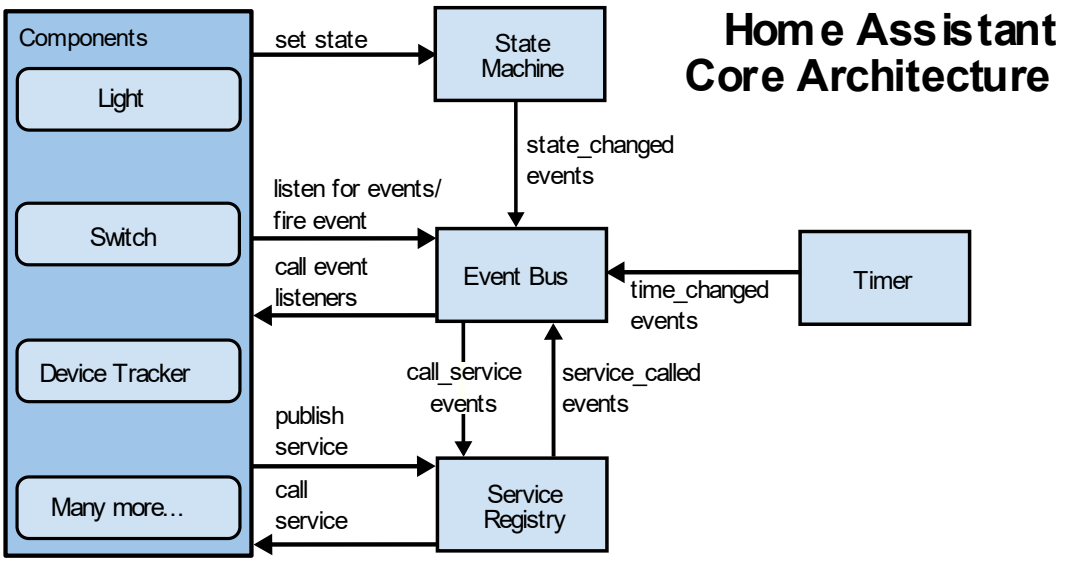

Figura 11. Arquitectura del núcleo de Home Assistant

Tal y como se muestra, existe una serie de componentes (como pueden ser luces, interruptores y servicios) que contienen un estado interno y una serie de eventos programados. Cuando el estado de un componente cambia, se informa a la máquina de estados, y cuando se produce un evento, éste se dispara hacia el bus de eventos. Si el componente es un servicio se publica en el registro de servicios.

Por otro lado se encuentra el **supervisor**, que se encarga de administrar el sistema operativo y la instalación de Home Assistant. Las responsabilidades de esta capa se comprenden entre ejecutar y actualizar el núcleo, crear y restaurar copias de seguridad, administrar los *add-ons* y unificar el audio del sistema. Se apoya en un DNS (*Domain Name System*) para permitir la comunicación entre el núcleo y los *add-ons* y en un mDNS (*multicast* DNS) para descubrir y conectar dispositivos y servicios en la red.

5. Tecnología utilizada

En este capítulo se va a explicar en detalle qué *hardware* y *software* se ha utilizado para llevar a cabo el desarrollo de este trabajo, incluyendo sus principales características y protocolos utilizados.

5.1 Hardware

El proyecto está desarrollado en el SBC Raspberry Pi, aunque para realizar las primeras pruebas se ha utilizado un ordenador con el sistema operativo Windows 10, un procesador Intel Core i9 9900K y una memoria RAM de 16 GB. A continuación se detallará cada componente *hardware* con el objetivo de comprender su papel en este trabajo.

5.1.1 Raspberry Pi 3 Model B

Figura 12. Raspberry Pi 3 Model B

La Raspberry Pi [22] es un mini ordenador de bajo coste que puede ser conectado a un monitor o televisión y utiliza un teclado y ratón estándar, el cual se muestra en la figura 12. Se categoriza en una serie de SBCs desarrollados por la Raspberry Pi Foundation, con el objetivo de permitir a los usuarios explorar áreas en la computación, ya que es capaz de realizar las mismas tareas que un ordenador de escritorio. Es ampliamente utilizado en proyectos de robótica.

Las especificaciones del modelo 3B se componen por un procesador Broadcom BCM2837 *quad-core* de 64 bits a 1,2 GHz, una memoria RAM de 1 GB LPDDR2 y una conectividad Wi-Fi de banda dual a 2,4 GHz y 5 GHz y Bluetooth 4.2. La GPU utilizada es una Dual Core VideoCore IV, la cual proporciona Open GL ES 2.0, OpenVG acelerado por *hardware* y 1080p30 H.264 de alto perfil de decodificación.

Dispone de 4 puertos USB 2.0, 1 puerto Ethernet, 1 puerto HDMI 1.4, 1 jack de 3,5 mm de salida de audio, una ranura micro SD y un encabezado GPIO de 40 pines. La potencia de entrada es de 5V DC, que puede ser proporcionada a través del conector micro USB o del encabezado GPIO.

Su sistema operativo oficial es una versión adaptada de Debian llamada Raspberry Pi OS, teniendo la posibilidad de utilizar otros sistemas operativos, incluyendo una versión de Windows 10.

5.1.2 Arduino UNO REV3

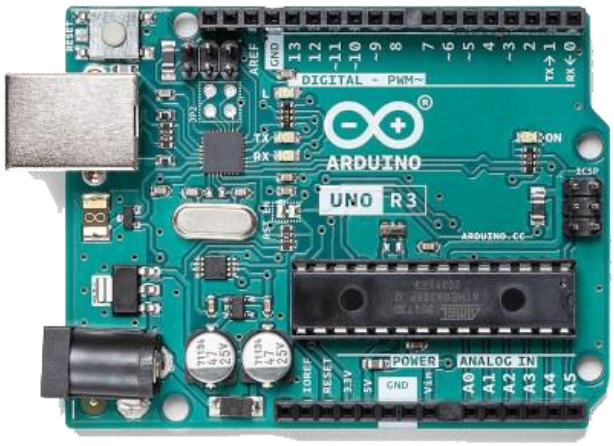

Figura 13. Arduino UNO REV3

Arduino UNO [1] es un microcontrolador basado en el microchip ATmega328P y desarrollado por Arduino, el cual se muestra en la figura 13. Está equipado por un conjunto de salidas y entradas analógicas y digitales que pueden ser conectadas a varias tarjetas de expansión y otros circuitos. El modelo REV3 de Arduino es el primero en disponer de un puerto USB y es una de las placas más utilizadas y con mayor documentación para desarrollo de proyectos, siendo excelente para aprender conceptos básicos sobre el funcionamiento de sensores y actuadores.

La placa contiene 14 pines digitales, de los cuales 6 de ellos tienen salida PWM, 6 pines analógicos, un cristal de cuarzo de 16 MHz, un encabezado ICSP (*In Circuit*

Serial Programming) y un botón de reinicio. Puede ser alimentado por un cable USB o una batería externa de 9 voltios (aunque acepta voltajes de entre 7 y 20 voltios).

Es programable con el denominado Arduino IDE (*Integrated Development Environment*) mediante un cable USB de tipo B. Este entorno permite desarrollar todo tipo de proyectos en los que intervengan otros componentes eléctricos.

5.1.3 Velleman VMA314

Figura 14. Velleman VMA314

Velleman VMA314 [15] es un detector de movimiento PIR genérico, el cual se muestra en la figura 14. Es un sensor electrónico que mide la luz infrarroja (IR) radiada de los objetos que se encuentran en su campo de acción. Una de las aplicaciones principales es su uso en alarmas de seguridad.

En cuanto a la conexión, tiene 1 pin de VCC, 1 pin de GND y 1 pin de salida, que es el que se activará si existe una detección de movimiento. Tiene una temporización de activación de 3 a 200 segundos, una distancia máxima del sensor de 7 metros y un ángulo de detección de 120 grados. Es posible ajustar la sensibilidad y temporización con el potenciómetro de ajuste.

5.1.4 EZVIZ C6N

Figura 15. EZVIZ C6N

La EZVIZ C6N [16] es uno de los modelos de cámara IP de EZVIZ, el cual se muestra en la figura 15. Este modelo permite acceder al *streaming* de la cámara con una referencia a su dirección IP, de forma inalámbrica y siempre que se encuentre en la misma red (o mediante un cable Ethernet). Incluye una función IR inteligente que usa luz infrarroja avanzada para capturar detalles con una baja luminosidad. Su campo de visión es de 360 grados, por lo que se puede categorizar como una cámara domo PTZ. Además, dispone de una función de seguimiento inteligente.

Dispone de una ranura micro SD para almacenar las grabaciones de manera flexible y segura. Existe un servicio en la nube, llamado EZVIZ Cloud, que es de pago y permite guardar las imágenes en su plataforma. Este modelo puede ser controlado mediante la aplicación móvil EZVIZ Studio, permitiendo cambiar los distintos ajustes de la cámara y controlarla en tiempo real.

5.2 Software

En los siguientes apartados se van a explicar los principales componentes *software* utilizados y qué funciones principales tiene cada uno, así como los protocolos o lenguajes de marcado de relevancia.

5.2.1 Home Assistant Lovelace UI

Figura 16. Home Assistant Lovelace UI

Lovelace [4] es el *dashboard* o tablero de Home Assistant, el cual se muestra en la figura 16. Es rápido, personalizable y una manera potente para que la persona usuaria controle su hogar desde su escritorio o el propio móvil. Permite configurar los iconos, la temática y los elementos que se muestran en la interfaz mediante código YAML. Estas son sus principales características:

- 29 tarjetas diferentes para colocar y configurar a gusto de la persona usuaria. Éstas contienen información comúnmente utilizada, como un panel de alarma, un botón, un calendario o una entidad. Esto incluye la posibilidad de mostrar el valor de un sensor en tiempo real.
- Tarjetas personalizadas además de las ya integradas, gracias a una API para crear elementos personalizados. Además, la comunidad puede compartirlas, lo que puede ser de ayuda para un gran número de usuarios.
- Un editor de *dashboard* que permite administrar la interfaz Lovelace mediante una vista previa al editar las tarjetas.
- Temas personalizados, incluyendo para cada tarjeta individual.
- Posibilidad de sobreescribir los nombres e iconos de las entidades.

5.2.2 Telegram

Telegram [17] es una aplicación de mensajería libre y de código abierto, enfocada en la velocidad y seguridad, diseñada para ser muy rápida y de uso gratuito. Es una de las aplicaciones con mayor carga de usuarios y permite sincronizar los mensajes en múltiples dispositivos. Los mensajes se encuentran en la nube y este servicio proporciona videollamadas con cifrado de extremo a extremo, VoIP y compartición de archivos, entre otras características. Se encuentra disponible tanto para iOS como para Android.

Ofrece la posibilidad de crear canales, que es un método de mensajería unidireccional en el que únicamente los administradores pueden publicar mensajes. Por otro lado, los desarrolladores pueden crear ***bots***, que son cuentas de Telegram operadas por programas. Pueden responder a mensajes o menciones, pueden ser invitados a grupos o canales y pueden ser integrados en otros programas. Además, tienen la capacidad de aceptar pagos en línea con tarjetas de crédito.

Esta plataforma es ampliamente utilizada para proyectos IoT, gracias al envío de notificaciones de manera automática mediante los *bots*. Para crear un *bot*, dispone de una API basada en una interfaz HTTP. De esta manera, es posible interactuar con el *bot* mediante órdenes.

5.2.3 MQTT

MQTT [2] es un protocolo estándar para la conectividad IoT que funciona sobre TCP/IP. Este protocolo permite publicar mensajes y suscribirse a tópicos, es simple y ligero, y está diseñado para dispositivos con restricciones de ancho de banda y redes de alta latencia. Los principios de diseño son minimizar el uso de la red y los requisitos de recursos de los dispositivos mientras se asegura la fiabilidad y cierto grado de garantía en el envío. Estos principios convierten a este protocolo en el ideal para interactuar

Su estructura normalmente comprende el uso de un MQTT *broker*. El *broker* actúa como un intermediario, el cual se encargará de recibir mensajes de uno o más publicadores y de enviarlos a uno o más suscriptores. Cada cliente puede producir y recibir datos mediante la publicación y la suscripción (es bidireccional), por lo que normalmente este patrón se utiliza para publicar información de los sensores, que posteriormente será procesada por los suscriptores para controlar otros dispositivos, entre otras muchas posibilidades.

Cada una de las conexiones establecidas con el *broker* puede especificar una medida de QoS (*Quality of Service*). Esto permite especificar cómo se va a tratar el envío de

mensajes y si se va a garantizar su recepción, o si por el contrario serán enviados sin una confirmación.

5.2.4 YAML

YAML (YAML *Ain't Markup Language*) [18] es un lenguaje de serialización de datos diseñado para ser legible por humanos. Es comúnmente utilizado en archivos de configuración y aplicaciones donde se almacenan o transmiten datos. Es muy similar al lenguaje XML, ya que se usa para el mismo objetivo que éste. La sangría utilizada es al estilo de Python, utilizando también los corchetes para denotar listas y las llaves para diccionarios. El objetivo de este lenguaje radica más en el procesamiento de datos que en el marcado de documentos.

Muchos lenguajes de programación son capaces de leer y escribir YAML, como C/C++, C#, Java, Javascript, Perl, PHP, Python o Ruby. Algunos editores de código fuente y varios IDE cuentan con integración para editar en este lenguaje de manera más sencilla.

Los contenidos en YAML se describen utilizando carácteres Unicode (UTF-8 o UTF-16) y la estructura del documento se denota solamente utilizando espacios en blanco como sangrado, por lo que no se admite la tabulación. De forma nativa, YAML codifica variables (como cadenas, enteros y coma flotante), listas y diccionarios, los cuales están basados en el lenguaje de programación Perl, aunque también admite tipos de datos personalizados.

Home Assistant usa este lenguaje para describir sus propios archivos de configuración y para crear las automatizaciones, así como para definir todas las entidades que serán utilizadas por esta plataforma.

5.2.5 SQLite

SQLite [19] es una biblioteca que implementa un motor de base de datos SQL independiente, transaccional, sin servidor y sin configuración. El código se encuentra en el dominio público y es de libre uso.

SQLite es un motor de base de datos embebido, por lo que no es ejecutada por un servicio separado. Este motor escribe y lee directamente archivos en disco, por lo que las bases de datos, incluyendo las múltiples tablas que las componen, están contenidas en un único archivo. Además, el formato de la base de datos supone que es intercambiable entre sistemas de 32 y 64 bits. Es una biblioteca muy compacta, siendo su tamaño de tan solo 600 KB con todas sus características activadas.

6. Desarrollo

6.1 Primeros pasos en Arduino IDE

El primer paso a realizar será descargar el IDE de Arduino desde la página web oficial, el cual nos permitirá crear programas para que el microcontrolador Arduino UNO funcione de la manera deseada. La versión elegida para la descarga será la Arduino 1.8.15.

Una vez descargado e instalado correctamente, se abrirá la ventana mostrada en la figura 17 al arrancar la aplicación.

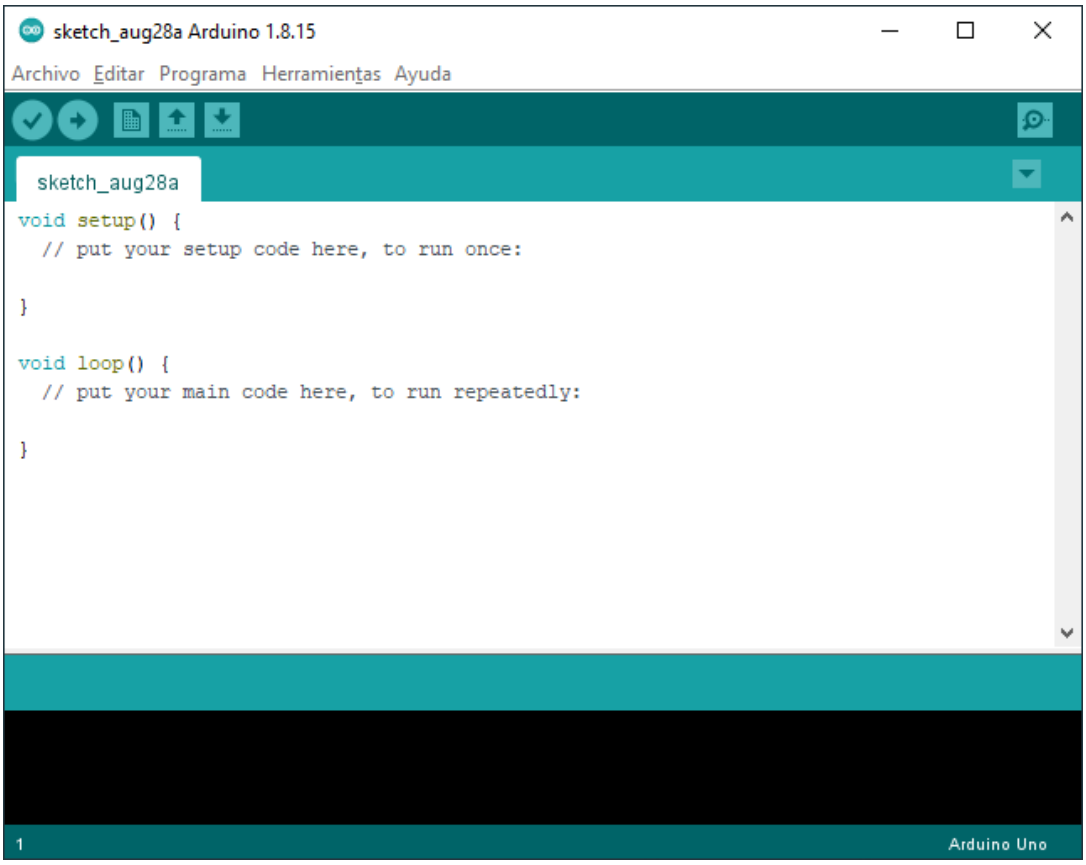

Figura 17. Ventana principal de Arduino IDE

Todos los programas de Arduino parten de dos funciones que deberán estar siempre presentes. En primer lugar, la función ***setup*** se ejecuta solamente una vez en el arranque del programa. Se utiliza normalmente para establecer qué pines del microcontrolador actuarán como entradas y como salidas, así como para iniciar la comunicación con el ordenador mediante el puerto serie, entre otras muchas posibilidades. En general, aquí se ubicarán todas aquellas sentencias para poner a

punto el programa actual. Por otro lado, se encuentra la función ***loop***, que se ejecutará en un bucle infinito hasta detener por completo el microcontrolador. Aquí es donde se llevarán a cabo las acciones pertinentes según el comportamiento deseado, pudiendo leer entradas, escribir salidas y ejecutar otras funciones auxiliares.

En nuestro programa van a intervenir varios componentes eléctricos, entre los que se encuentra el sensor PIR, un LED y un zumbador. Es una buena práctica definir las entradas y salidas al comienzo del archivo como constantes, de forma que es más legible, y si hubiera algún cambio sólo habría que reemplazarlo en un único lugar.

Las variables globales serán las que se muestran en la figura 18, en las que se observa que el sensor PIR ocupará el pin 2, el LED ocupará el pin 4, el zumbador ocupará el pin 6 y se establece el estado del sensor PIR a nivel bajo por defecto.

```
const int inputPir = 2;
const int outputLed = 4;
const int outputBuzzer = 6;
int pirState = LOW;
```

Figura 18. Constantes del programa de Arduino

Cuando hemos definido las constantes, el siguiente paso será definir el cuerpo de la función *setup*. Aquí se denotarán qué pines del microcontrolador actuarán como entradas y cuáles como salidas. Gracias a las constantes definidas previamente el código presentado en la figura 19 resulta mucho más legible, y se observa que el sensor PIR (pin 2) actuará como entrada, el LED (pin 4) actuará como salida, el zumbador (pin 6) también actuará como salida y se iniciará la comunicación por puerto serie a 9600 baudios.

```
void setup() {
  pinMode(inputPir, INPUT);
  pinMode(outputLed, OUTPUT);
  pinMode(outputBuzzer, OUTPUT);

  Serial.begin(9600);
}
```

Figura 19. Cuerpo de la función setup en Arduino

El objetivo que queremos lograr con nuestro programa es leer la entrada asociada al sensor de movimiento, y en el momento que la entrada se encuentre a nivel alto, enviaremos la cadena **"Sensor ON"** por el puerto serie, encenderemos el LED y activaremos el zumbador. Cuando la entrada del sensor se encuentre a nivel bajo, se apagará el LED y se desactivará el zumbador. En la figura 20 se muestra el código anterior en la función *loop*, que se ejecutará de forma indefinida.

```
void loop(){
  int value = digitalRead(inputPir);

  if (value == HIGH && pirState != value) {
    digitalWrite(outputLed, value);
    Serial.println("Sensor ON");
    tone(outputBuzzer, 1000);
    delay(1000);
    noTone(outputBuzzer);

    pirState = value;
  } else if (value == LOW && pirState != value) {
    digitalWrite(outputLed, value);
    Serial.println("Sensor OFF");

    pirState = value;
  }
}
```

Figura 20. Cuerpo de la función loop en Arduino

Tal y como se observa en la figura 20, se añade un retardo de 1 segundo entre la función ***tone*** y ***noTone***, pertenecientes a la activación del zumbador. Esto es debido a que debe dejarse un espacio de tiempo para que el oído humano aprecie el sonido de éste dispositivo, para posteriormente desactivarse. El argumento numérico de la función *tone* indica la frecuencia y es posible modificar el sonido si se cambia este parámetro. Otro dato importante es que se envía la cadena "Sensor OFF" por el puerto serie cuando la entrada del sensor se encuentra a nivel bajo, para así poder diferenciar cuándo ha habido una detección de movimiento en tiempo real.

La cadena enviada por el puerto serie será procesada posteriormente por Home Assistant, quien se encargará de manejar esta serie de eventos y de mostrar en la interfaz el estado del sensor.

6.2 Pruebas en Arduino IDE

El siguiente paso a realizar cuando el código esté terminado será verificar y compilar el programa. La verificación comprueba que no haya errores de sintaxis y la compilación convierte el programa a un fichero binario que se carga en la *flash* del MCU (*Microcontroller Unit*) y que será ejecutado. Para ello, nos debemos dirigir al menú y pulsar en la opción **Verificar/Compilar** (Ctrl + R) bajo la pestaña Programa.

Si todo ha ido correctamente, el programa comenzará a ejecutarse indefinidamente en Arduino UNO. Comprobará periódicamente el estado del sensor PIR, de modo que si se encuentra a nivel alto se visualizará la cadena "Sensor ON" en el puerto serie asociado al microcontrolador, o la cadena "Sensor OFF" si se encuentra a nivel bajo. Si deseamos verificar que se comporta de la manera deseada disponemos de la opción **Monitor Serie** (Ctrl + Mayús + R), situada bajo la pestaña Herramientas (figura 21).

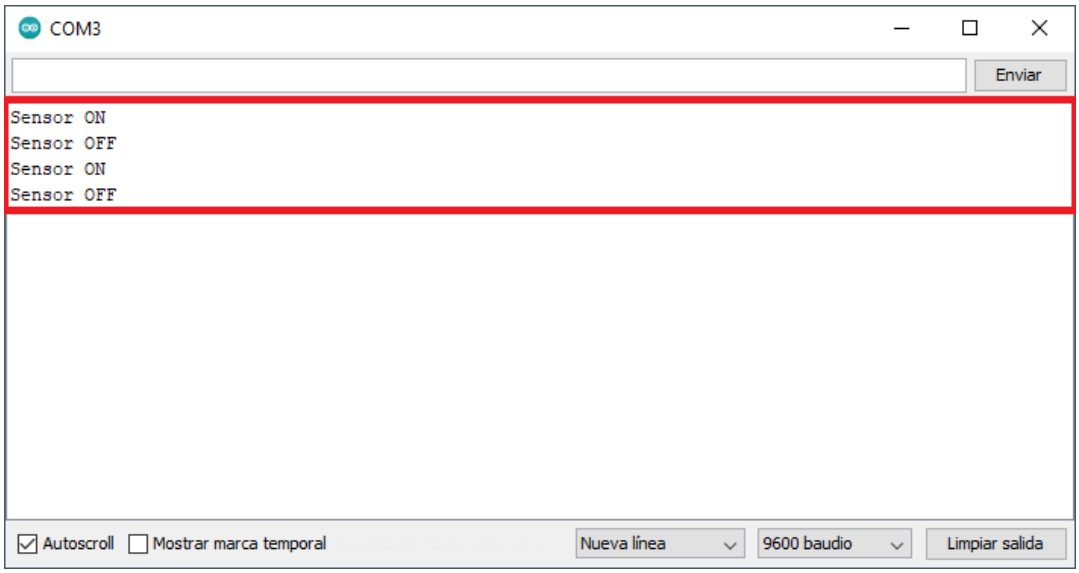

Figura 21. Monitor serie de Arduino

En este caso, el puerto serie asociado a Arduino UNO es el COM3, por lo que deberemos seleccionar el puerto correcto en la ventana emergente en caso de no serlo. Cuando nos situamos frente al campo de acción del sensor se envía la cadena "Sensor ON", por lo que el funcionamiento esperado es correcto, y cuando nos retiramos de éste vuelve a enviar "Sensor OFF". Si no apareciera ninguna cadena, deberíamos asegurarnos de que la velocidad de comunicación del puerto está establecida en los 9600 baudios. Repetimos este proceso de nuevo para verificar que el comportamiento es el esperado y que las pruebas son satisfactorias.

6.3 Instalación de Raspbian en Raspberry Pi

Para poder continuar es necesario disponer de una Raspberry Pi y de una tarjeta micro SD, la cual contendrá el sistema operativo de ésta. Una memoria de 32 GB es suficiente para contener los archivos necesarios. El sistema operativo elegido será Raspbian (también llamado Raspberry Pi OS) [26], una distribución de GNU/Linux basada en Debian optimizada para su *hardware*. Es posible encontrar imágenes de este sistema operativo en la página web oficial de Raspberry Pi.

Con los materiales necesarios ya preparados, emplearemos el software denominado **balenaEtcher** [27], que se encargará del proceso de instalación de la imagen en la tarjeta micro SD. Este proceso puede demorarse unos minutos hasta que esté completo.

6.4 Instalación de Home Assistant

Cuando el sistema operativo Raspbian ya está instalado, deberemos instalar Home Assistant, el controlador domótico que manejará los eventos producidos por el sensor de movimiento y con el que podremos crear automatizaciones [4].

Por razones de comodidad, es posible visualizar la pantalla de la Raspberry Pi directamente en nuestro ordenador, con la ayuda de un *software* de control por acceso remoto. En este caso utilizaremos xRDP para proyectar la pantalla del mini ordenador, y también será necesario que ambos equipos se encuentren en la misma red inalámbrica (o mediante cable Ethernet). Para ello, escribiremos las siguientes órdenes en una terminal:

```
$ sudo apt update
$ sudo apt install xrdp
```

Cuando se haya instalado, el servicio se ejecutará automáticamente, por lo que no será necesario iniciarlo manualmente. El siguiente paso será instalar las dependencias para el correcto funcionamiento de Home Assistant, previa comprobación de que el sistema está completamente actualizado:

```
$ sudo apt update
$ sudo apt upgrade -y
$ sudo apt install -y python3 python3-dev python3-venv python3-pip
libffi-dev libssl-dev libjpeg-dev zlib1g-dev autoconf build-
essential libopenjp2-7 libtiff5 tzdata
```

Una vez estén instaladas todas las dependencias, es importante crear un usuario llamado ***homeassistant***. Esta cuenta servirá solo para ejecutar Home Assistant. El argumento -rm se añade para crear una cuenta de sistema y un directorio de inicio, y el argumento -G añade el usuario a los grupos indicados, que son requeridos para que éste pueda comunicar con GPIO y usar los controladores Z-Wave y Zigbee:

```
$ sudo useradd -rm homeassistant -G dialout,gpio,i2c
```

Ahora deberemos crear un directorio para Home Assistant usando la cuenta de usuario creada en el paso anterior. Esta cuenta será la propietaria de este directorio, por lo que habrá que cambiarle los permisos:

```
$ sudo mkdir /srv/homeassistant
$ sudo chown homeassistant:homeassistant /srv/homeassistant
```

Lo próximo será crear un entorno virtual para Home Assistant, utilizando nuevamente el usuario *homeassistant*. Este entorno será creado mediante una orden de Python:

```
$ sudo -u homeassistant -H -s
$ cd /srv/homeassistant
$ python3.8 -m venv .
$ source bin/activate
```

Cuando hayamos activado el entorno virtual (este paso debe realizarse siempre en cada arranque), deberemos instalar un paquete Python requerido y el propio Home Assistant con las siguientes órdenes:

```
$ python3 -m pip install wheel
$ pip3 install homeassistant
```

Con todos los pasos anteriores, ya estamos listos para ejecutar Home Assistant:

```
$ hass
```

Esto abrirá el puerto 8123 y podremos acceder a la interfaz web del controlador domótico mediante la URL http://localhost:8123, siendo *localhost* una referencia a la dirección IP de la propia máquina.

6.5 Primeros pasos en Home Assistant

Una vez completado el proceso de instalación de Home Assistant, accedemos a nuestra dirección IP local en nuestro navegador, usando para ello el puerto 8123. Esto abrirá un panel de registro para crear nuestra primera cuenta, tal y como se muestra en la figura 22:

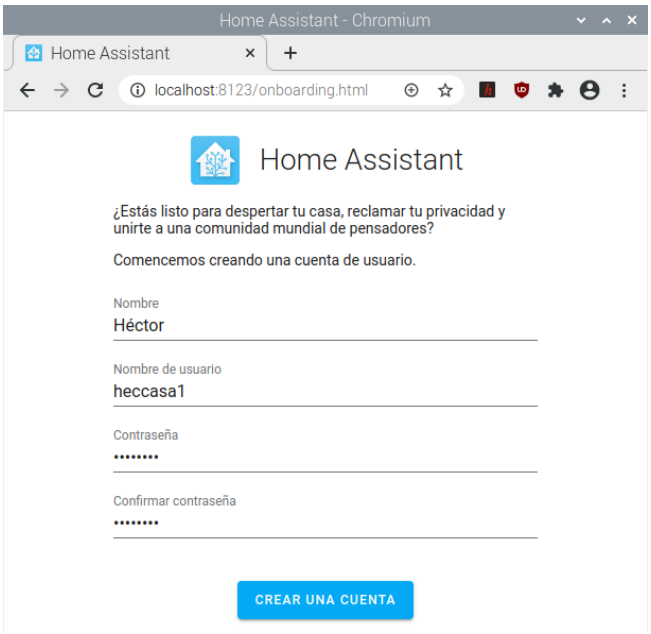

Figura 22. Creación de una cuenta en Home Assistant

A la cuenta que acabamos de crear se le otorgarán permisos de administrador y no podrá eliminarse desde la interfaz, por lo que habría que crear un nuevo usuario si se desea reemplazar el ya existente. Con la nueva cuenta creada, nos aparecerá una ventana para elegir el nombre de la instalación y la zona horaria actual (figura 23).

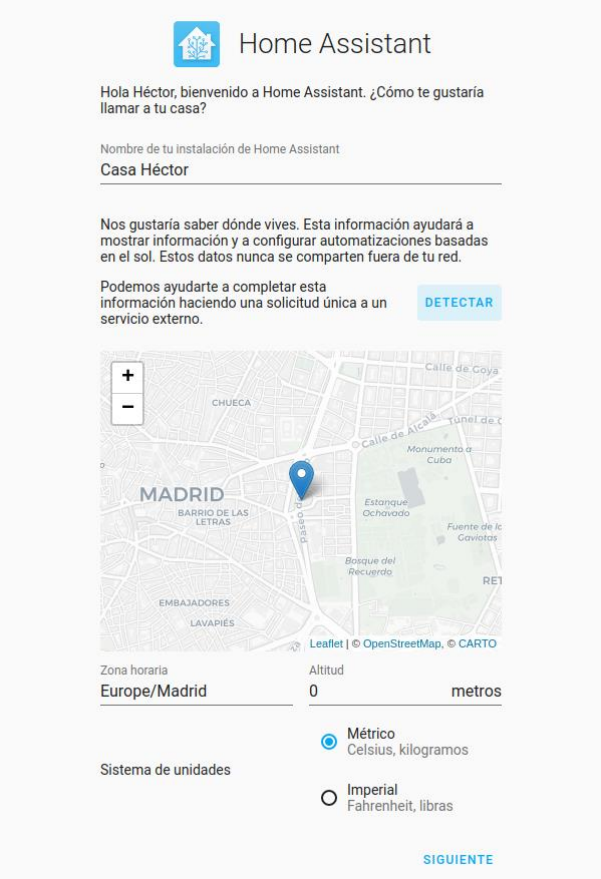

Figura 23. Creación de una casa en Home Assistant

Cuando hayamos finalizado el proceso de bienvenida se abrirá por primera vez la interfaz Lovelace. En primer lugar estará completamente vacío, ya que no tenemos ningún dispositivo configurado hasta el momento. La estructura principal es la siguiente:

- En la pestaña **Resumen** se podrán añadir tarjetas, que contendrán datos importantes como el valor actual de los sensores o cualquier otro dispositivo conectado.
- La pestaña **Mapa** se muestra la ubicación actual de nuestra instalación mediante OpenStreetMap [28], un proyecto libre de creación de mapas editables.
- La pestaña **Registro** muestra un *log* del cambio de valor de nuestros dispositivos, entre otros datos relevantes.
- En la pestaña **Historial** se muestra en un diagrama cronológico de barras el valor de nuestros dispositivos en determinadas franjas de tiempo, así como el estado de activación de las automatizaciones.
- La pestaña **Herramientas para desarrolladores** muestra el estado de las entidades y automatizaciones, pudiendo alterarlo mediante la interfaz. Además, permite llamar a servicios directamente desde este panel, entre otras opciones.

- La pestaña **Configuración** permite administrar las integraciones instaladas, los dispositivos conectados, las entidades conocidas y las áreas en el hogar, así como las automatizaciones, las escenas, los *scripts* y los ayudantes (elementos que ayudan a construir automatizaciones). Desde aquí es posible cambiar la propia configuración general de Home Assistant, como también reiniciar y detener el servidor.

En la figura 24 se muestra el menú principal de la interfaz, el cual se encuentra vacío porque no hay ningún dispositivo conectado.

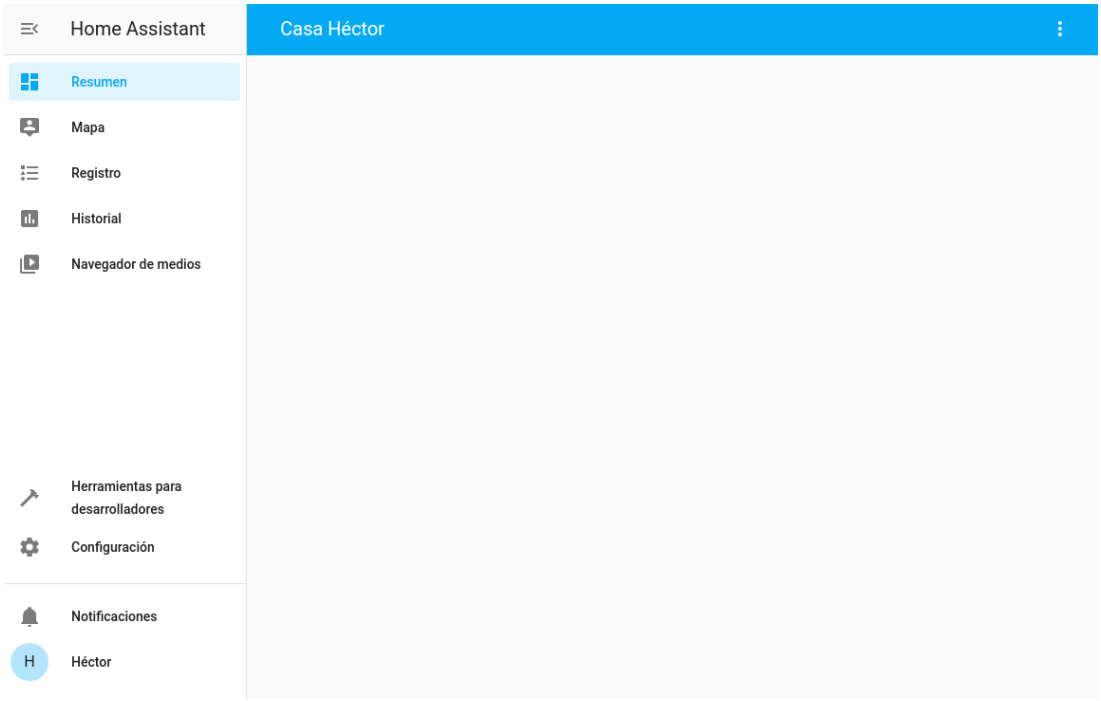

Figura 24. Menú principal de la interfaz Lovelace

Cada vez que agreguemos un nuevo dispositivo a Home Assistant, deberemos reiniciar la aplicación para que los cambios surtan efecto. Esto se debe a que los cambios producidos no se actualizan hasta que la aplicación se detiene y se inicia nuevamente.

Este proceso se hace mediante la opción **Controles del servidor**, bajo la pestaña Configuración, la cual permite controlar el estado del servidor directamente desde la interfaz y sin necesidad de detener la terminal. En la figura 25 se muestra gráficamente la ubicación de esta opción.

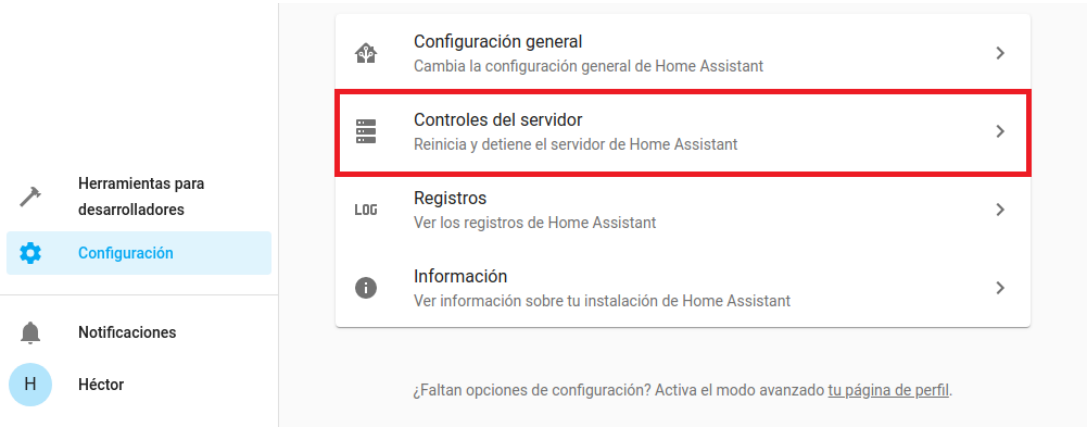

Figura 25. Controles del servidor de Home Assistant

Una vez accedemos al panel de administración del servidor, pulsamos el botón de reinicio (figura 26), de forma que en el próximo arranque se detectarán los nuevos dispositivos y automatizaciones añadidas.

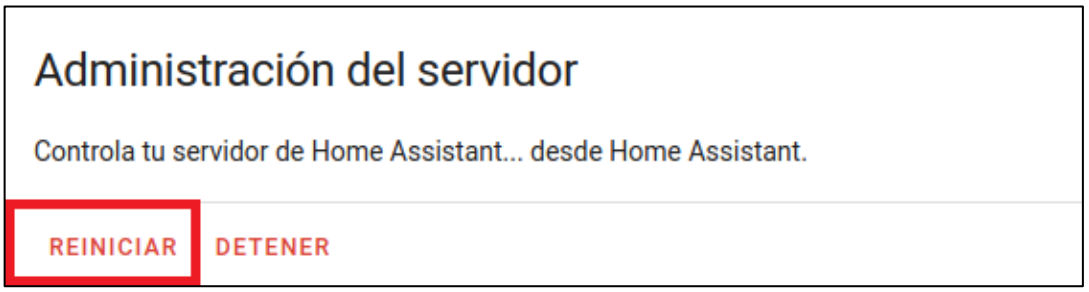

Figura 26. Panel de administración del servidor

6.6 Fase 1. Integración del sensor PIR mediante puerto serie

En este apartado vamos a integrar la implementación del sensor realizada en Arduino con Home Assistant, de modo que podamos visualizar el valor del sensor PIR en tiempo real y comprobar su correcto funcionamiento. Esta fase será importante a la hora de integrar las notificaciones de detección de movimiento en la aplicación de mensajería.

En primer lugar, es recomendable comprobar que el puerto serie se encuentra activado en la Raspberry Pi. Para ello, deberemos acceder al menú de inicio, seleccionar la pestaña Preferencias y pulsar en la opción de Configuración de Raspberry Pi. En la ventana emergente seleccionaremos la pestaña Interfaces y comprobaremos que la interfaz Serial Port se encuentra activa, como se muestra en la figura 27:

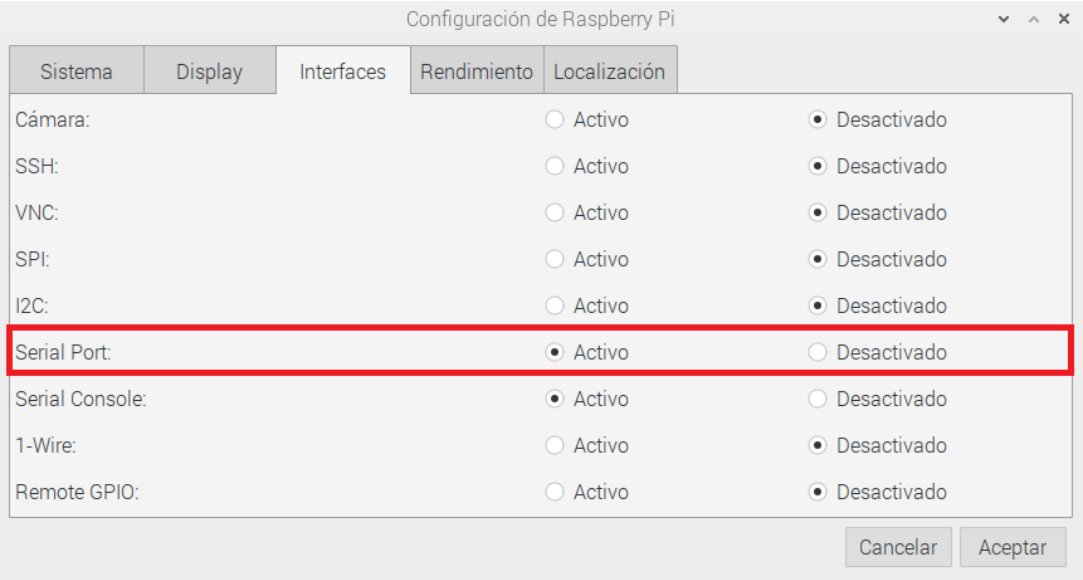

Figura 27. Activación del puerto serie en la Raspberry Pi

Después de estas comprobaciones, será necesario conectar el microcontrolador a la Raspberry Pi mediante el cable USB de tipo B (si no estaba conectado previamente). Esto dotará de alimentación a Arduino UNO para poder ejecutar el programa que le habíamos cargado anteriormente.

Para conocer en qué puerto serie está conectado el microcontrolador, consultamos para ello el directorio */dev* mediante la orden *ls*. Normalmente, el puerto asociado a la placa será ***/dev/ttyACM0***, pero en caso de duda es posible comparar y deducir qué puerto es añadido cuando establecemos la conexión por USB. Ése será el puerto que utilizaremos en la configuración de Home Assistant.

El paso más importante será crear la configuración para el sensor de movimiento en Home Assistant. La configuración se encuentra en el directorio */home/homeassistant/.homeassistant* bajo el nombre *configuration.yaml*. Este archivo contendrá todas las integraciones del controlador domótico con sus respectivos parámetros.

Existe una integración para el puerto serie denominada ***Serial***, la cual nos ayudará a establecer este tipo de conexión. Los parámetros necesarios para esta integración ya los hemos obtenido anteriormente, por lo que conocemos el puerto serie a utilizar. Por otro lado, utilizaremos una integración auxiliar llamada *Template*, que nos ayudará a mostrar un conjunto de datos u otros, dependiendo de la información recibida por el puerto serie especificado. En la figura 28 se muestra la implementación de ambas integraciones con sus respectivos parámetros.

```
sensor:
  - platform: serial
    baudrate: 9600
    serial_port: /dev/ttyACM0

  - platform: template
    sensors:
      sensor_dormitorio:
        friendly_name: "Sensor dormitorio"
        value_template: >
          {% if states.sensor.serial_sensor.state == 'Sensor ON' %}
            Presencia detectada
          {% elif states.sensor.serial_sensor.state == 'Sensor OFF' %}
            Sensor desactivado
          {% else %}
            Sensor desactivado
          {% endif %}
```

Figura 28. Configuración del sensor de puerto serie

La plataforma *serial* contiene la información relativa a la conexión mediante puerto serie. El puerto serie es */dev/ttyACM0* y la tasa de baudios se sitúa en 9600. Este último parámetro es opcional, ya que por defecto se establece en este valor. En cambio, la plataforma *template* actúa como plantilla, de modo que ejecuta un fragmento de código Python en el atributo ***value_template***. Cuando recibamos la cadena "Sensor ON" por el puerto serie, la interfaz mostrará "Presencia detectada", y cuando nos llegue la cadena "Sensor OFF", la interfaz mostrará "Sensor desactivado".

Cuando ya hemos completado la configuración, lo siguiente será crear una tarjeta en la interfaz Lovelace de Home Assistant. Esto nos permitirá comprobar en tiempo real que el funcionamiento es el esperado y podremos tener un mayor control de todos nuestros dispositivos en un mismo lugar. Para ello, nos dirigimos a la pestaña Resumen y presionamos la opción Editar panel de control en el icono de la esquina superior derecha, como aparece en la figura 29.

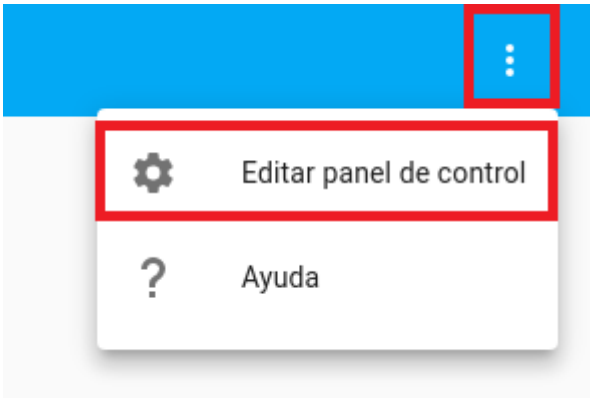

Figura 29. Editar panel de control en la interfaz Lovelace

Una vez entramos en el modo de edición, aparecerá un icono amarillo en forma de "+", el cual deberemos pulsar para añadir una nueva vista. En esta se almacenarán los dispositivos utilizados para este proyecto. Cuando pulsemos el botón, aparecerá la ventana mostrada en la figura 30, en la cual deberemos añadir un nombre para la vista a añadir, bajo la pestaña Configuración.

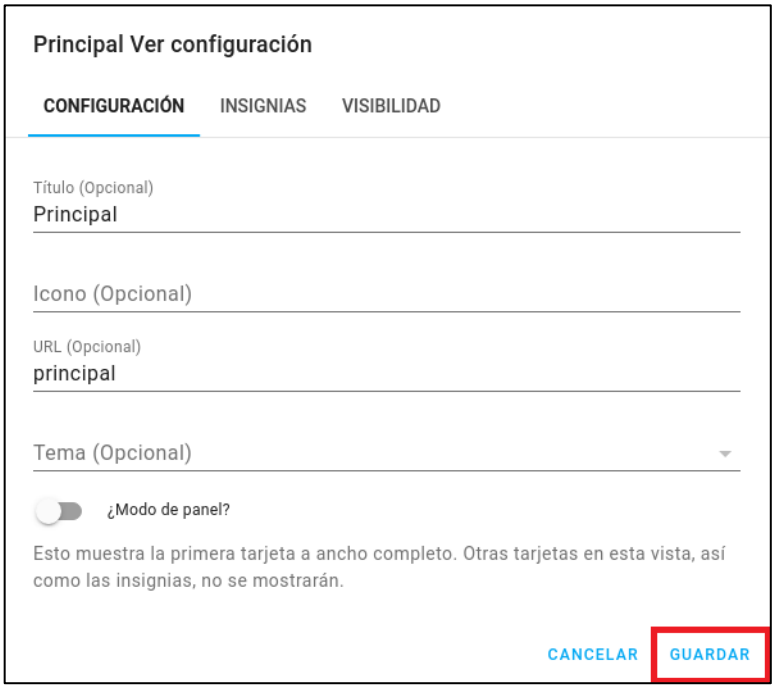

Figura 30. Creación de una vista en el panel de control

Con la vista ya creada, estamos listos para añadir la tarjeta asociada a la entidad del sensor PIR. Nos dirigimos de nuevo al icono de la esquina superior derecha y presionamos la opción **Entidades no utilizadas** (figura 31).

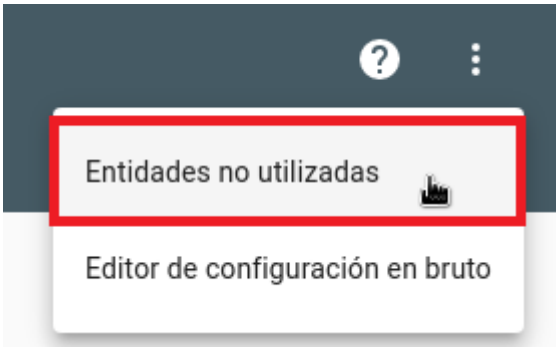

Figura 31. Menú de entidades no utilizadas

Aquí se mostrarán todas aquellas entidades registradas en el archivo *configuration.yaml* que todavía no han sido añadidas a la interfaz (figura 32). Es posible buscar directamente el sensor mediante el nombre indicado en el archivo, **sensor.sensor_dormitorio**, que actuará como identificador de la entidad:

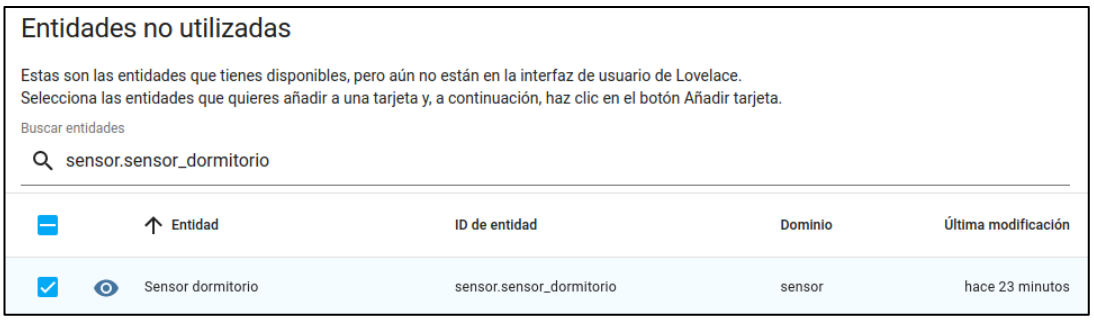

Figura 32. Adición del sensor de movimiento a la vista principal

Posteriormente a la adición del sensor a la interfaz Lovelace (aparecerá una ventana de confirmación), ya tendremos disponible la tarjeta para la visualización del valor del sensor. Inicialmente, el valor del sensor se establece en "Sensor desactivado", lo cual es correcto ya que no ha habido ninguna detección de movimiento por el momento, como se observa en la figura 33.

Figura 33. Tarjeta asociada al sensor PIR con sensor desactivado

A modo de prueba, nos situamos en el campo de acción del sensor y observamos que el valor en la tarjeta ha sido reemplazado por "Presencia detectada". Este es el comportamiento esperado, por lo que podemos deducir que las pruebas han sido satisfactorias. En la figura 34 se muestra el resultado final.

Figura 34. Tarjeta asociada al sensor PIR con presencia detectada

6.7 Fase 2. Integración de la mensajería MQTT

Con el objetivo de poder enviar notificaciones a la aplicación de mensajería, en este caso Telegram, necesitamos poder almacenar los mensajes a ser enviados. El motivo por el que se va a realizar de esta forma es porque es un método rápido y eficiente, y es la forma más sencilla de poder transmitir información a otro destino.

La información será almacenada en un tópico del sensor MQTT de Home Assistant a modo de *buffer*, ya que posteriormente vamos a interactuar con *scripts* en Python que consultarán la base de datos, y estos datos deberán ser almacenados para ser volcados en una notificación a Telegram. Este proceso no se puede llevar a cabo directamente desde el propio archivo YAML al operar con *scripts* externos, por lo que es necesario este sensor auxiliar.

El método será muy similar al realizado con el sensor de movimiento, por lo que deberemos modificar el archivo *configuration.yaml*, añadir la nueva integración y modificar los parámetros necesarios para el correcto funcionamiento. Esto se deberá realizar en dos pasos, teniendo que añadir en primera instancia el *broker* MQTT (figura 35).

```
mqtt:
  broker: 127.0.0.1
```

Figura 35. Integración MQTT broker

Recordemos que el *broker* actúa como intermediario entre publicadores y suscriptores en el protocolo MQTT, por lo que es un requisito fundamental para transmitir y recibir mensajes entre dos o más destinatarios. En este caso, nuestra propia máquina actuará como *broker*, ya que no es necesario un equipo externo para este proceso.

Una vez completado el paso anterior, procedemos a implementar la integración del sensor MQTT, que se muestra en la figura 36:

```
sensor:
  - platform: mqtt
    name: access
    state_topic: telegram/access/query
    value_template: '{{ value_json.type }}'
    json_attributes_topic: telegram/access/query
    json_attributes_template: '{{ value_json | tojson }}'
```

Figura 36. Integración sensor MQTT

Como se puede observar, debe especificarse un tópico en el atributo *state_topic*, que en este caso hemos nombrado como ***telegram/access/query***, en el que residirá la

información obtenida desde la base de datos y otras operaciones. Los demás atributos indican que la información del sensor será guardada en formato JSON a partir de la cadena guardada en el tópico anterior. Por lo tanto, una vez volcados los datos a este sensor, se convertirán a este formato y podrán ser referenciados mediante el nombre especificado a este sensor, que en este caso hemos denominado como ***access***.

6.8 Fase 3. Integración de la cámara IP de vigilancia

El próximo dispositivo que integraremos en Home Assistant será la cámara IP de vigilancia. Ésta se encargará de obtener las imágenes y las grabaciones de vídeo cuando haya una detección de movimiento por parte del sensor PIR, las cuales serán enviadas a la aplicación de mensajería para la visualización de la persona usuaria.

Para ello será necesario utilizar la integración *FFmpeg*, a la que especificaremos una entrada y un nombre, tal y como se muestra en la figura 37.

```
camera:
   - platform: ffmpeg
     name: Camara dormitorio
     input: rtsp://admin:BMFNXC@169.254.1.93:554/H.264
```

Figura 37. Integración FFmpeg

El atributo *input* es la entrada del *streaming* FFmpeg, el cual utiliza el protocolo RTSP (*Real Time Streaming Protocol*) [20] para la transferencia de datos. El usuario por defecto es *admin* y la contraseña es el código de verificación de la cámara. A continuación se especifica la dirección IP de la cámara y el puerto 554, que es estándar para este protocolo. El códec de vídeo utilizado es el H.264, el cual es de alta compresión y suministra imágenes de alta calidad sin consumir una gran cantidad de ancho de banda.

Añadiremos dos sensores adicionales, *FFmpeg Motion* y *FFmpeg Noise*, que complementarán la detección de movimiento. El primero de ellos captará movimiento y el otro detectará el ruido producido en el ambiente. Ambos tomarán como fuente el *streaming* de la cámara, por lo que se trata de un análisis en tiempo real. En la figura 38 se presenta la implementación de ambos sensores.

```
binary_sensor:
   - platform: ffmpeg_motion
     name: Movimiento dormitorio
     input: rtsp://admin:BMFNXC@169.254.1.93:554/H.264
     changes: 20
     reset: 30
   - platform: ffmpeg_noise
     name: Ruido dormitorio
     input: rtsp://admin:BMFNXC@169.254.1.93:554/H.264
     peak: -10
     reset: 30
```

Figura 38. Integraciones FFmpeg Motion y FFmpeg Noise

Tomando en cuenta el primer sensor de captura de movimiento, se incluyen dos parámetros adicionales. El parámetro *changes* indica el porcentaje de la imagen que deberá ser diferente entre un fotograma y el anterior para que se perciba movimiento, y la opción *reset* indica el tiempo en segundos para reiniciar el estado si no se detecta movimiento.

En cuanto al sensor de detección de ruido, también incluye dos parámetros importantes. El parámetro *peak* es el umbral de detección de ruido en decibelios (de 0 a -100, siendo 0 un valor muy alto y -100 un valor bajo), y la opción *reset* indica el tiempo en segundos para reiniciar el estado si el ruido percibido no supera el pico establecido.

Con todos los pasos realizados, sólo nos quedará añadir las tarjetas de los dos sensores anteriores a la interfaz Lovelace, para así tener un mayor control junto con el sensor PIR. Antes debemos asegurarnos de que la cámara está encendida y conectada en la misma red que la Raspberry Pi. El estado inicial de los sensores se muestra en la figura 39:

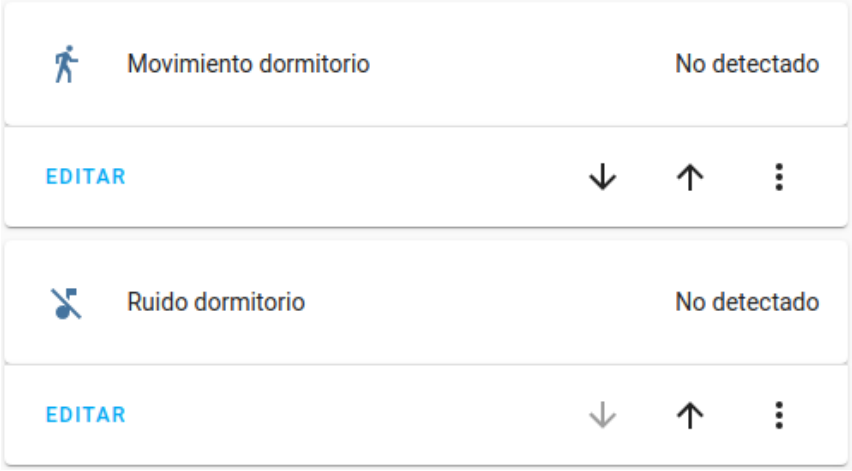

Figura 39. Tarjetas asociadas a los sensores de ruido y movimiento sin detección

Ahora nos situamos frente a la cámara y hacemos un ruido lo suficientemente fuerte para sobrepasar el umbral de detección de ruido. Esto causará que ambos sensores se disparen y capten el ruido y movimiento que hemos creado (figura 40):

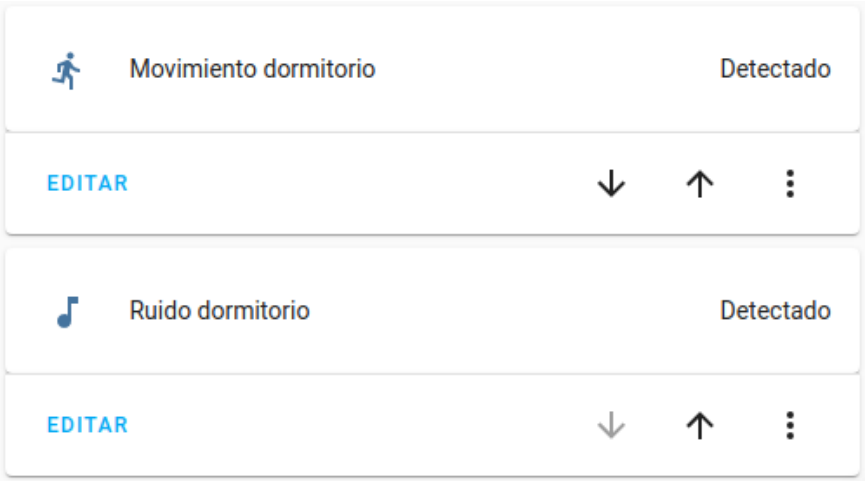

Figura 40. Tarjetas asociadas a los sensores de ruido y movimiento con detección

Con los resultados obtenidos y observando que el comportamiento es el esperado, podemos concluir que las pruebas han sido satisfactorias y que la cámara funciona correctamente, por lo que ya está lista para ser integrada con Telegram.

Para poder almacenar las imágenes y las grabaciones de vídeo, crearemos dos directorios, */config/photos* y */config/videos*, mediante las siguientes órdenes:

```
$ sudo mkdir /config/photos
$ sudo mkdir /config/videos
```

Por último, habrá que añadir los directorios que acabamos de crear en la lista blanca o *whitelist*, en el archivo de configuración. Esta lista le otorga permisos a Home Assistant para trabajar con directorios externos. En la figura 41 se muestra cómo añadir estos directorios.

```
homeassistant:
  allowlist_external_dirs:
    - /config/photos
    - /config/videos
```

Figura 41. Whitelist de directorios de Home Assistant

Como paso adicional, vamos a dotar a la cámara IP de capacidad para moverse en uno de los ejes desde la aplicación de mensajería. Esto sólo será posible en una cámara PTZ, que permite a una cámara moverse de izquierda a derecha y de arriba a abajo, con la opción de hacer *zoom* a cierta distancia.

Para esta tarea vamos a utilizar una adaptación del código de Pierre Ourdouille en GitHub, llamada **pyEzviz** [20]. Esto se debe a que el modelo concreto de cámara IP que vamos a utilizar posee su propia API REST para realizar esta función y por ello prescinde del protocolo estándar ONVIF [30].

En este caso solo vamos a necesitar elaborar una clase principal que contenga todos los argumentos necesarios para el uso de la biblioteca. A esta clase la llamaremos *main.py* en el directorio */home/homeassistant/.homeassistant/scripts/pyezviz*, junto con todos los demás archivos del repositorio, y será ejecutada por Home Assistant al recibir una orden desde la aplicación de mensajería con sus respectivos argumentos. El código completo de esta clase se proporciona en el código 1 en el anexo (*Programa de movimiento PTZ de una cámara IP EZVIZ*).

Los argumentos necesarios son el usuario de la cámara (-u), la contraseña (-p), la ubicación en el hogar del dispositivo (-l) y la dirección (-d), que indica el eje en el que se desplazará la cámara, siendo admitidos los valores *up*, *down*, *left* y *right*.

Será necesario operar con un fichero adicional, al que hemos llamado *camera_serial.json*, que contendrá un diccionario con los números de serie de las cámaras IP por ubicación del hogar. Se presupone el uso de una única cámara por estancia.

Cuando este *script* sea invocado mediante una automatización en Home Assistant, la cámara situada en la ubicación dada se desplazará en la dirección especificada mediante la API propia de este dispositivo, la cual requiere conexión a Internet para poder procesar la orden.

6.9 Fase 4. Integración de Telegram. Configuración del *bot*

La última integración que falta por añadir corresponde a la aplicación de mensajería Telegram. Desde esta aplicación será posible interactuar con nuestra instalación, bien recibiendo notificaciones de detección de movimiento o mediante órdenes que operarán con la base de datos.

En primer lugar, será necesario crear un *bot* en Telegram, el cual funcionará mediante una API en HTTP y podrá ser manejado desde las automatizaciones de Home Assistant. Este *bot* actuará como un usuario corriente que procesará todas las peticiones y recibirá todas las notificaciones provenientes del controlador domótico. Para ello, es imprescindible interactuar con **BotFather** [17], un *bot* cuya función es crear otros *bots* dentro de la plataforma.

Cuando hayamos encontrado a BotFather dentro de la plataforma, pulsaremos el botón Iniciar, ejecutando la orden */start* de inicio del *bot*. Esto mostrará un mensaje

de ayuda con todas las órdenes posibles que pueden ser procesadas por éste. En la figura 42 se enseña el resultado.

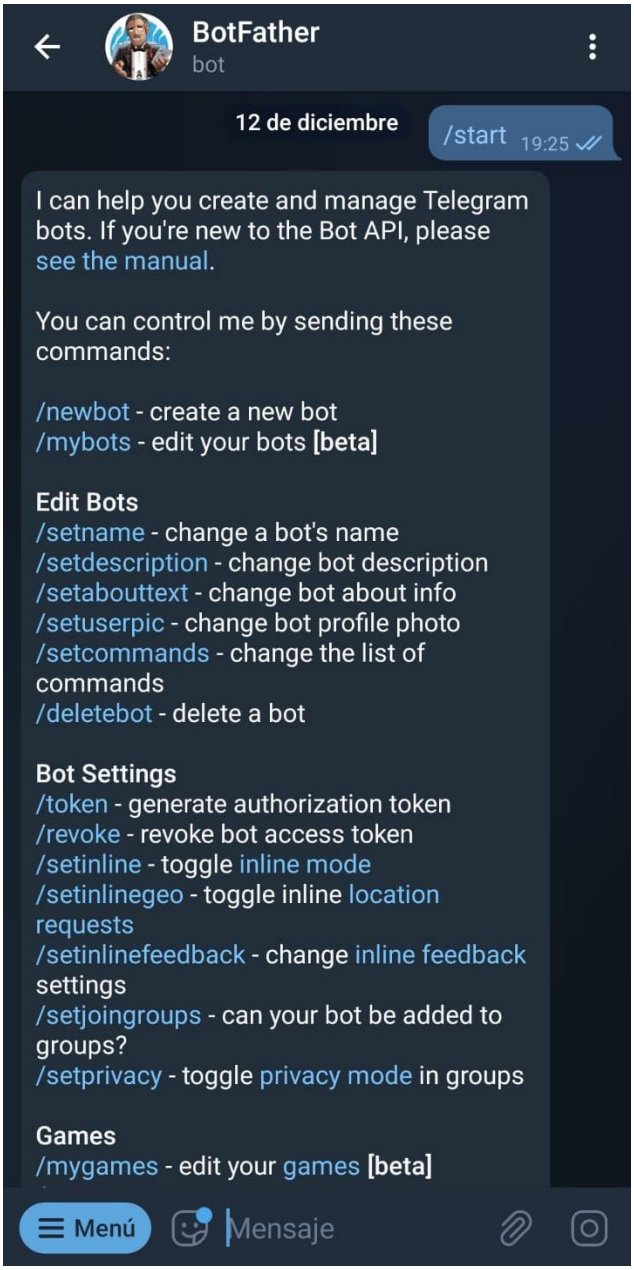

Figura 42. Inicio de BotFather

Lo próximo será crear nuestro propio ***bot***. El menú principal nos ofrece muchas opciones, pero la que nos interesa es */newbot*. Escribimos la orden y nos solicita ingresar un nombre y un nombre de usuario para éste. Si hemos proporcionado toda la información necesaria correctamente, nos devolverá el API *token*, el cual deberá guardarse y no ser revelado públicamente por temas de seguridad. En la figura 43 se muestra el proceso completo.

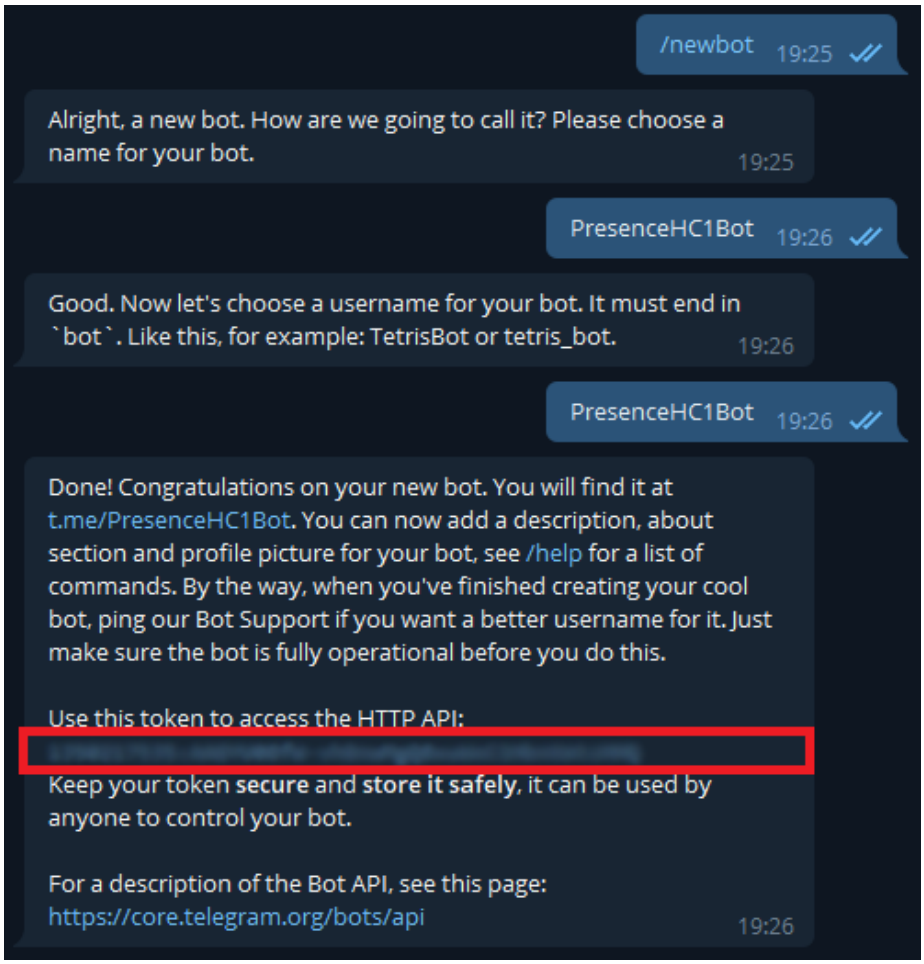

Figura 43. Creación de un bot en Telegram

El *token* que acabamos de obtener es el que nos permitirá interactuar con nuestro *bot* mediante la API de Telegram, al ser una cadena única que lo referencia. La configuración del nuevo *bot* debe ser añadida en el archivo *configuration.yaml* de Home Assistant. Deben añadirse las dos integraciones que aparecen en la figura 44:

```
telegram_bot:
    - platform: polling
      api_key: !secret api_token
      parse_mode: html
      allowed_chat_ids:
        - 42490470
```

```
notify:
    - platform: telegram
      name: Telegram
      api_key: !secret api_token
      chat_id: 42490470
```

Figura 44. Integración de Telegram

La primera integración, ***Telegram Bot***, contiene la configuración relativa al propio *bot* de Telegram. La plataforma especifica la implementación a utilizar para interactuar con el *bot*, denominada *polling*, la cual es más segura y no requiere que la instancia de Home Assistant esté expuesta a Internet. El atributo *parse_mode* indica que el texto enviado a Telegram será tratado como código HTML, por lo que las etiquetas de este lenguaje serán interpretadas en el formato de texto.

La integración ***Notify*** se encargará de enviar las notificaciones a la plataforma elegida, en este caso Telegram. El atributo *chat_id* indica el chat donde se enviarán las notificaciones generadas. Se utilizará la propia conversación con el *bot* como destino, por lo que sólo nosotros las recibiremos de forma privada, aunque existe la opción de ser dirigidas a un canal de Telegram dedicado para el hogar si así se desea, con el objetivo de alcanzar a más usuarios

Nótese que ambas integraciones comparten el atributo ***api_key***. Este es el *token* que hemos obtenido al finalizar la creación de nuestro *bot*. Esta cadena es referenciada desde el archivo *secrets.yaml* mediante la palabra clave *!secret*, con el objetivo de proteger la información privada si este archivo se comparte al público.

6.10 Fase 5. Integración de la base de datos. *Shell Command*.

Para almacenar toda la información relativa a las detecciones de movimiento vamos a utilizar una base de datos en SQLite. A cada una de estas detecciones la llamaremos **acceso**, y contendrá una serie de datos que serán de interés para el usuario. Estos campos están explicados en detalle en la sección 4.2.2 (*Diseño de la base de datos*).

La implementación de la base de datos va a ser realizada en *scripts* de Python, dado que no existe una integración concreta en Home Assistant que pueda resolver un problema tan específico. Estos *scripts* podrán ser ejecutados al ser llamados por otra integración, *Shell Command*.

El directorio que contiene los *scripts* mencionados se ubica en la ruta */home/homeassistant/.homeassistant/scripts*, el cual contiene todo el código Python necesario para el funcionamiento de la base de datos. Ésta se ubicará en la ruta */home/homeassistant/.homeassistant* con el nombre de archivo ***access-control.db***.

A continuación se van a explicar en detalle los *scripts* que componen esta implementación y cómo interactúan entre ellos. El código completo se proporciona en el código 2 en el anexo (*Implementación de la base de datos de accesos*).

- El archivo ***db.py*** se encarga de establecer la conexión a la base de datos SQLite o crearla si no existe. Además, crea una sesión desde la que podremos interactuar con ella. Para ello se emplea el ORM llamado SQLAlchemy, que nos permitirá tratar los registros de la base de datos como objetos.

- El archivo ***models.py*** representa la clase **Access**, que se utilizará para mapear e insertar accesos en bases de datos o consultarlos cuando sea necesario. Además, incluye los métodos de consulta y administración de accesos a base de datos según unos filtros establecidos, como el identificador, el código o la fecha.
- El archivo ***main.py*** es el ejecutable principal y, según los argumentos que reciba, se encargará de insertar un acceso en base de datos, consultarlo según el parámetro recibido o eliminarlo (pueden ser múltiples accesos).

```
shell_command:
  store_access: python /home/homeassistant/.homeassistant/scripts/main.py
 -l {{ location }} -s {{ snapshot_path }} -r {{ record_path }} -
a {{ automatic }}
  get_access_by_id: python /home/homeassistant/.homeassistant/scripts/mai
n.py -m get_access_by_id -i {{ id }}
  get_access_by_code: python /home/homeassistant/.homeassistant/scripts/m
ain.py -m get_access_by_code -c {{ code }}
  get_access_by_date: python /home/homeassistant/.homeassistant/scripts/m
ain.py -m get_access_by_date --from-date {{ from_date }}
  get_access_by_date_range: python /home/homeassistant/.homeassistant/scr
ipts/main.py -m get_access_by_date_range --from-date {{ from_date }} --
to-date {{ to_date }}
  get_access_by_location: python /home/homeassistant/.homeassistant/scrip
ts/main.py -m get_access_by_location -l {{ location }}
  get_access_by_filename: python /home/homeassistant/.homeassistant/scrip
ts/main.py -m get_access_by_filename -f {{ file }}
  get_automatic_access: python /home/homeassistant/.homeassistant/scripts
/main.py -m get_automatic_access -a {{ automatic }}
  get_snapshot_by_access_id: python /home/homeassistant/.homeassistant/sc
ripts/main.py -m get_snapshot_by_access_id -i {{ id }}
  get_snapshot_by_filename: python /home/homeassistant/.homeassistant/scr
ipts/main.py -m get_snapshot_by_filename -f {{ file }}
  get_record_by_access_id: python /home/homeassistant/.homeassistant/scri
pts/main.py -m get_record_by_access_id -i {{ id }}
  get_record_by_filename: python /home/homeassistant/.homeassistant/scrip
ts/main.py -m get_record_by_filename -f {{ file }}
  delete_access_by_id: python /home/homeassistant/.homeassistant/scripts/
main.py -m delete_access_by_id -i {{ id }}
  delete_access_by_date: python /home/homeassistant/.homeassistant/script
s/main.py -m delete_access_by_date --from-date {{ from_date }}
  delete_access_by_date_range: python /home/homeassistant/.homeassistant/
scripts/main.py -m delete_access_by_date_range --from-
date {{ from_date }} --to-date {{ to_date }}
  clear: python /home/homeassistant/.homeassistant/scripts/main.py -
m clear
  move_camera: python /home/homeassistant/.homeassistant/scripts/pyezviz/
main.py -u {{ user }} -p {{ password }} -l {{ location }} -
d {{ direction }}
```

Figura 45. Integración Shell Command

El objetivo del código implementado es poder ser ejecutado desde Home Assistant mediante la integración *Shell Command*. Estas llamadas se ubicarán en el archivo *configuration.yaml* y ejecutará las órdenes especificadas en una terminal. Un dato importante a tener en cuenta es que estas órdenes deben ser invocadas en una automatización para ser utilizadas, por lo que ahora definiremos únicamente su estructura, tal y como se muestra en la figura 45.

A continuación se va a explicar una orden de cada tipo, ya que estas órdenes pueden dividirse en grupos que realizan la misma función.

- La orden ***store_access*** inserta un nuevo acceso en base de datos dada la ubicación del acceso, la ruta de imagen, la ruta de la grabación y el tipo de acceso (si es automático o no).
- Las órdenes ***get*** consultan a la base de datos los accesos según el método de consulta dado y que contengan un campo según los argumentos proporcionados. Estas órdenes requieren nombre del método de consulta para poder ser ejecutadas.
- Las órdenes ***delete*** eliminan de la base de datos los accesos según el método de administración de accesos dado y que contengan un campo según los argumentos proporcionados. Estas órdenes requieren el nombre del método de administración de accesos para poder ser ejecutadas.
- La orden ***clear*** cumple la misma función que una orden *delete*, pero elimina completamente los registros de la base de datos.
- La orden ***move_camera*** se encarga de desplazar una cámara PTZ de una ubicación dada en la dirección especificada.

El parámetro **-m** denota el método de consulta o administración de accesos. Si no es indicado se entiende que se va a almacenar un acceso con los archivos de imagen (-s) y grabación (-r) indicados. Los demás parámetros indican información adicional relativa a un campo de los accesos que va a ser usada para la consulta, como el identificador (-i), el código (-c), la fecha de creación (--from-date) o la ubicación (-l).

Con todas las órdenes implementadas, ya está todo listo para que puedan ser invocadas desde las automatizaciones. Las automatizaciones relativas a las órdenes se dispararán cuando la persona usuaria escriba una orden concreta con su respectivo argumento en la aplicación de mensajería.

6.11 Automatización y detección de movimiento

Las automatizaciones de Home Assistant funcionan por **eventos**, por lo que siempre es necesario un disparador. Cuando la condición para disparar el evento se cumple, éste es invocado y se ejecutan una serie de acciones definidas por el usuario. Gracias a este comportamiento es posible crear una instalación automática y reactiva a las acciones de la persona usuaria.

El paso final consistirá en definir todas estas automatizaciones. Las automatizaciones se dividirán en varios grupos lógicos, los cuales coinciden con los tipos de órdenes de la integración *Shell Command*. Cada automatización se encargará de invocar la orden correspondiente, ya que la mayoría operará con la base de datos o realizará algún tipo de acción.

El objetivo principal será que, cuando exista una detección de movimiento por parte del sensor PIR, se almacene una imagen y una grabación, para que ésta última sea enviada a Telegram para su visualización. Esto creará un nuevo acceso en base de datos, con toda la información relativa a éste.

Dado que existe un gran número de automatizaciones y la mayoría funciona de manera similar dentro de su grupo lógico, vamos a explicar aquellas más relevantes y con un comportamiento diferente. El código completo de las automatizaciones, así como el de la configuración de las integraciones, se encuentra disponible en los códigos 3 y 4 en el anexo.

A continuación se va a explicar en detalle la automatización que maneja la detección de movimiento.

En primer lugar, y como se observa en la automatización de la figura 46, en la cabecera definimos unas variables que contienen la ruta de las imágenes y grabaciones de vídeo que serán tomadas por la cámara, utilizando como formato la fecha y hora actual. El ***trigger*** es la condición mediante la cual se disparará la automatización. Si el valor del sensor cambia de “Sensor desactivado” a “Presencia detectada” significa que éste ha capturado movimiento en la ubicación en la que se encuentra.

Una vez se ha disparado la automatización, se ejecutan una serie de acciones definidas en orden secuencial. La primera de ellas será enviar una notificación de aviso de detección de movimiento al servicio Telegram. Posteriormente, iniciará el *streaming* de la cámara y se almacenará tanto la imagen como la grabación del acceso en la ruta que se ha definido en las variables de la cabecera. Para que esto suceda, se aplica un *delay* o retraso de 20 segundos, suficiente para que la grabación que hemos configurado con una duración de 8 segundos se almacene con éxito en memoria.

Con la imagen y la grabación ya guardadas, se enviará otra notificación a Telegram con el *streaming* de la cámara, por lo que la persona usuaria conocerá quién o qué ha sido el causante del movimiento producido. Por último, se creará un acceso en base de datos, con su respectivo identificador, código, fecha de creación, ubicación y tipo de acceso. El tipo de acceso en este caso será automático (*True*).

```
- alias: Almacenar acceso por presencia
  variables:
    snapshot_filename: /config/photos/snapshot_{{ now().strftime("%Y-%m-
%dT%H.%M.%S") }}.jpg
    record_filename: /config/videos/record_{{ now().strftime("%Y-%m-
%dT%H.%M.%S") }}.mp4
  trigger:
  - platform: state
    entity_id: sensor.sensor_dormitorio
    from: Sensor desactivado
    to: Presencia detectada
  action:
  - service: notify.telegram
    data:
      message: PRESENCIA DETECTADA. Enviando vídeo...
  - service: camera.snapshot
    data:
      entity_id: camera.camara_dormitorio
      filename: '{{ snapshot_filename }}'
  - service: camera.record
    data:
      entity_id: camera.camara_dormitorio
      filename: '{{ record_filename }}'
      duration: 8
  - delay: 00:00:20
  - service: notify.telegram
    data:
      message: PRESENCIA DETECTADA
      data:
        video:
        - file: '{{ record_filename }}'
          caption: '{{ record_filename }}'
  - service: shell_command.store_access
    data_template:
      location: Dormitorio
      snapshot_path: '{{ snapshot_filename }}'
      record_path: '{{ record_filename }}'
      automatic: True
  mode: single
```

Figura 46. Automatización para almacenar acceso por presencia

Además de la automatización por detección de presencia, existen otras automatizaciones con diferentes funciones. Unas se encargan de realizar las operaciones en base de datos (consulta o administración) y otras manejan el comportamiento de la aplicación de mensajería cuando reciben una orden de la persona usuaria.

En los siguientes puntos se va a explicar a grandes rasgos cada una de las automatizaciones y su función a nivel global:

- **Almacenar acceso por presencia**: Se dispara a causa de una detección de movimiento por el sensor PIR. Almacena una imagen y una grabación de vídeo, y envía ésta última a Telegram. Se inserta un nuevo acceso en base de datos.
- **Almacenar acceso por movimiento**: Se dispara a causa de una detección de movimiento por el *streaming* de la cámara. Almacena una imagen y una grabación de vídeo, y envía ésta última a Telegram. Se inserta un nuevo acceso en base de datos.
- **Almacenar acceso por ruido**: Se dispara a causa de una detección de ruido por el *streaming* de la cámara. Almacena una imagen y una grabación de vídeo, y envía ésta última a Telegram. Se inserta un nuevo acceso en base de datos.
- **Arrancar *bot* Telegram**: Se dispara al recibir la orden */start* desde Telegram. Muestra el menú principal en un mensaje como bienvenida al usuario.
- **Consultar accesos**: Se dispara al recibir la orden */query_menu* desde Telegram. Muestra el menú de órdenes de consulta de accesos.
- **Administrar accesos**: Se dispara al recibir la orden */manage_menu* desde Telegram. Muestra el menú de órdenes de administración de accesos.
- **Menú live *streaming***: Se dispara al recibir la orden */live_menu* desde Telegram. Muestra el menú de órdenes de administración de accesos.
- **Menú controlar PTZ**: Se dispara al recibir la orden */ptz_menu* desde Telegram. Muestra el menú de movimiento PTZ de la cámara.
- **Obtener acceso por campo**: Son un conjunto de automatizaciones que realizan una consulta a base de datos y obtienen uno o varios accesos dependiendo de la orden y argumentos dados. Se disparan al recibir una orden */get* desde Telegram. Muestra en un mensaje los accesos recibidos y el valor de cada uno de sus campos (si se ha encontrado al menos uno con el filtro especificado).
- **Eliminar acceso por campo**: Son un conjunto de automatizaciones que realizan un borrado en base de datos de uno o varios accesos obtenidos dependiendo de la orden y argumentos dados. Se disparan al recibir una orden */delete* desde Telegram. Muestra un mensaje de confirmación con el acceso eliminado (si se ha encontrado al menos uno con el filtro especificado).
- **Reiniciar base de datos**: Se dispara al recibir una orden */clear* desde Telegram. Reinicia completamente la base de datos, eliminando todos los accesos guardados hasta la fecha.

- **Live *streaming***: Se dispara al recibir una orden */live* desde Telegram. Genera un acceso de tipo manual, proporcionando una grabación de vídeo en el momento de envío de la orden.
- **Movimiento de rotación cámara**: Son un conjunto de automatizaciones que realizan un movimiento de rotación en una cámara PTZ. Se disparan al recibir una orden */go* desde Telegram. Muestra una imagen de la cámara en el momento de envío de la orden para verificar su posición actual.

Como se ha podido observar, todas las automatizaciones relacionadas con la base de datos interactúan con el servicio *Shell Command* para realizar la operación correspondiente, ya sea consultar o eliminar registros, a excepción de las automatizaciones de movimiento de rotación, que también llaman a este servicio, pero con el objetivo de ejecutar una acción independiente.

La automatización *Live streaming* es muy similar a aquellas de almacenamiento de acceso, pero creando un acceso manual. Fue diseñada con el objetivo de obtener el estado actual de una estancia de forma persistente, sin la necesidad de un disparador automático para crear un acceso.

Con todas las automatizaciones necesarias implementadas, solo faltaría realizar un conjunto de pruebas finales para verificar que el conjunto de la instalación funciona satisfactoriamente.

7. Pruebas

En este apartado se van a elaborar una serie de casos de prueba para verificar que todas las funcionalidades implementadas funcionan según lo esperado.

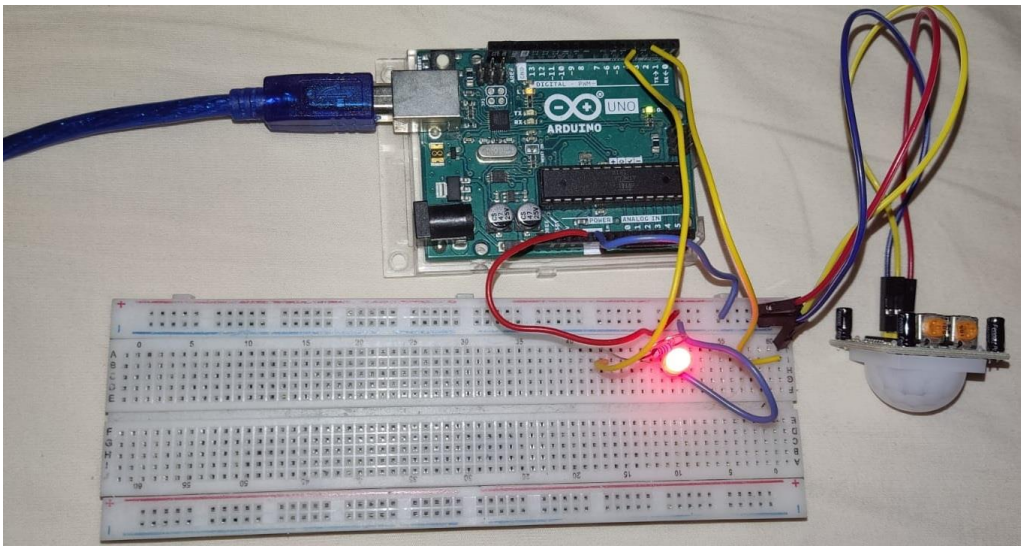

Figura 47. Activación del sensor de movimiento PIR

Nos situamos inicialmente en el campo de acción del sensor de movimiento PIR. En la figura 47 se puede observar cómo se activa el LED en el momento de la detección, por lo que se deduce que la captura de movimiento funciona correctamente. A su vez, el valor de las tarjetas en la interfaz Lovelace de Home Assistant es alterado (figura 48).

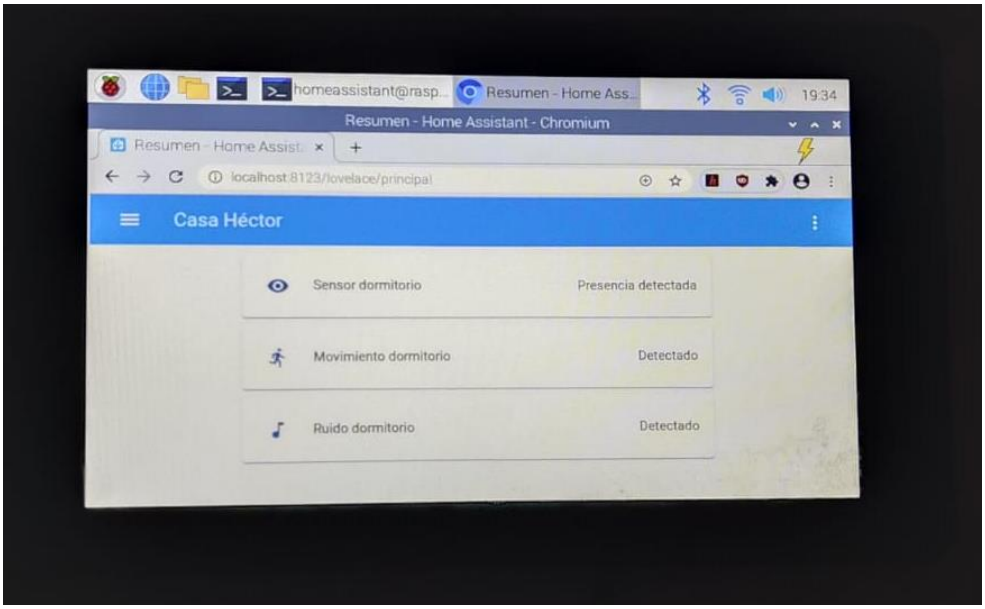

Figura 48. Presencia detectada en interfaz Lovelace

En ese mismo instante recibimos una notificación en Telegram, con el mensaje "PRESENCIA DETECTADA" y la grabación de vídeo resultante (figura 49).

Figura 49. Notificación de presencia detectada en Telegram

Realizamos una consulta a la base de datos enviando la orden */getaccessbyid 340*, la cual devolverá la información relativa al acceso con identificador 340. También enviaremos la orden */getsnapshotbyaccessid 340*, que devolverá la imagen de este acceso (figura 50).

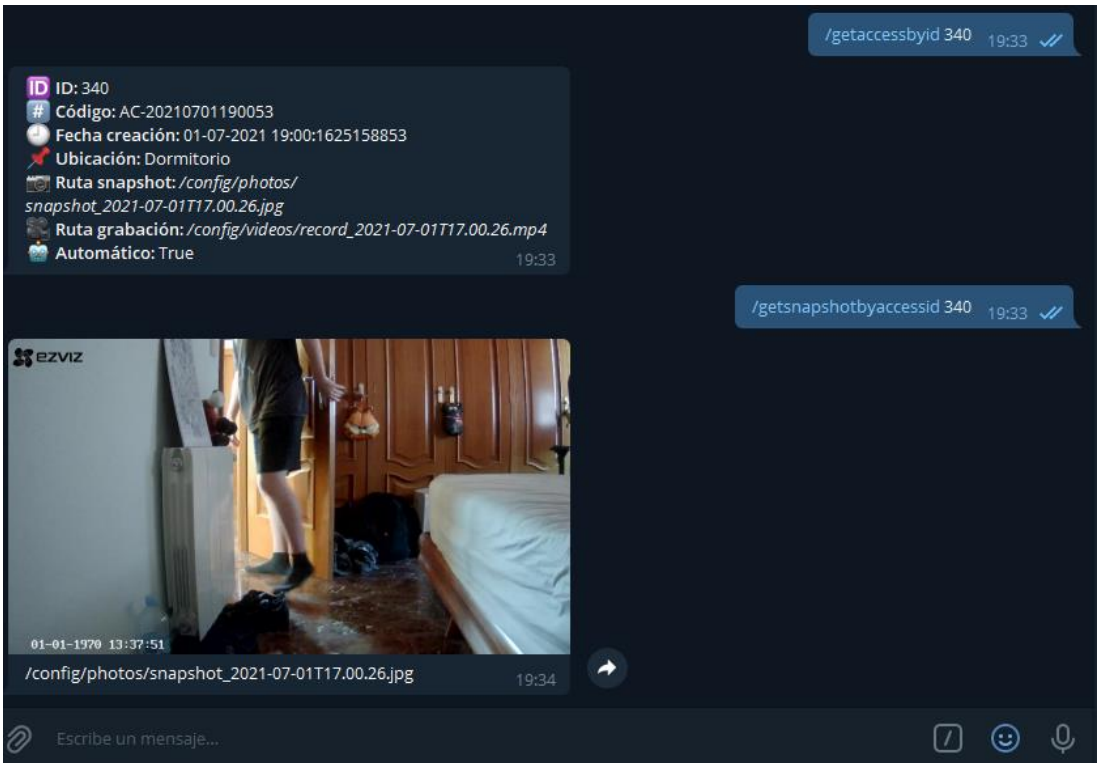

Figura 50. Obtener acceso e imagen por identificador

Para obtener una lista de accesos, podemos utilizar la orden */getaccessbydaterange*, la cual devuelve un conjunto de accesos que se encuentre entre una fecha de inicio y una fecha de fin. En la figura 51 se obtienen los accesos del día 1 de julio de 2021, entre las 19:00:16 y las 19:05:00.

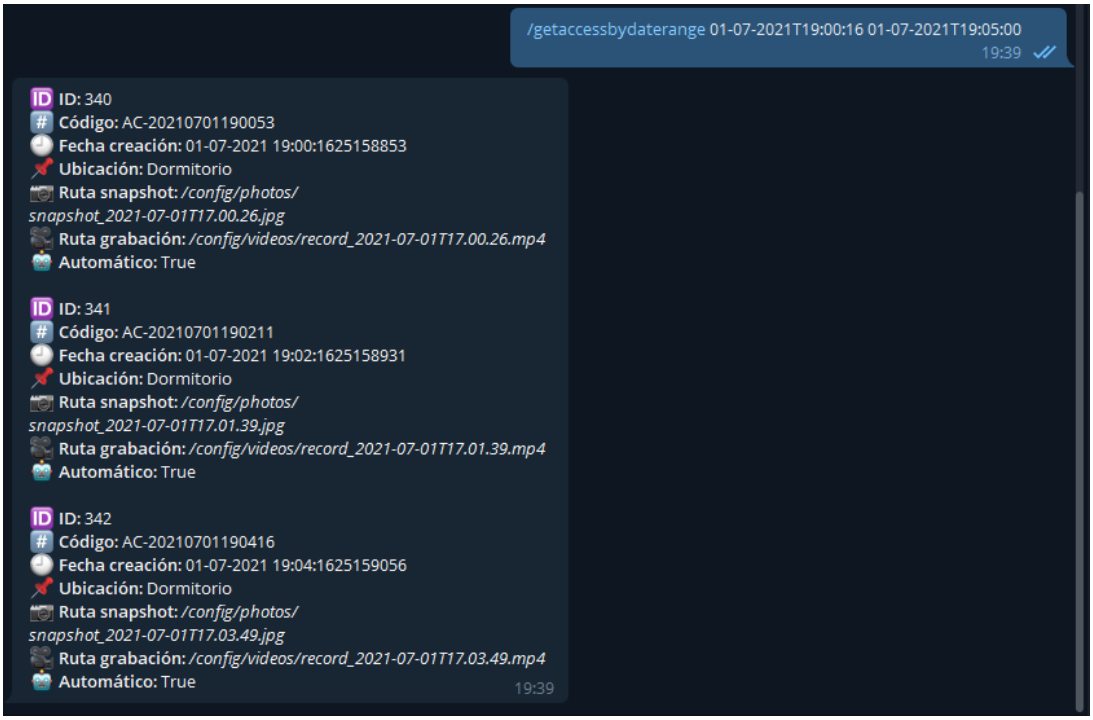

Figura 51. Obtener accesos por rango de fecha

A continuación vamos a eliminar un acceso. Se observa en la figura 52 que primero se ejecuta la orden de borrado, */deleteaccessbyid 1*, la cual elimina el acceso con identificador 1, y posteriormente se realiza una consulta del mismo, la cual no devuelve el acceso porque ha sido eliminado por la orden anterior:

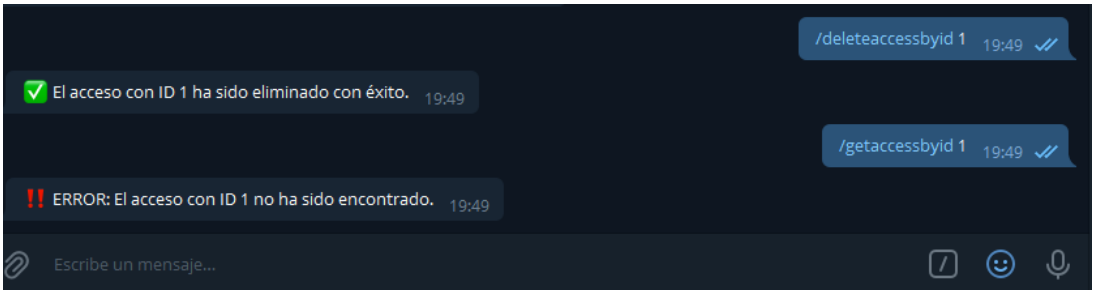

Figura 52. Eliminar un acceso por identificador

Es posible desplazar una cámara a una ubicación determinada mediante las órdenes */goup*, */godown*, */goleft* y */goright*, las cuales devuelven una imagen de prueba para verificar la nueva posición. El ejemplo de la figura 53 muestra cómo varía la posición dependiendo de la orden enviada.

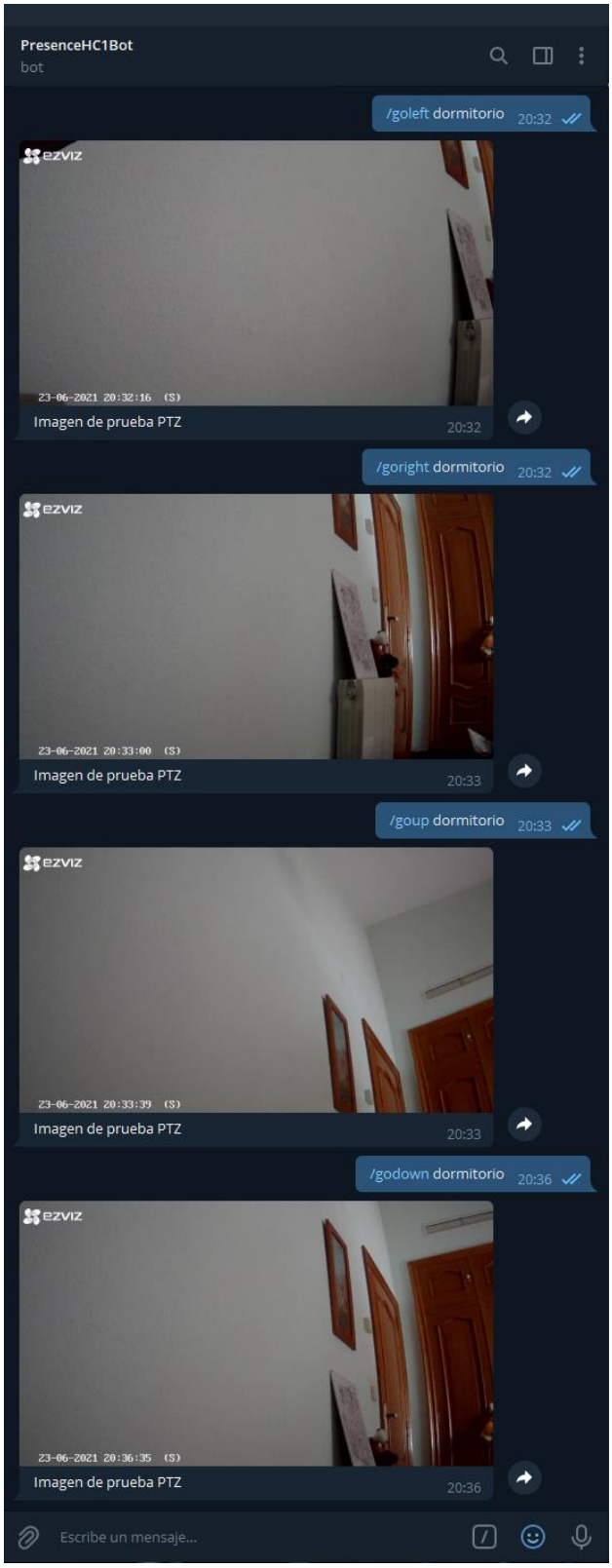

Figura 53. Movimiento PTZ de una cámara en Telegram

Si queremos obtener el *streaming* de la cámara en tiempo real sin la necesidad de un trigger como el sensor PIR, podemos enviar la orden */live*, que creará un acceso manual y nos devolverá la grabación de vídeo obtenida (figura 54).

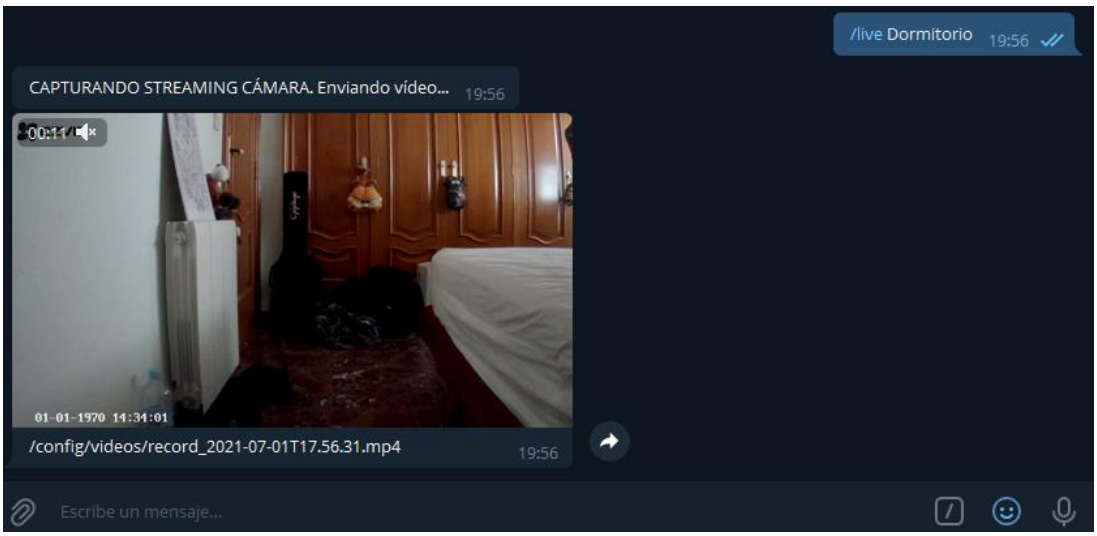

Figura 54. Obtener streaming de la cámara IP

8. Conclusiones

En este trabajo se han mostrado muchas de las posibilidades que se pueden lograr con una instalación domótica de código abierto, utilizando para ello equipo de bajo coste. La domótica y la automatización tienen cada vez un peso mayor en la sociedad actual y los usos prácticos que presentan las tecnologías que intervienen en el proceso facilitan y mejoran la calidad de vida de sus usuarios.

Se ha logrado el objetivo de crear un sistema domótico de vigilancia que interactúa con una aplicación de mensajería, desde la cual es posible controlar toda la instalación desde la palma de la mano. Para ello ha sido necesario aprender a diseñar un circuito eléctrico desde cero e implementarlo en una placa mediante un microcontrolador Arduino UNO, con todos los componentes necesarios para su funcionamiento y programando en éste el comportamiento deseado.

Se han integrado correctamente todos los componentes del sistema de vigilancia, incluyendo la cámara y los sensores, para elaborar un sistema completo con multitud de tecnologías, como MQTT, YAML y SQLite.

Se ha introducido a quien lea este documento en el mundo de la domótica, permitiendo aprender los conceptos básicos requeridos para iniciarse en este campo emergente, además de entender la interacción entre los distintos componentes de la arquitectura.

8.1 Relación del trabajo desarrollado con los estudios cursados

En gran medida, el desarrollo de este trabajo ha requerido autoaprendizaje por parte del alumno, sobre todo en el campo de la domótica y en el uso de los dispositivos y componentes eléctricos. Principalmente guarda relación con la asignatura FFI (Fundamentos Físicos de la Informática) y con TCO (Tecnología de Computadores), en cuanto al diseño de circuitos y arquitectura se refiere.

Por otro lado, han sido necesarios conocimientos en la asignatura SSI (Seguridad en los Sistemas de Información), ya que para llevar a cabo las integraciones de la plataforma se han utilizado *scripts* en el lenguaje de programación Python. La asignatura CCO (Control por COmputador) ha sido de utilidad como introducción en el campo de la automatización. Ha sido de vital importancia el conocimiento adquirido en la rama de Ingeniería de Computadores, tanto para la planificación como para el desarrollo del trabajo.

9. Trabajos futuros

Para complementar el desarrollado realizado y añadir nueva funcionalidad se propone un nuevo trabajo:

Integración de un sistema de vigilancia en una plataforma de código abierto mediante el uso de Inteligencia Artificial

Este trabajo podría realizarse en una plataforma *open source* como Home Assistant y utilizar la plataforma de mensajería deseada para el envío de notificaciones. El objetivo principal sería emplear la Inteligencia Artificial para crear un sistema de vigilancia aún más práctico, con la capacidad de detectar quién ha sido el causante de la detección de movimiento, manteniendo para ello una base de datos con todos los rostros de los miembros que viven en nuestra estancia. Esto sería muy útil para detectar intrusiones y prevenir posibles robos en el hogar.

Por agregar más funcionalidad, también sería posible distinguir entre personas o vehículos. Puede ser interesante colocar una cámara en el garaje de nuestra vivienda y detectar la matrícula del vehículo que está aparcando, con el objetivo de mantener un historial de entradas y salidas.

Por otro lado, sería de gran utilidad en los tiempos que corren integrar un termómetro infrarrojo para detectar aquellas personas con una temperatura más elevada de lo normal, para prevenir la entrada de personas posiblemente contagiadas por coronavirus.

10. Bibliografía

[1] Arduino, *Getting Started with Arduino UNO*, 2018. [En línea]. Disponible en: https://www.arduino.cc/en/Guide/ArduinoUno.

[2] IBM, *Conozca MQTT*, 2017. [En línea]. Disponible en: https://developer.ibm.com/es/articles/iot-mqtt-why-good-for-iot.

[3] D. Evans, *Internet of Things*, 2011. [En línea]. Disponible en: https://www.cisco.com/c/dam/global/es_es/assets/executives/pdf/Internet_of_Things_IoT_IBSG_0411FINAL.pdf.

[4] Home Assistant, *Home Assistant documentation*, [En línea]. Disponible en: https://www.home-assistant.io/docs.

[5] C. Lasa Gómez, A. Álvarez García y R. de las Heras del Dedo, *Métodos Ágiles. Scrum, Kanban, Lean*, Anaya, 2017.

[6] CEDOM, *Qué es domótica*, [En línea]. Disponible en: http://www.cedom.es/sobre-domotica/que-es-domotica.

[7] J. Moreno Gil, E. Rodríguez Dieguez y D. Lasso Tarraga, *Instalaciones Automatizadas en Viviendas y Edificios*, Paraninfo, 2001.

[8] Micrónica, *Sensores, actuadores y elementos del sistema de control*, [En línea]. Disponible en: http://www.micronica.es/files/pdfs/SIHD/SIHD_Sens_Actu_EC.pdf.

[9] L. M. Cerdá Filiu y M. Gas Bueno, *Instalaciones domóticas*, Paraninfo, 2020.

[10] H. P. Talancón, *La matriz FODA: Alternativa de diagnóstico y determinación de estrategias de intervención en diversas organizaciones*, *Redalyc*, 2007.

[11] Domoticz, *About Domoticz*, [En línea]. Disponible en: https://www.domoticz.com/wiki/About_Domoticz.

[12] R. Ierusalimschy, *Programming in Lua*, Lua.org, 2016.

[13] openHAB, *openHAB introduction*, [En línea]. Disponible en: https://www.openhab.org/docs.

[14] ioBroker, *ioBroker Grundlagen*, [En línea]. Disponible en: https://www.iobroker.net/#de/documentation/basics/README.md.

[15] HomeGenie, *HomeGenie introduction*, [En línea]. Disponible en: https://genielabs.github.io/HomeGenie/#/about.

[16] Jeedom, *Documentación de Jeedom*, [En línea]. Disponible en: https://doc.jeedom.com/es_ES.

[17] P. A. Henning, *Smart Home mit FHEM*, Hanser, 2019.

[18] LinuxMCE, *LinuxMCE User Manual*, 2007. [En línea]. Disponible en: http://wiki.linuxmce.org/index.php/User_Manual.

[19] Switchur, *Automating your life*, [En línea]. Disponible en: https://blog.switchur.com/2017/12/03/Automating-your-life.

[20] Telegram, *Telegram APIs*, [En línea]. Disponible en: https://core.telegram.org/api.

[21] T. Nield, *Getting Started with SQL*, O'Reilly, 2016.

[22] Raspberry Pi Foundation, *Raspberry Pi 3 Model B Overview & Specifications*, [En línea]. Disponible en: https://static.raspberrypi.org/files/product-briefs/Raspberry-Pi-Model-Bplus-Product-Brief.pdf.

[23] Velleman, *VMA314: PIR motion sensor for Arduino*, 2018. [En línea]. Disponible en: https://www.velleman.eu/downloads/29/vma314_a4v02.pdf.

[24] EZVIZ, *Smart Wi-Fi Pan & Tilt Camera*, [En línea]. Disponible en: https://mfs.ezvizlife.com/C6N_datasheet.pdf?ver=26728.

[25] O. Ben-Kiki, C. Evans y I. döt Net, *YAML Ain't Markup Language Version 1.2*, 2009. [En línea]. Disponible en: https://yaml.org/spec/1.2.

[26] Raspberry Pi Foundation, *Raspberry Pi Quick Start Guide*, [En línea]. Disponible en: https://www.raspberrypi.org/app/uploads/2012/12/quick-start-guide-v1.1.pdf.

[27] Balena, *etcherPro Getting Started Guide*, [En línea]. Disponible en: https://www.balena.io/etcher/pro/docs/getting-started.

[28] OpenStreetMap, *Usando OpenStreetMap*, [En línea]. Disponible en: https://wiki.openstreetmap.org/wiki/ES:Usando_OpenStreetMap.

[29] D. Mateos Costilla y S. Reaño Montoro, *Streaming de Audio/Video. Protocolo RTSP*, *Serveis Telemàtics*, 2007.

[30] P. Ourdouille, *Pilot your EZVIZ cameras*, [En línea]. Disponible en: https://github.com/BaQs/pyEzviz.

[31] P. Björkdahl, *Guía ONVIF para principiantes*, *Negocios de Seguridad*, 2016.

11. Anexos

Código 1: Programa de movimiento PTZ de una cámara IP EZVIZ

Código 2: Implementación de la base de datos de accesos

Código 3: Configuración de las integraciones de Home Assistant

Código 4: Automatizaciones de Home Assistant

Código 1. Programa de movimiento PTZ de una cámara IP EZVIZ

```
##### Archivo main.py #####

import sys
import getopt
import json
from client import EzvizClient
from camera import EzvizCamera

def run():
    arguments, values = getopt.getopt(sys.argv[1:], 'u:p:l:d:', ['user=', 'password
=', 'location=' 'direction='])
    user = None
    password = None
    direction = None
    location = None

    for currentArgument, currentValue in arguments:
        if currentArgument in ('-u', '--user'):
            user = currentValue

        if currentArgument in ('-p', '--password'):
            password = currentValue

        if currentArgument in ('-l', '--location'):
            location = currentValue

        if currentArgument in ('-d', '--direction'):
            direction = currentValue

    with open('/home/homeassistant/.homeassistant/scripts/pyezviz/camera_serial.jso
n') as jsonFile:
        jsonObject = json.load(jsonFile)
        jsonFile.close()

    client = EzvizClient(user, password, 'apiieu.ezvizlife.com')
    client.login()

    camera = EzvizCamera(client, jsonObject[location])
    camera.move(direction)

if __name__ == '__main__':
    run()

```

```
/***** Archivo camera_serial.json *****/

{
    "dormitorio": "E56406355"
}

```

Código 2. Implementación de la base de datos de accesos

```
##### Archivo db.py #####

from pathlib import Path
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from sqlalchemy.ext.declarative import declarative_base

engine = create_engine('sqlite:///' + str(Path(__file__).parents[1].joinpath('access-control.db')))
Session = sessionmaker(bind=engine)
session = Session()

Base = declarative_base()
```

```
##### Archivo models.py #####

import db
from datetime import datetime
from pathlib import Path
from sqlalchemy import Column, Integer, Boolean, String, Date, DateTime, func, or_

class Access(db.Base):
    __tablename__ = 'access'
    now = datetime.now()

    id = Column(Integer, primary_key=True)
    code = Column(String, unique=True, default=f'AC-{now.strftime("%Y%m%d%H%M%S")}')
    created_date = Column(DateTime, default=now)
    location = Column(String)
    snapshot_path = Column(String)
    record_path = Column(String)
    automatic = Column(Boolean, default=True)

    def __init__(self, location, snapshot_path, record_path, automatic):
        self.location = location
        self.automatic = automatic

        if Path(snapshot_path).exists():
            self.snapshot_path = snapshot_path

        if Path(record_path).exists():
            self.record_path = record_path

    def __str__(self):
        text = ''

        text += f'\U0001F194 <b>ID</b>: {self.id}\\n'
        text += f'\U00000023\U000020E3 <b>Código</b>: {self.code}\\n'
        text += f'\U0001F558 <b>Fecha creación</b>: {self.created_date:%d-%m-%Y %H:%M:%s}\\n'
        text += f'\U0001F4CC <b>Ubicación</b>: {self.location}\\n'
        text += f'\U0001F4F7 <b>Ruta snapshot</b>: <i>{self.snapshot_path}</i>\\n'
        text += f'\U0001F3A5 <b>Ruta grabación</b>: <i>{self.record_path}</i>\\n'
        text += f'\U0001F916 <b>Automático</b>: {self.automatic}\\n'

        return text

    @staticmethod
    def get_access_by_id(id):
        return db.session.query(Access).get(id)

    @staticmethod
    def get_access_by_code(code):
        return db.session.query(Access).filter(Access.code == code).first()

```

```
51.     @staticmethod
52.     def get_access_by_date(date):
53.         return db.session.query(Access).filter(func.DATE(Access.created_date) == da
    tetime.strptime(date, '%d-%m-%Y').date()).all()
54.
55.     @staticmethod
56.     def get_access_by_date_range(from_date, to_date):
57.         return db.session.query(Access).filter(
58.             Access.created_date >= datetime.strptime(from_date, '%d-%m-
    %YT%H:%M:%S'),
59.             Access.created_date <= datetime.strptime(to_date, '%d-%m-
    %YT%H:%M:%S')).all()
60.
61.     @staticmethod
62.     def get_access_by_location(location):
63.         return db.session.query(Access).filter(Access.location == location).all()
64.
65.     @staticmethod
66.     def get_access_by_filename(file):
67.         return db.session.query(Access).filter(
68.             or_(
69.                 Access.snapshot_path.contains(file),
70.                 Access.record_path.contains(file))).first()
71.
72.     @staticmethod
73.     def get_automatic_access(automatic):
74.         return db.session.query(Access).filter(Access.automatic == automatic).all()
75.
76.     @staticmethod
77.     def delete_access_by_id(id):
78.         db.session.query(Access).filter(Access.id == id).delete()
79.         db.session.commit()
80.
81.     @staticmethod
82.     def delete_access_by_date(date):
83.         db.session.query(Access).filter(func.DATE(Access.created_date) == datetime.
    strptime(date, '%d-%m-%Y').date()).delete(synchronize_session='fetch')
84.         db.session.commit()
85.
86.     @staticmethod
87.     def delete_access_by_date_range(from_date, to_date):
88.         db.session.query(Access).filter(
89.             Access.created_date >= datetime.strptime(from_date, '%d-%m-
    %YT%H:%M:%S'),
90.             Access.created_date <= datetime.strptime(to_date, '%d-%m-
    %YT%H:%M:%S')).delete(synchronize_session='fetch')
91.
92.         db.session.commit()
93.
94.     @staticmethod
95.     def clear():
96.         db.session.query(Access).delete()
97.         db.session.commit()
98.
```

```
1.  ##### Archivo main.py #####
2.
3.  import sys
4.  import getopt
5.  import subprocess
6.  import db
7.  from models import Access
8.
9.  def run():
10.     arguments, values = getopt.getopt(sys.argv[1:], 'i:c:l:f:s:r:a:m:', ['id=', 'co
    de=', 'from-date=', 'to-date=', 'file=', 'snapshot-path=', 'record-
    path=', 'automatic=', 'method='])
11.     id = None
12.     code = None
```

```
13.     from_date = None
14.     to_date = None
15.     file = None
16.     location = None
17.     snapshot_path = None
18.     record_path = None
19.     automatic = True
20.     method = None
21.
22.     for currentArgument, currentValue in arguments:
23.         if currentArgument in ('-i', '--id'):
24.             id = int(currentValue)
25.
26.         if currentArgument in ('-c', '--code'):
27.             code = currentValue
28.
29.         if currentArgument in ('--from-date'):
30.             from_date = currentValue
31.
32.         if currentArgument in ('--to-date'):
33.             to_date = currentValue
34.
35.         if currentArgument in ('-f', '--file'):
36.             file = currentValue
37.
38.         if currentArgument in ('-l', '--location'):
39.             location = currentValue
40.
41.         if currentArgument in ('-s', '--snapshot-path'):
42.             snapshot_path = currentValue
43.
44.         if currentArgument in ('-r', '--record-path'):
45.             record_path = currentValue
46.
47.         if currentArgument in ('-a', '--automatic'):
48.             automatic = currentValue.lower() in ('1', 't', 'true', 'y', 'yes')
49.
50.         if currentArgument in ('-m', '--method'):
51.             method = currentValue
52.
53.     if (method == None):
54.         access = Access(location, snapshot_path, record_path, automatic)
55.         db.session.add(access)
56.         db.session.commit()
57.
58.     else:
59.         result = ''
60.
61.         if method == 'get_access_by_id':
62.             if id == None:
63.                 result = '\U0000203C ERROR: El ID del acceso no ha sido especificad
    o en el comando.'
64.             else:
65.                 access = Access.get_access_by_id(id)
66.
67.                 if access == None:
68.                     result = f'\U0000203C ERROR: El acceso con ID {id} no ha sido e
    ncontrado.'
69.                 else:
70.                     result = str(access)
71.         elif method == 'get_access_by_code':
72.             if code == None:
73.                 result = '\U0000203C ERROR: El código del acceso no ha sido especif
    icado en el comando.'
74.             else:
75.                 access = Access.get_access_by_code(code)
76.
77.                 if access == None:
78.                     result = f'\U0000203C ERROR: El acceso con código {code} no ha
    sido encontrado.'
```


```
79.                 else:
80.                     result = str(access)
81.         elif method == 'get_access_by_date':
82.             if from_date == None:
83.                 result = '\U0000203C ERROR: La fecha del acceso no ha sido especifi
    cada en el comando.'
84.             else:
85.                 access = Access.get_access_by_date(from_date)
86.
87.                 if not access:
88.                     result = f'\U0000203C ERROR: No ha sido encontrado ningún acces
    o con fecha {from_date}'
89.                 else:
90.                     for a in access:
91.                         result += f'{a}\\n'
92.         elif method == 'get_access_by_date_range':
93.             if from_date == None or to_date == None:
94.                 result = '\U0000203C ERROR: La fecha de inicio o de fin del acceso
    no ha sido especificada en el comando.'
95.             else:
96.                 access = Access.get_access_by_date_range(from_date, to_date)
97.
98.                 if not access:
99.                     result = f'\U0000203C ERROR: No ha sido encontrado ningún acces
    o con fecha desde {from_date} hasta {to_date}'
100.                else:
101.                    for a in access:
102.                        result += f'{a}\\n'
103.         elif method == 'get_access_by_location':
104.             if location == None:
105.                 result = '\U0000203C ERROR: La localización no ha sido especifica
    da en el comando.'
106.             else:
107.                 access = Access.get_access_by_location(location)
108.
109.                 if not access:
110.                     result = f'\U0000203C ERROR: No ha sido encontrado ningún acc
    eso con localización {location}'
111.                 else:
112.                     for a in access:
113.                         result += f'{a}\\n'
114.         elif method == 'get_access_by_filename':
115.             if file == None:
116.                 result = '\U0000203C ERROR: El nombre del archivo no ha sido espe
    cificado en el comando.'
117.             else:
118.                 access = Access.get_access_by_filename(file)
119.
120.                 if access == None:
121.                     result = f'\U0000203C ERROR: No ha sido encontrado ningún acc
    eso con nombre de archivo {file}.'
122.                 else:
123.                     result = str(access)
124.         elif method == 'get_automatic_access':
125.             access = Access.get_automatic_access(automatic)
126.
127.             if not access:
128.                 result = f'\U0000203C ERROR: No ha sido encontrado ningún acceso
    {"automático" if automatic else "manual"}'
129.             else:
130.                 for a in access:
131.                     result += f'{a}\\n'
132.         elif method == 'get_snapshot_by_access_id':
133.             if id == None:
134.                 result = '\U0000203C ERROR: El ID del acceso no ha sido especific
    ado en el comando.'
135.             else:
136.                 access = Access.get_access_by_id(id)
137.
138.                 if access == None:
```

```
139.                     result = f'\U0000203C ERROR: El acceso con ID {id} no ha sido
     encontrado.'
140.                 else:
141.                     result = access.snapshot_path
142.         elif method == 'get_record_by_access_id':
143.             if id == None:
144.                 result = '\U0000203C ERROR: El ID del acceso no ha sido especific
    ado en el comando.'
145.             else:
146.                 access = Access.get_access_by_id(id)
147.
148.                 if access == None:
149.                     result = f'\U0000203C ERROR: El acceso con ID {id} no ha sido
     encontrado.'
150.                 else:
151.                     result = access.record_path
152.         elif method == 'get_snapshot_by_filename':
153.             if file == None:
154.                 result = '\U0000203C ERROR: El nombre del archivo no ha sido espe
    cificado en el comando.'
155.             else:
156.                 access = Access.get_access_by_filename(file)
157.
158.                 if access == None:
159.                     result = f'\U0000203C ERROR: No ha sido encontrado ningún acc
    eso con nombre de archivo {file}.'
160.                 else:
161.                     result = access.snapshot_path
162.         elif method == 'get_record_by_filename':
163.             if file == None:
164.                 result = '\U0000203C ERROR: El nombre del archivo no ha sido espe
    cificado en el comando.'
165.             else:
166.                 access = Access.get_access_by_filename(file)
167.
168.                 if access == None:
169.                     result = f'\U0000203C ERROR: No ha sido encontrado ningún acc
    eso con nombre de archivo {file}.'
170.                 else:
171.                     result = access.record_path
172.         elif method == 'delete_access_by_id':
173.             if id == None:
174.                 result = '\U0000203C ERROR: El ID del acceso no ha sido especific
    ado en el comando.'
175.             else:
176.                 Access.delete_access_by_id(id)
177.                 result = f'\U00002705 El acceso con ID {id} ha sido eliminado con
     éxito.'
178.         elif method == 'delete_access_by_date':
179.             if from_date == None:
180.                 result = '\U0000203C ERROR: La fecha del acceso no ha sido especi
    ficada en el comando.'
181.             else:
182.                 Access.delete_access_by_date(from_date)
183.                 result = f"\U00002705 Los accesos con fecha {from_date} han sido
    eliminados con éxito."
184.         elif method == 'delete_access_by_date_range':
185.             if from_date == None or to_date == None:
186.                 result = '\U0000203C ERROR: La fecha de inicio o de fin del acces
    o no ha sido especificada en el comando.'
187.             else:
188.                 Access.delete_access_by_date_range(from_date, to_date)
189.                 result = f"\U00002705 Los accesos con fecha desde {from_date} has
    ta {to_date} han sido eliminados con éxito."
190.         elif method == 'clear':
191.             access = Access.clear()
192.             result = f'\U00002705 Base de datos reiniciada con éxito.'
193.
194.         result = f'{{"type":"access_update","message":"{result}"}}'
```


```
195.           subprocess.run(['mosquitto_pub', '-h', '127.0.0.1', '-
    t', 'telegram/access/query', '-m', result])
196.
197.  if __name__ == '__main__':
198.      db.Base.metadata.create_all(db.engine)
199.      run()
```

Código 3. Configuración de las integraciones de Home Assistant

```
##### Archivo configuration.yaml #####

# Configure a default setup of Home Assistant (frontend, api, etc)
default_config:

# Text to speech
tts:
  - platform: google_translate

homeassistant:
  allowlist_external_dirs:
    - /config/photos
    - /config/videos

group: !include groups.yaml
automation: !include automations.yaml
script: !include scripts.yaml
scene: !include scenes.yaml

mqtt:
  broker: 127.0.0.1

sensor:
  - platform: serial
    baudrate: 9600
    serial_port: /dev/ttyACM0

  - platform: template
    sensors:
      sensor_dormitorio:
        friendly_name: Sensor dormitorio
        value_template: >
          '{% if states.sensor.serial_sensor.state == 'Sensor ON' %}
            Presencia detectada
          {% elif states.sensor.serial_sensor.state == 'Sensor OFF' %}
            Sensor desactivado
          {% else %}
            Sensor desactivado
          {% endif %}'

  - platform: mqtt
    name: access
    state_topic: telegram/access/query
    value_template: '{{ value_json.type }}'
    json_attributes_topic: telegram/access/query
    json_attributes_template: '{{ value_json | tojson }}'

camera:
   - platform: ffmpeg
     name: Camara dormitorio
     input: rtsp://admin:BMFNXC@169.254.1.93:554/H.264

stream:

binary_sensor:
   - platform: ffmpeg_motion
     name: Movimiento dormitorio
     input: rtsp://admin:BMFNXC@169.254.1.93:554/H.264
     changes: 20
     reset: 30
   - platform: ffmpeg_noise
     name: Ruido dormitorio
     input: rtsp://admin:BMFNXC@169.254.1.93:554/H.264
     peak: -10
     reset: 30

telegram_bot:
```

```
68.     - platform: polling
69.       api_key: !secret api_token
70.       parse_mode: html
71.       allowed_chat_ids:
72.         - 42490470
73.
74. notify:
75.     - platform: telegram
76.       name: Telegram
77.       api_key: !secret api_token
78.       chat_id: 42490470
79.
80. logger:
81.     default: info
82.
83. shell_command:
84.   store_access: python /home/homeassistant/.homeassistant/scripts/main.py -
    l {{ location }} -s {{ snapshot_path }} -r {{ record_path }} -a {{ automatic }}
85.   get_access_by_id: python /home/homeassistant/.homeassistant/scripts/main.py -
    m get_access_by_id -i {{ id }}
86.   get_access_by_code: python /home/homeassistant/.homeassistant/scripts/main.py -
    m get_access_by_code -c {{ code }}
87.   get_access_by_date: python /home/homeassistant/.homeassistant/scripts/main.py -
    m get_access_by_date --from-date {{ from_date }}
88.   get_access_by_date_range: python /home/homeassistant/.homeassistant/scripts/main.
    py -m get_access_by_date_range --from-date {{ from_date }} --to-date {{ to_date }}
89.   get_access_by_location: python /home/homeassistant/.homeassistant/scripts/main.py
     -m get_access_by_location -l {{ location }}
90.   get_access_by_filename: python /home/homeassistant/.homeassistant/scripts/main.py
     -m get_access_by_filename -f {{ file }}
91.   get_automatic_access: python /home/homeassistant/.homeassistant/scripts/main.py -
    m get_automatic_access -a {{ automatic }}
92.   get_snapshot_by_access_id: python /home/homeassistant/.homeassistant/scripts/main
    .py -m get_snapshot_by_access_id -i {{ id }}
93.   get_snapshot_by_filename: python /home/homeassistant/.homeassistant/scripts/main.
    py -m get_snapshot_by_filename -f {{ file }}
94.   get_record_by_access_id: python /home/homeassistant/.homeassistant/scripts/main.p
    y -m get_record_by_access_id -i {{ id }}
95.   get_record_by_filename: python /home/homeassistant/.homeassistant/scripts/main.py
     -m get_record_by_filename -f {{ file }}
96.   delete_access_by_id: python /home/homeassistant/.homeassistant/scripts/main.py -
    m delete_access_by_id -i {{ id }}
97.   delete_access_by_date: python /home/homeassistant/.homeassistant/scripts/main.py
    -m delete_access_by_date --from-date {{ from_date }}
98.   delete_access_by_date_range: python /home/homeassistant/.homeassistant/scripts/ma
    in.py -m delete_access_by_date_range --from-date {{ from_date }} --to-
    date {{ to_date }}
99.   clear: python /home/homeassistant/.homeassistant/scripts/main.py -m clear
100.    move_camera: python /home/homeassistant/.homeassistant/scripts/pyezviz/main.py
    -u {{ user }} -p {{ password }} -l {{ location }} -d {{ direction }}
```

Código 4. Automatizaciones de Home Assistant

```
##### Archivo automations.yaml #####

- alias: Almacenar acceso por presencia
  variables:
    snapshot_filename: /config/photos/snapshot_{{ now().strftime('%Y-%m-%dT%H.%M.%S') }}.jpg
    record_filename: /config/videos/record_{{ now().strftime('%Y-%m-%dT%H.%M.%S') }}.mp4
  trigger:
  - platform: state
    entity_id: sensor.sensor_dormitorio
    from: Sensor desactivado
    to: Presencia detectada
  action:
  - service: notify.telegram
    data:
      message: PRESENCIA DETECTADA. Enviando vídeo...
  - service: camera.snapshot
    data:
      entity_id: camera.camara_dormitorio
      filename: '{{ snapshot_filename }}'
  - service: camera.record
    data:
      entity_id: camera.camara_dormitorio
      filename: '{{ record_filename }}'
      duration: 8
  - delay: 00:00:20
  - service: notify.telegram
    data:
      message: PRESENCIA DETECTADA
      data:
        video:
        - file: '{{ record_filename }}'
          caption: '{{ record_filename }}'
  - service: shell_command.store_access
    data_template:
      location: Dormitorio
      snapshot_path: '{{ snapshot_filename }}'
      record_path: '{{ record_filename }}'
      automatic: True
  mode: single
- alias: Almacenar acceso por movimiento
  variables:
    snapshot_filename: /config/photos/snapshot_{{ now().strftime('%Y-%m-%dT%H.%M.%S') }}.jpg
    record_filename: /config/videos/record_{{ now().strftime('%Y-%m-%dT%H.%M.%S') }}.mp4
  trigger:
  - platform: state
    entity_id: binary_sensor.movimiento_dormitorio
    to: on
  action:
  - service: notify.telegram
    data:
      message: MOVIMIENTO DETECTADO. Enviando vídeo...
  - service: camera.snapshot
    data:
      entity_id: camera.camara_dormitorio
      filename: '{{ snapshot_filename }}'
  - service: camera.record
    data:
      entity_id: camera.camara_dormitorio
      filename: '{{ record_filename }}'
      duration: 8
  - delay: 00:00:20
  - service: notify.telegram
    data:
```

```
64.       message: MOVIMIENTO DETECTADO
65.       data:
66.         photo:
67.         - file: '{{ snapshot_filename }}'
68.           caption: '{{ snapshot_filename }}'
69.   - service: shell_command.store_access
70.     data_template:
71.       location: Dormitorio
72.       snapshot_path: '{{ snapshot_filename }}'
73.       record_path: '{{ record_filename }}'
74.       automatic: True
75.   mode: single
76. - alias: Almacenar acceso por ruido
77.   variables:
78.     snapshot_filename: /config/photos/snapshot_{{ now().strftime('%Y-%m-
    %dT%H.%M.%S') }}.jpg
79.     record_filename: /config/videos/record_{{ now().strftime('%Y-%m-
    %dT%H.%M.%S') }}.mp4
80.   trigger:
81.   - platform: state
82.     entity_id: binary_sensor.ruido_dormitorio
83.     to: on
84.   action:
85.   - service: notify.telegram
86.     data:
87.       message: RUIDO DETECTADO. Enviando vídeo...
88.   - service: camera.snapshot
89.     data:
90.       entity_id: camera.camara_dormitorio
91.       filename: '{{ snapshot_filename }}'
92.   - service: camera.record
93.     data:
94.       entity_id: camera.camara_dormitorio
95.       filename: '{{ record_filename }}'
96.       duration: 8
97.   - delay: 00:00:20
98.   - service: notify.telegram
99.     data:
100.        message: RUIDO DETECTADO
101.        data:
102.          video:
103.          - file: '{{ record_filename }}'
104.            caption: '{{ record_filename }}'
105.    - service: shell_command.store_access
106.      data_template:
107.        location: Dormitorio
108.        snapshot_path: '{{ snapshot_filename }}'
109.        record_path: '{{ record_filename }}'
110.        automatic: True
111.  - alias: Arrancar bot Telegram
112.    variables:
113.      query_access: "{{ '\U0001F50E' }} Consultar accesos"
114.      manage_access: "{{ '\U0000270F' }} Administrar accesos"
115.      live_streaming: "{{ '\U0001F3A5' }} Live streaming"
116.      control_ptz: "{{ '\U00002195' }} Controlar PTZ"
117.    trigger:
118.    - platform: event
119.      event_type: telegram_command
120.      event_data:
121.        command: /start
122.    - platform: event
123.      event_type: telegram_command
124.      event_data:
125.        command: /menu
126.    action:
127.    - service: notify.telegram
128.      data:
129.        message: "Bienvenido a <b>PresenceHC1Bot</b>!
130.
```

```
131.         Este bot te permite realizar un control de accesos mediante un sistema de v
    igilancia doméstica.
132.
133.         Aquí recibirás notificaciones cuando se detecte presencia en tu hogar y pod
    rás consultar los datos de cada acceso.
134.
135.         Presiona <b>{{ query_access }}</b> para ver los comandos de consulta de acc
    esos.
136.
137.         Presiona <b>{{ manage_access }}</b> para ver los comandos de administración
     de accesos.
138.
139.         Presiona <b>{{ live_streaming }}</b> para capturar un vídeo en tiempo real
    de la cámara seleccionada.
140.
141.         Presiona <b>{{ control_ptz }}</b> para ver los comandos de control PTZ de l
    a cámara."
142.         data:
143.           inline_keyboard:
144.             - '{{ query_access }}:/query_menu'
145.             - '{{ manage_access }}:/manage_menu'
146.             - '{{ live_streaming }}:/live_menu'
147.             - '{{ control_ptz }}:/ptz_menu'
148.     mode: single
149.   - alias: Consultar accesos
150.     trigger:
151.     - platform: event
152.       event_type: telegram_callback
153.       event_data:
154.         command: /query_menu
155.     action:
156.     - service: notify.telegram
157.       data:
158.         message: "Para realizar una consulta de accesos usa uno de los siguientes c
    omandos:
159.
160.         /getaccessbyid [id] - Obtiene el acceso con el <b>ID</b> proporcionado como
     argumento.
161.
162.         /getaccessbycode [code] - Obtiene el acceso con el <b>código</b> proporcion
    ado como argumento.
163.
164.         /getaccessbydate [date] - Obtiene los accesos con la <b>fecha</b> proporcio
    nada como argumento. El formato a seguir debe ser dd-MM-YYYY.
165.
166.         /getaccessbydaterange [datetime_from] [datetime_to] - Obtiene los accesos c
    on el <b>rango de fecha</b> proporcionado como argumento. El formato a seguir debe
    ser dd-MM-YYYYTHH:mm:ss.
167.
168.         /getaccessbylocation [location] - Obtiene los accesos con la <b>ubicación</
    b> proporcionada como argumento.
169.
170.         /getaccessbyfilename [filename] - Obtiene el acceso que contiene el <b>nomb
    re de archivo</b> proporcionado como argumento.
171.
172.
173.         /getautomaticaccess [automatic] - Obtiene los accesos <b>automáticos</b> o
    manuales, según el argumento booleano proporcionado. Los valores pueden ser: <b>0</
    b>, <b>1</b>, <b>false</b>, <b>true</b>, <b>f</b>, <b>t</b>, <b>yes</b>, <b>no</b>,
     <b>y</b>, <b>n</b>.
174.
175.         /getsnapshotbyaccessid [id] - Obtiene el snapshot de la cámara para el acce
    so con el <b>ID</b> proporcionado como argumento.
176.
177.         /getsnapshotbyfilename [filename] - Obtiene el snapshot de la cámara para e
    l acceso que contiene el <b>nombre de archivo</b> proporcionado como argumento.
178.
179.         /getrecordbyaccessid [id] - Obtiene la grabación de la cámara para el acces
    o con el <b>ID</b> proporcionado como argumento.
180.
```

```
181.         /getrecordbyfilename [filename] - Obtiene la grabación de la cámara para el
     acceso que contiene el <b>nombre de archivo</b> proporcionado como argumento."
182.     mode: single
183.   - alias: Obtener acceso por identificador
184.     trigger:
185.     - platform: event
186.       event_type: telegram_command
187.       event_data:
188.         command: /getaccessbyid
189.     action:
190.     - service: shell_command.get_access_by_id
191.       data_template:
192.         id: "{{ trigger.event.data['args'][0] }}"
193.     - service: notify.telegram
194.       data:
195.         message: "{{ state_attr('sensor.access', 'message') }}"
196.     mode: single
197.   - alias: Obtener acceso por código
198.     trigger:
199.     - platform: event
200.       event_type: telegram_command
201.       event_data:
202.         command: /getaccessbycode
203.     action:
204.     - service: shell_command.get_access_by_code
205.       data_template:
206.         code: "{{ trigger.event.data['args'][0] }}"
207.     - service: notify.telegram
208.       data:
209.         message: "{{ state_attr('sensor.access', 'message') }}"
210.     mode: single
211.   - alias: Obtener acceso por fecha
212.     trigger:
213.     - platform: event
214.       event_type: telegram_command
215.       event_data:
216.         command: /getaccessbydate
217.     action:
218.     - service: shell_command.get_access_by_date
219.       data_template:
220.         from_date: "{{ trigger.event.data['args'][0] }}"
221.     - service: notify.telegram
222.       data:
223.         message: "{{ state_attr('sensor.access', 'message') }}"
224.     mode: single
225.   - alias: Obtener acceso por rango de fecha
226.     trigger:
227.     - platform: event
228.       event_type: telegram_command
229.       event_data:
230.         command: /getaccessbydaterange
231.     action:
232.     - service: shell_command.get_access_by_date_range
233.       data_template:
234.         from_date: "{{ trigger.event.data['args'][0] }}"
235.         to_date: "{{ trigger.event.data['args'][1] }}"
236.     - service: notify.telegram
237.       data:
238.         message: "{{ state_attr('sensor.access', 'message') }}"
239.     mode: single
240.   - alias: Obtener acceso por ubicación
241.     trigger:
242.     - platform: event
243.       event_type: telegram_command
244.       event_data:
245.         command: /getaccessbylocation
246.     action:
247.     - service: shell_command.get_access_by_location
248.       data_template:
249.         location: "{{ trigger.event.data['args'][0] }}"
```

```
250.     - service: notify.telegram
251.       data:
252.         message: "{{ state_attr('sensor.access', 'message') }}"
253.     mode: single
254.   - alias: Obtener acceso por nombre de archivo
255.     trigger:
256.     - platform: event
257.       event_type: telegram_command
258.       event_data:
259.         command: /getaccessbyfilename
260.     action:
261.     - service: shell_command.get_access_by_filename
262.       data_template:
263.         file: "{{ trigger.event.data['args'][0] }}"
264.     - service: notify.telegram
265.       data:
266.         message: "{{ state_attr('sensor.access', 'message') }}"
267.     mode: single
268.   - alias: Obtener accesos automáticos
269.     trigger:
270.     - platform: event
271.       event_type: telegram_command
272.       event_data:
273.         command: /getautomaticaccess
274.     action:
275.     - service: shell_command.get_automatic_access
276.       data_template:
277.         automatic: "{{ trigger.event.data['args'][0] }}"
278.     - service: notify.telegram
279.       data:
280.         message: "{{ state_attr('sensor.access', 'message') }}"
281.     mode: single
282.   - alias: Obtener snapshot por ID de acceso
283.     trigger:
284.     - platform: event
285.       event_type: telegram_command
286.       event_data:
287.         command: /getsnapshotbyaccessid
288.     action:
289.     - service: shell_command.get_snapshot_by_access_id
290.       data_template:
291.         id: "{{ trigger.event.data['args'][0] }}"
292.     - service: notify.telegram
293.       data:
294.         message: OBTENIDO SNAPSHOT
295.         data:
296.           photo:
297.             - file: "{{ state_attr('sensor.access', 'message') }}"
298.               caption: "{{ state_attr('sensor.access', 'message') }}"
299.     mode: single
300.   - alias: Obtener snapshot por nombre de archivo
301.     trigger:
302.     - platform: event
303.       event_type: telegram_command
304.       event_data:
305.         command: /getsnapshotbyfilename
306.     action:
307.     - service: shell_command.get_snapshot_by_filename
308.       data_template:
309.         file: "{{ trigger.event.data['args'][0] }}"
310.     - service: notify.telegram
311.       data:
312.         message: OBTENIDO SNAPSHOT
313.         data:
314.           photo:
315.             - file: "{{ state_attr('sensor.access', 'message') }}"
316.               caption: "{{ state_attr('sensor.access', 'message') }}"
317.     mode: single
318.   - alias: Obtener grabación por ID de acceso
319.     trigger:
```


```
320.     - platform: event
321.       event_type: telegram_command
322.       event_data:
323.         command: /getrecordbyaccessid
324.     action:
325.     - service: shell_command.get_record_by_access_id
326.       data_template:
327.         id: "{{ trigger.event.data['args'][0] }}"
328.     - service: notify.telegram
329.       data:
330.         message: OBTENIDA GRABACIÓN
331.         data:
332.           video:
333.             - file: "{{ state_attr('sensor.access', 'message') }}"
334.               caption: "{{ state_attr('sensor.access', 'message') }}"
335.     mode: single
336.   - alias: Obtener grabación por nombre de archivo
337.     trigger:
338.     - platform: event
339.       event_type: telegram_command
340.       event_data:
341.         command: /getrecordbyfilename
342.     action:
343.     - service: shell_command.get_record_by_filename
344.       data_template:
345.         file: "{{ trigger.event.data['args'][0] }}"
346.     - service: notify.telegram
347.       data:
348.         message: OBTENIDA GRABACIÓN
349.         data:
350.           video:
351.             - file: "{{ state_attr('sensor.access', 'message') }}"
352.               caption: "{{ state_attr('sensor.access', 'message') }}"
353.     mode: single
354.   - alias: Administrar accesos
355.     trigger:
356.     - platform: event
357.       event_type: telegram_callback
358.       event_data:
359.         command: /manage_menu
360.     action:
361.     - service: notify.telegram
362.       data:
363.         message: "Para administrar los accesos usa uno de los siguientes comandos:
364.
365.         /deleteaccessbyid [id] - Elimina el acceso con el <b>ID</b> proporcionado c
    omo argumento.
366.
367.         /deleteaccessbydate [date] - Elimina los accesos con la <b>fecha</b> propor
    cionada como argumento. El formato a seguir debe ser YYYY/MM/dd.
368.
369.         /deleteaccessbydaterange [datetime_from] [datetime_to] - Elimina los acceso
    s con el <b>rango de fecha</b> proporcionado como argumento. El formato a seguir de
    be ser YYYY/MM/ddTHH:mm:ss.
370.
371.         /clear: Reinicia completamente la base de datos, eliminando todos los acces
    os guardados hasta la fecha."
372.     mode: single
373.   - alias: Eliminar acceso por ID
374.     trigger:
375.     - platform: event
376.       event_type: telegram_command
377.       event_data:
378.         command: /deleteaccessbyid
379.     action:
380.     - service: shell_command.delete_access_by_id
381.       data_template:
382.         id: "{{ trigger.event.data['args'][0] }}"
383.     - service: notify.telegram
384.       data:
```

```
385.        message: "{{ state_attr('sensor.access', 'message') }}"
386.    mode: single
387.  - alias: Eliminar acceso por fecha
388.    trigger:
389.    - platform: event
390.      event_type: telegram_command
391.      event_data:
392.        command: /deleteaccessbydate
393.    action:
394.    - service: shell_command.delete_access_by_date
395.      data_template:
396.        from_date: "{{ trigger.event.data['args'][0] }}"
397.    - service: notify.telegram
398.      data:
399.        message: "{{ state_attr('sensor.access', 'message') }}"
400.    mode: single
401.  - alias: Eliminar acceso por rango de fecha
402.    trigger:
403.    - platform: event
404.      event_type: telegram_command
405.      event_data:
406.        command: /deleteaccessbydaterange
407.    action:
408.    - service: shell_command.delete_access_by_date_range
409.      data_template:
410.        from_date: "{{ trigger.event.data['args'][0] }}"
411.        to_date: "{{ trigger.event.data['args'][1] }}"
412.    - service: notify.telegram
413.      data:
414.        message: "{{ state_attr('sensor.access', 'message') }}"
415.    mode: single
416.  - alias: Reiniciar base de datos
417.    trigger:
418.    - platform: event
419.      event_type: telegram_command
420.      event_data:
421.        command: /clear
422.    action:
423.    - service: shell_command.clear
424.      data_template:
425.        id: "{{ trigger.event.data['args'][0] }}"
426.    - service: notify.telegram
427.      data:
428.        message: "{{ state_attr('sensor.access', 'message') }}"
429.    mode: single
430.  - alias: Menú Live streaming
431.    variables:
432.      snapshot_filename: /config/photos/snapshot_{{ now().strftime('%Y-%m-
    %dT%H.%M.%S') }}.jpg
433.      record_filename: /config/videos/record_{{ now().strftime('%Y-%m-
    %dT%H.%M.%S') }}.mp4
434.    trigger:
435.    - platform: event
436.      event_type: telegram_callback
437.      event_data:
438.        command: /live_menu
439.    action:
440.    - service: notify.telegram
441.      data:
442.        message: "Para realizar un live streaming de una cámara determinada usa el
    siguiente comando:
443.
444.        /live [location] - Obtiene un vídeo de corta duración desde el tiempo actua
    l de la <b>ubicación</b> proporcionada como argumento.
445.
446.        Esto almacenará un acceso manual (<b>automatic</b> = False) de la ubicación
     especificada con sus respectivos archivos."
447.    mode: single
448.  - alias: Live streaming
449.    variables:
```


```
450.      snapshot_filename: /config/photos/snapshot_{{ now().strftime('%Y-%m-
    %dT%H.%M.%S') }}.jpg
451.      record_filename: /config/videos/record_{{ now().strftime('%Y-%m-
    %dT%H.%M.%S') }}.mp4
452.      camera_location: "{{ trigger.event.data['args'][0].lower() }}"
453.    trigger:
454.    - platform: event
455.      event_type: telegram_command
456.      event_data:
457.        command: /live
458.    action:
459.    - service: notify.telegram
460.      data:
461.        message: CAPTURANDO STREAMING CÁMARA. Enviando vídeo...
462.    - service: camera.snapshot
463.      data:
464.        entity_id: 'camera.camara_{{ camera_location }}'
465.        filename: '{{ snapshot_filename }}'
466.    - service: camera.record
467.      data:
468.        entity_id: 'camera.camara_{{ camera_location }}'
469.        filename: '{{ record_filename }}'
470.        duration: 8
471.    - delay: 00:00:20
472.    - service: notify.telegram
473.      data:
474.        title: Video
475.        message: CAPTURANDO STREAMING CÁMARA
476.        data:
477.          video:
478.          - file: '{{ record_filename }}'
479.            caption: '{{ record_filename }}'
480.    - service: shell_command.store_access
481.      data_template:
482.        location: '{{ camera_location | capitalize }}'
483.        snapshot_path: '{{ snapshot_filename }}'
484.        record_path: '{{ record_filename }}'
485.        automatic: False
486.    mode: single
487.  - alias: Menú controlar PTZ
488.    trigger:
489.    - platform: event
490.      event_type: telegram_callback
491.      event_data:
492.        command: /ptz_menu
493.    action:
494.    - service: notify.telegram
495.      data:
496.        message: "Para realizar un movimiento PTZ de la cámara seleccionada usa uno
     de los siguientes comandos:
497.
498.        /goleft [location] - Rota hacia la izquierda la cámara en la <b>ubicación</
    b> proporcionada como argumento.
499.
500.        /goright [location] - Rota hacia la derecha la cámara en la <b>ubicación</b
    > proporcionada como argumento.
501.
502.
503.        /goup [location] - Rota hacia arriba la cámara en la <b>ubicación</b> propo
    rcionada como argumento.
504.
505.        /godown [location] - Rota hacia abajo la cámara en la <b>ubicación</b> prop
    orcionada como argumento.
506.
507.        Cuando el movimiento de rotación haya sido completado se enviará una imagen
     de prueba para que el usuario verifique el posicionamiento actual de la cámara."
508.    mode: single
509.  - alias: Movimiento de rotación cámara en dirección izquierda
510.    variables:
```

```
511.      snapshot_filename: /config/photos/snapshot_{{ now().strftime('%Y-%m-
    %dT%H.%M.%S') }}_test.jpg
512.      camera_location: "{{ trigger.event.data['args'][0].lower() }}"
513.    trigger:
514.    - platform: event
515.      event_type: telegram_command
516.      event_data:
517.        command: /goleft
518.    action:
519.    - service: shell_command.move_camera
520.      data:
521.        user: heccasa1@inf.upv.es
522.        password: ezviz1234
523.        location: '{{ camera_location }}'
524.        direction: left
525.    - delay: 00:00:10
526.    - service: camera.snapshot
527.      data:
528.        entity_id: 'camera.camara_{{ camera_location }}'
529.        filename: '{{ snapshot_filename }}'
530.    - service: notify.telegram
531.      data:
532.        message: OBTENIDO SNAPSHOT
533.        data:
534.          photo:
535.            - file: '{{ snapshot_filename }}'
536.            - caption: Imagen de prueba PTZ
537.    mode: single
538.  - alias: Movimiento de rotación cámara en dirección derecha
539.    variables:
540.      snapshot_filename: /config/photos/snapshot_{{ now().strftime('%Y-%m-
    %dT%H.%M.%S') }}_test.jpg
541.      camera_location: "{{ trigger.event.data['args'][0].lower() }}"
542.    trigger:
543.    - platform: event
544.      event_type: telegram_command
545.      event_data:
546.        command: /goright
547.    action:
548.    - service: shell_command.move_camera
549.      data:
550.        user: heccasa1@inf.upv.es
551.        password: ezviz1234
552.        location: '{{ camera_location }}'
553.        direction: right
554.    - delay: 00:00:10
555.    - service: camera.snapshot
556.      data:
557.        entity_id: 'camera.camara_{{ camera_location }}'
558.        filename: '{{ snapshot_filename }}'
559.    - service: notify.telegram
560.      data:
561.        message: OBTENIDO SNAPSHOT
562.        data:
563.          photo:
564.            - file: '{{ snapshot_filename }}'
565.            - caption: Imagen de prueba PTZ
566.    mode: single
567.  - alias: Movimiento de rotación cámara en dirección arriba
568.    variables:
569.      snapshot_filename: /config/photos/snapshot_{{ now().strftime('%Y-%m-
    %dT%H.%M.%S') }}_test.jpg
570.      camera_location: "{{ trigger.event.data['args'][0].lower() }}"
571.    trigger:
572.    - platform: event
573.      event_type: telegram_command
574.      event_data:
575.        command: /goup
576.    action:
577.    - service: shell_command.move_camera
```

```
      data:
        user: heccasa1@inf.upv.es
        password: ezviz1234
        location: '{{ camera_location }}'
        direction: up
    - delay: 00:00:10
    - service: camera.snapshot
      data:
        entity_id: 'camera.camara_{{ camera_location }}'
        filename: '{{ snapshot_filename }}'
    - service: notify.telegram
      data:
        message: OBTENIDO SNAPSHOT
        data:
          photo:
            - file: '{{ snapshot_filename }}'
            - caption: Imagen de prueba PTZ
    mode: single
  - alias: Movimiento de rotación cámara en dirección abajo
    variables:
      snapshot_filename: /config/photos/snapshot_{{ now().strftime('%Y-%m-
    %dT%H.%M.%S') }}_test.jpg
      camera_location: "{{ trigger.event.data['args'][0].lower() }}"
    trigger:
    - platform: event
      event_type: telegram_command
      event_data:
        command: /godown
    action:
    - service: shell_command.move_camera
      data:
        user: heccasa1@inf.upv.es
        password: ezviz1234
        location: '{{ camera_location }}'
        direction: down
    - delay: 00:00:10
    - service: camera.snapshot
      data:
        entity_id: 'camera.camara_{{ camera_location }}'
        filename: '{{ snapshot_filename }}'
    - service: notify.telegram
      data:
        message: OBTENIDO SNAPSHOT
        data:
          photo:
            - file: '{{ snapshot_filename }}'
            - caption: Imagen de prueba PTZ
    mode: single
```