Trabajo Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Diseño y realización de un sistema de telecontrol de dispositivos eléctricos para simulación de presencia en el hogar

Autor: Pedro Tomás Martínez Flores
Tutor: Manuel Ángel Perales Esteve

Dpto. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2023

Proyecto Fin de Carrera
Ingeniería Electrónica, Robótica y Mecatrónica

Diseño y realización de un sistema de telecontrol de dispositivos eléctricos para simulación de presencia en el hogar

Autor:
Pedro Tomás Martínez Flores

Tutor:
Manuel Ángel Perales Esteve
Profesor titular

Dpto. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2023

Proyecto Fin de Carrera: Diseño y realización de un sistema de telecontrol de dispositivos eléctricos para simulación de presencia en el hogar

Autor: Pedro Tomás Martínez Flores

Tutor: Manuel Ángel Perales Esteve

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2023

El secretario del Tribunal

Agradecimientos

Muchas gracias a Manuel Perales por su ayuda realizando este proyecto. Gracias también a mi familia, que ha estado apoyándome en esta época tan ajetreada, si la cual no se si podría haber llevado todo adelante.

Además, agradezco también a mis amigos y compañeros por ayudarme durante todos estos años.

Sinceramente, muchas gracias.

Pedro Tomás Martínez Flores

Alumno del Grado de Ingeniería Electrónica, Robótica y Mecatrónica

Sevilla, 2023

Resumen

La época de vacaciones es cuando más robos y allanamientos se dan. Y, realmente, los ladrones no son tontos. Aprovechan esta época del año, pues es cuando suele haber más casas solas, mientras sus dueños se encuentran de viaje.

El objetivo de este trabajo es aprovechar todas las ventajas que los avances de hoy en día se pone a disposición de las personas, desde la comunicación global que ofrece Internet, hasta las posibilidades de cómputo que ofrecen placas microcontroladoras gracias a la miniaturización de componentes electrónicos, para crear una solución al problema mencionado, creando un simulador de presencia en casa, es decir, un sistema que sea capaz de manejar la activación y desactivación de dispositivos de manera remota.

Además, se ha hecho hincapié en la cuestión de la seguridad, pues siendo el objetivo el de aumentar la seguridad de las casas, es necesario realizar protecciones extra para no exponer vulnerabilidades en la red o para posibles accidentes, lo cual sería ciertamente contradictorio.

Abstract

The vacation season of the year is one of most burglaries are made. And this is because the thieves take advantage of the amount of lone houses while their owners are off.

The target of this project is to use the advantages that there are available now a days to the people, from the global communications of Internet, to the compute capabilities that microcontrollers offer thanks of the miniaturization of electronic components, to create a solution, making a house presence simulator, a system that can activate and deactivate devices on the remote way.

Furthermore, emphasis has been placed on the security matter, because the objective has been enhancing the security of houses. It is necessary to do extra protections to protect the vulnerabilities online and from occasionally accidents, a thing that would be contradictory.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

2FA: Two Factor Authentication	9, 16
ADC: Analog-digital Converter	18, 20
AIO: Adafruit IO	6, 7, 8, 11, 13, 15, 16, 23
API: Application Programming Interface	11, 13, 15
HOTP: HMAC One time Password	11, 12
HTML: HyperText Markup Language	16
HTTP: Hypertext Transfer Protocol	7
IoT: Internet of Things	1, 6, 7, 8
IPS: In-Plane Switching	23
json: JavaScript Object Notation	14, 15, 16
LCD: Liquid Crystal Display	23
mCU: Microcontrolador	5, 6, 8, 9, 13, 14, 15, 16, 17, 18, 21, 23, 24, 25, 28
MQTT: Message Queue Telemetry Transport	7, 8, 11, 24
OPAMP: OPerational AMPlifier	18
OTP: One-Time Password	11
QoS: Quality of Service	7, 8
QR: Quick Response	9, 14, 23, 25
RP: Raspberry	21, 31
SPI: Serial Peripheral Interface	21
TOTP: Time-based one-time password	9, 11, 12, 13, 14, 16, 23, 24, 25, 33
URL: Uniform Resource Identifier	14
WiFi: Wireless Fidelity	15, 16

1 INTRODUCCIÓN

Una de las mayores preocupaciones que todo el mundo ha tenido alguna vez cuando está de vacaciones es sobre el hogar. Es un momento donde la casa se encuentra sola durante bastante tiempo. Y los ladrones saben esto. De hecho, el periodo vacacional es uno donde más robos en casas se producen[1]. Por otro lado, el avance de la interconexión de todo tipo de aparatos: luces, electrodomésticos o cepillos de dientes, es una tendencia que se puede aprovechar en nuestro beneficio. Para aumentar la seguridad de nuestras casas en periodos vacacionales de mayor vulnerabilidad, el objetivo del trabajo es construir un sistema automatizado que sea capaz de manejar dispositivos, encendiendo y apagándose según un horario configurable a través de internet, además de añadir cierta capa de seguridad extra en la configuración de este Simulador de Presencia.

El avance de la tecnología y la miniaturización de la electrónica ha permitido llevar dispositivos capaces de hacer cosas impensables hace solo unas décadas en nuestro bolsillo. Somos capaces de, no solo llamar, sino también de buscar por Internet, gestionar el calendario, mirar el correo electrónico, escuchar música, ver vídeos online, hacer fotografías y hasta jugar con un aparato que ha sustituido, en muchas ocasiones, a un ordenador personal.

Además, el acceso global a Internet ha revolucionado nuestra forma de vida. Toda la información del mundo, noticias, entretenimiento y, además, servicios online que permiten la extensión de la manera en la que vivimos a la red. Esto tiene numerosas ventajas, pues permite la comunicación global de personas en tiempo real, independientemente del lugar donde se encuentre. Esta característica en concreto ha sido muy importante recientemente debido a la pandemia del COVID-19, donde nos vimos todos encerrados en casa, y obligados a comunicarnos a través de estos medios.

Sin embargo, el gran vuelco de nuestra vida en Internet nos hace vulnerables. Exponemos gran cantidad de información en línea sin percatarnos realmente de que todo el mundo puede acceder a ella, personas cercanas y en quienes confiamos; y personas externas con intenciones maliciosas. Actualmente es relativamente sencillo saber cómo se llama alguien, a qué se dedica, dónde vive y hasta cuándo se va de vacaciones. Esta información es sensible y puede ser aprovechada por quién no queremos.

Otra rama de la seguridad en Internet es la ciberseguridad, muy importante hoy en día, con la gran cantidad de gestiones online que se realizan diariamente, ya sea una transacción bancaria, o simplemente mirar nuestro correo electrónico.

Aunque, atendiendo al objetivo de este trabajo, es también conveniente centrarse en peligros pre-Internet. Es conocido que los ladrones estudian a sus víctimas y sus horarios antes de asestar algún golpe. Y uno de los momentos más vulnerables de nuestra casa es cuando estamos de vacaciones, pues los ladrones tienen "vía libre" para poder acceder a nuestras propiedades.

El objetivo de este trabajo es, como se mencionó antes, el de idear una solución a dicho problema, aprovechando las ventajas que hoy en día ofrece Internet, integrando en nuestro día a día la interconectividad de los aparatos, el llamado IoT, y mejorando la seguridad de nuestras casas en los momentos de mayor vulnerabilidad. Por otro lado, también se pretende dar un paso en la ciberseguridad, implementando sistemas de doble factor de autenticación para mejorar la seguridad de nuestro hogar y de nuestras interacciones en línea.

1.1 Especificaciones

Ahora, se detallarán las especificaciones que buscamos en el dispositivo que vamos a desarrollar.

- Controlar la activación de aparatos eléctricos y/o electrónicos
- Controlar la configuración de estos aparatos de manera remota
- Configurar 2 tramos horarios independientes.
- Controlar 2 dispositivos de manera independiente
- Mostrar información del sistema en una pantalla
- Incorporar seguridad extra en la comunicación remota
- Incorporar seguridad a nivel físico

1.2 Posibles soluciones comerciales

En primer lugar, la mayoría de las soluciones que se encuentran en el mercado para simular presencia de personas en casas durante temporadas vacacionales son propuestas de empresas dedicadas al mundo de la domótica y, que explican en qué consiste sus servicios y ofrecen sus soluciones.

Otra opción más sencilla es el uso de bombillas o interruptores inteligentes, capaces de conectarse a internet y ser controlados desde un punto de control centralizado. Para el propósito principal de este trabajo, el encender y apagar luces, existen varias opciones:

1.2.1 Bombillas inteligentes

Las bombillas inteligentes son, como su propio nombre indica, bombillas capaces de conectarse a internet para ofrecer funcionalidades a través de este.

La función que primero se nos viene a la mente es la de encender y apagar estas luces, pero las hay con muchas más capacidades, como poder elegir la luz emitida con un selector RGB o poder modificar la luminosidad de esta. Además, la mayoría de los fabricantes ya desarrollan ecosistemas que integran todos los aparatos inteligentes de nuestra casa para poder gestionarlos. Es en estos centros donde se pueden manejar las bombillas y planear rutinas de encendido y apagado de esta.

Hay que tener en cuenta también que existen bombillas que usan protocolos de conectividad específicos y que necesitan un hub o puente como intermediario entre el dispositivo y nuestro control.

Uno de los inconvenientes de este tipo de soluciones es que, al reemplazar las bombillas originales, las cuales estaban instaladas en el sistema eléctrico de la casa, necesitan que los interruptores se encuentren encendidos en todo momento. Y no es posible sincronizar el funcionamiento del interruptor con el de la bombilla, es decir, el interruptor de la pared pasa a ser un aparato que no puede volver a tocarse nunca.

Algunas de las marcas que ofrecen esta solución son: Philips, Xiaomi, Ikea, TP-Link, etc.

Figura 1.1: Bombilla Inteligente[33]

1.2.2 Interruptores inteligentes

Otra alternativa son los interruptores inteligentes. Estos son elementos conectados a internet o BlueTooth que encienden y apagan luces de manera remota. Al igual que las bombillas, puede ser capaces de realizar acciones automatizadas.

Su mayor ventaja frente a las bombillas inteligentes es que también permite encender y apagar las luces físicamente. Por otro lado, requiere una mayor complejidad en la instalación, pues es necesario sustituir el interruptor antiguo y colocar el nuevo en su caja empotrada.

Existe todo un mundo de interruptores, aunque muy parecido a las bombillas. Hay que tener en cuenta su conectividad o su facilidad de uso. Por otro lado, hay que tener en cuenta la instalación eléctrica de tu vivienda, pues en muchas ocasiones, el interruptor requiere de cable de neutro para funcionar.

Algunas de las marcas que ofrecen interruptores inteligentes son: Phillips, TP-link o Meross.

Figura 1.2: Interruptor Inteligente[34]

2 SOLUCIÓN ADOPTADA

Teniendo en cuenta todos los requerimientos mostrados anteriormente, se he decidido realizar el proyecto basándonos en un microcontrolador. Los mCUs son circuitos integrados programables, capaces de ejecutar acciones programadas de antemano[2]. La gran ventaja de estos es su versatilidad y su tamaño que, combinado con su precio, les dan a estos aparatos una gran versatilidad. Gracias a estas características, se plantea como una gran posible solución

Además. la potencia de estos últimamente a aumentado enormemente, permitiendo implementar soluciones de mayor nivel, pero pudiendo interactuar con el mundo físico gracias a sensores y actuadores.

De hecho, son necesarios relés, capaces de manejar circuitos eléctricos de altas tensiones con circuitos de control a poca tensión, y un par de enchufes para poder conectar los aparatos que nos interesa manejar. Estos son los actuadores del proyecto.

Además, se ha decidido usar una pantalla para poder comunicar información relevante a los usuarios. Esta también forma parte importante de la seguridad, pues será necesario leer de esta al iniciar el sistema.

2.1 Elección de Plataforma

El mCU usado es la Raspberry Pi pico W, versión mejorada de la Raspberry Pi pico, creada por la fundación Rasberry Pi. Esta placa de microcontroladora está basada en el RP2040, y está diseñada para ser una placa de desarrollo flexible y de bajo coste. La principal diferencia de esta versión W con su hermana pequeña es la incorporación de un módulo inalámbrico que permite la comunicación por WiFi y BlueTooth de manera sencilla[3].

La elección de esta plataforma de desarrollo viene principalmente por su acceso a Internet vía WiFi, además de que, al ser desarrollada por una entidad como la fundación Raspberry Pi, posee de una extensa documentación, lo cual es de gran ayuda para el desarrollo, facilitando su uso, así como de una gran comunidad que mejora también la facilidad de encontrar soluciones en línea.

Por otro lado, la gran potencia de estos mCUs les permite funcionar comuna variedad de lenguajes de programación. Entre ellos, C o Python, en alguna de sus adaptaciones[4]. De hecho, una de las características que más importancia tuvo a la hora de elegir este mCU frente a Arduino, por ejemplo, es justamente la posibilidad de usar Python en él, pues es un lenguaje más sencillo y fácil de usar, teniendo también una gran potencia. Y, aunque existan versiones de Arduino con Wifi, estas tienen un alto coste.

Por otro lado, existen también plataformas con capacidades muy parecidas a las de la Raspberry Pi pico W, como el famoso ESP32, que incorpora conectividad WiFi y BlueTooth, además de ser compatible también con las versiones adaptadas de Python. En este caso, la Raspberry ha sido la opción elegida por su precio.

A continuación, una tabla comparativa:

Tabla 1: Comparación microcontroladores

Microcontrolador	Precio [€]	WiFi	Python
Raspberry Pi Pico W	8,40[5]	Si	Si
ESP32	10[6]	Si	Si
Arduino UNO WiFi	46,70[7]	Si	No

2.2 Uso de CircuitPython

Por otro lado, la potencia que desarrollan los mCUs hoy en día ha hecho posible la adaptación de lenguajes de programación de más alto nivel a estos dispositivos. Una de las principales razones de mi elección de plataforma ha sido la posibilidad de usar Python en el mCU, pues me parece un lenguaje de programación sencillo y potente, además de ser muy utilizado a nivel empresarial en la actualidad.

Pero no es simplemente poner Python en un mCU. Hay que adaptar el funcionamiento de este lenguaje para que sea soportado por el procesador que incluyen los mCUs, además de añadir soporte para todas las funciones que este puede realizar, desde utilizar los pines de entrada y salida, tanto digitales como analógicos, como utilizar los módulos de Wifi o Bluetooth que algunos de los mCUs más famosos incluyen.

Un ejemplo de ello es MicroPython, una implementación de Python 3, optimizada para su uso en microcontroladores y con un grupo de los módulos estándar implementados, llevando la facilidad de uso de este lenguaje al mundo del hardware[8].

Sin embargo, para el curso de este trabajo, se ha decidido usar una variante de MicroPython creada por Adafruit: CircuitPython. Esta variante es preparada por Adafruit, incluyendo compatibilidad con gran cantidad de sus placas y otros dispositivos. Una de las razones por la que decidí el uso de CircuitPython fue la gran cantidad de documentación y tutoriales que se encuentran disponibles, una gran ayuda en el desarrollo, además del lote de librerías de Adafruit para CircuitPython, que incluye compatibilidad con gran cantidad de hardware extra. Esto es de mucha ayuda, sobre todo para principiantes que están comenzando con esta tecnología, como es mi caso[9].

Además, CircuitPython incluye un paquete de módulos muy parecidos a los originales de Python, lo que permite el traspaso de programas casi directamente desde una consola de Python nativa a una placa con CircuitPython funcionando. Esta es una gran ventaja a la hora de realizar pruebas y desarrollar aplicaciones en estas plataformas.

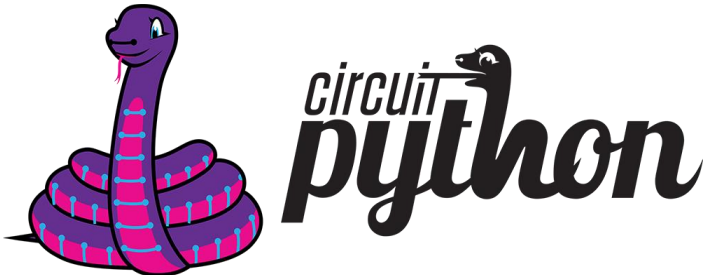

Figura 2.1: Logo de CircuitPython

2.3 Descripción de Adafruit IO

El manejo de nuestros dispositivos y proyectos desde la red es más complejo de lo que parece. Sería necesario crear un servidor, alojarlo en el mCU, configurar la red para que sea accesible desde el exterior de la red LAN, etc. Y para alguien con poca experiencia puede ser bastante tedioso.

Justamente para eliminar esta complejidad, Adafruit ofrece un servicio en la nube, donde se puede conectar tanto desde los mCUs o aparatos con acceso a internet como desde la web para el manejo del sistema. Adafruit IO (AIO) principalmente almacena y recupera información en la web. Pero la gran ventaja de AIO es su API, que permite el acceso a esta información desde cualquier dispositivo conectados a la red.

Por otra parte, Adafruit IO permite exponer de forma amigable para las personas la información que almacena en forma de dashboards (paneles de instrumentos) pudiendo mejorar la interactividad de las personas con los datos y con los dispositivos físicos. Este sistema es muy útil en proyectos de IoT.

Figura 2.2: Logo de Adafruit IO

Debido a que el lenguaje CircuitPython también es ofrecido por Adafruit, se ofrecen tutoriales, mucha documentación y librerías que aportan gran compatibilidad para la conexión de dispositivos al sistema en la nube[10].

3 CONEXIÓN A INTERNET

Como se ha mencionado anteriormente, una de las características principales del trabajo es la conexión a Internet. Pero en el mundo de la web, existen muchas maneras de comunicaciones, dependiendo de su utilidad, su rapidez o su seguridad. Estas maneras de comunicarse con los llamados Protocolos.

3.1 Adafruit IO

El servicio de Adafruit IO cuenta con una API con la que poder acceder a los datos almacenados, permitiendo leer y escribir en los feeds, almacenes donde se guarda un valor, ya sea numérico o de texto. Esta API está disponible para su uso tanto en HTTP, protocolo usado ampliamente en la actualidad en la navegación web, como en MQTT, este último método ha sido seleccionado para la realización del trabajo[10].

3.1.1 Limitaciones de Adafruit IO

El Sistema que ofrece Adafruit es muy útil para el Desarrollo de aplicaciones en dispositivos IoT. Pero no todo es perfecto.

AIO funciona con un plan Premium (IO+) de pago que da acceso ilimitado a las características que implementa. Pero también cuenta con una versión gratuita, con ciertas limitaciones[11]:

- 10 Feeds, es decir, la capacidad de almacenar hasta 10 variables
- 5 Dashboards
- Almacenamiento de datos online de 30 días
- Tasa de datos de 30 datos por minuto

Las mayores limitaciones son la de cantidad de feeds y la limitada tasa de datos.

Para poder hacer uso de la cuenta gratuita, ha sido necesario implementar una multiplexación de información, donde se usan los mismos feeds para poder modificar varios aspectos de nuestro dispositivo. Esta multiplexación es controlada por un feed extra, booleano, que permite seleccionar qué apartado estamos modificando.

Por otro lado, no se está actualizado los datos almacenados en AIO constantemente, para evitar flujos de datos innecesarios. Una vez cada minuto, se comprueba si es necesaria la actualización de estos valores. Esto limita ampliamente la cantidad de transacciones que se realizan, dejando margen suficiente a los usuarios para poder utilizar sin miedo a que se quede bloqueado los dashboards para manejar el aparato perfectamente.

3.2 Funcionamiento de MQTT

MQTT es un protocolo de comunicación Máquina-Máquina (M2M) basado en la pila TCP/IP. MQTT es muy utilizado hoy en día para proyectos de IoT.

Este protocolo funciona como un servicio de mensajería con un patrón de Publicador – Suscriptor. Los clientes se conectan a un servidor central llamado broker. En resumen, los clientes publican mensajes en temas (*topics*) organizados, y todos los clientes suscritos a dicho tema son notificados con la nueva información por el broker.

Además, el cliente y el broker se comunican constantemente para cerciorarse de que los clientes siguen presentes. Esto evita enviar mensajes a clientes que han sido desconectados o que han dejado de funcionar correctamente. Por otro lado, MQTT implementa un sistema de calidad del servicio (QoS), que asegura la

integridad de los mensajes, aportando robustez al servicio. Es posible distinguir 3 niveles de QoS;

- QoS 0 unacknowledged: Los mensajes solamente se envían una vez, pudiendo existir fallos cuando alguno de los mensajes no se entregue.
- QoS 1 acknowledged: Los mensajes se envían hasta que son recibidos, es decir, que hasta que el receptor no verifique que se ha entregado correctamente un mensaje, el emisor lo envía constantemente. Esto puede desencadenar que el receptor reciba varias veces el mismo mensaje.
- QoS 2 assured: Se garantiza que los mensajes son entregados solamente una vez. Este sistema incluye una mayor cantidad de mensajes de confirmación, lo que repercute en la carga del sistema.

En cuanto a seguridad, MQTT permite varias maneras de autenticación por usuario y contraseña o por certificados, pero teniendo en cuenta la escasa potencia de la mayoría de los dispositivos de IoT, la autenticación consta de usuario y contraseña enviada en texto plano. Esta carencia se puede suplir con una API key, una clave que permite confirmar que la conexión es del usuario indicado; y sistema de doble factor de autenticación 2FA. En el caso de este último sistema, se ha implementado externamente a la conexión MQTT, y permite asegurarse de que todas las interacciones del usuario son realmente del dueño del dispositivo.

Figura 3.1: Logo de MQTT

En definitiva, el protocolo MQTT es ampliamente usado en IoT debido a su sencillez y ligereza, consumiendo poco ancho de banda y el consumo de energía de los dispositivos es mínimo, además de las ventajas que aporta el utilizar un modelo de publicador-suscriptor, que permite desacoplar dispositivos y hacerlos independientes[12].

3.3 Implementación en Adafruit IO

Como se comentó anteriormente, la estructuración del sistema en AIO debe ser eficiente para poder funcionar con las limitaciones de las cuentas gratuitas. Esto aumenta la complejidad de esta estructura.

Principalmente, existen 9 feeds que permiten la comunicación entre el dispositivo y los usuarios. Esta comunicación es bidireccional, es decir, que cada cosa que se realiza en el dashboard se refleja en el mCU, quien responde para mantener la retroalimentación de la comunicación. Debido a las limitaciones anteriormente comentadas, la comunicación del mCU con AIO se realiza cada minuto, si es necesaria.

Estos feeds son:

- Comunicación Usuario -> mCU:
  - Hora Inicio
  - Hora Apagado
  - Hora Inicio 2
  - Hora Apagado 2
  - Selector Aparato
  - TOTP Input
- Comunicación mCU -> Usuario:
  - Text Messages
  - Estado Aparato 1
  - Estado Aparato 2

Como se puede observar, se usan los feeds de Horas de inicio/apagado para configurar las franjas horarias donde se desea que dicho aparato se encuentre encendido. Se puede modificar estor horarios del aparato 1 o 2 en función del valor del Selector de aparato. Además, los feeds de estados de aparatos informan al usuario de si, en ese momento, dicho aparato se encuentra encendido o apagado.

Además, el feed de mensajes es usado por el mCU para informar al usuario de los procesos que ocurren. Informa del horario de cada aparato en función del valor del selector y alerta de que el valor usado en la 2FA no es correcto. Como se indicó antes, estas comunicaciones no son inmediatas, sino que se realizan cada minuto, lo que permite descargar ampliamente el límite de tasa de datos establecido por Adafruit. Si bien los mensajes de realimentación son enviados cada minuto, las actualizaciones de los feeds de entrada del usuario son inmediatas, es decir, el mCU ejecuta una interrupción cada vez que el bróker informa de que se ha producido una modificación en uno de los topics a los que se ha suscrito. A fin de cuentas, el mCU ha de ser capaz de lidiar con esta información en cualquier momento que le llegue.

Para que el usuario pueda configurar los horarios a su gusto, lo único que debe hacer es insertar el código de 6 dígitos del TOTP. Mientras el valor que se ha colocado en este feed sea el correcto, se permitirán las modificaciones de los horarios que se indiquen mediante los feeds. Se debe respetar que la hora de inicio de un periodo de encendido debe de situarse antes que la hora de finalización de dicho periodo. Las claves del 2FA son válidas durante 30 segundos, es decir, que cada 30 segundos, la clave que se debe utilizar para la modificación de los horarios cambia. Esta clave es un factor extra de seguridad, pues para conocerla, se debe escanear un código QR que se muestra en el inicio del dispositivo.

4 SEGURIDAD

Una de las principales preocupaciones a la hora de elaborar este proyecto ha sido abordar un punto de vista seguro, es decir, aumentar las medidas de seguridad para permitir aumentar la tranquilidad de los usuarios.

Por un lado, el propio proyecto tiene un fin asegurador. El tener un sistema que sea capaz de encender y apagar luces automáticamente, en horas configurables tiene como objetivo simular la presencia de personas en casa cuando estas se encuentran vacías, principalmente en periodos vacacionales, lo cual ahuyenta a ladrones. Pero otro aspecto muy importante para tener en cuenta es la seguridad de la conexión. Al fin y al cabo, estamos controlando nuestra casa, y no es deseable que cualquiera pueda hacerlo.

Por este motivo, el protocolo MQTT, explicado en el punto anterior, implementa varios sistemas de seguridad para reforzar y asegurar las conexiones con el broker[12]. También hemos visto que estos sistemas aumentan la complejidad de uso, dejando de ser soluciones factibles para en dispositivos de poca potencia como los mCUs. Por este motivo, Adafruit en AIO ha creado una *API key* necesaria para poder acceder al broker. Esta API key es un código personal que identifica una conexión con el usuario dueño de dicha key[10].

Aun así, se puede seguir encontrando vulnerabilidades. AIO es muy beneficioso en cuanto a que puede ser manejado sencillamente desde su web, pero también implica que cualquier persona con acceso a dicha web tendría acceso. Y en la seguridad en la web se usa simplemente un usuario y contraseña.

Para poder suplir este "agujero" de seguridad, se ha implementado un sistema de verificación en 2 pasos. Este es, en concreto, el TOTP. Explicaremos su funcionamiento.

4.1 Algoritmos de seguridad

4.1.1 HOTP

La clave de la seguridad de los mecanismos de OTP es que dicha contraseña varia cada vez que es utilizada, en transacciones o inicios de sesión. Esto hace el sistema más resistentes a ataques de fuerza bruta, ya que, tras obtener una vez dicha contraseña, esta es cambiada y es necesario volver a realizar el ataque. Esto permite no abusar de una contraseña que ha sido obtenida[13].

El Sistema de verificación en 2 pasos HOTP es un algoritmo simple de verificación con contraseña única basado en el algoritmo HMAC-SHA-1[14], utilizado para la firma y verificación criptográfica de documentos, además de para verificar la integridad de este[15].

El HOTP se basa en una cuenta incremental y una clave simétrica estática conocida como token. La contraseña específica sería el resultado del algoritmo HASH-SHA-1, dando como datos de entrada el token y la clave incremental, truncado (de una forma específica) el resultado para obtener un código de 6 dígitos, fácil de utilizar por los usuarios[14].

$$HOTP(K, C) = Truncate^{1}(HMAC - SHA - 1(K, C))$$

4.1.2 TOTP

El TOTP es una vuelta de tuerca más al sistema de verificación con contraseña única. El HOTP usa un contador incremental para evaluar la contraseña a utilizar, en cambio, en el TOTP dicho contador se computa

[1] Función definida en la sección 5.3 de la RFC 4226[14]

en base al tiempo.

El contador se calcula como el número de pasos de tiempo que han pasado desde una referencia[16], establecida en el tiempo UNIX, el 1 de enero de 1970[17].

$$TOTP = HOTP(K,T)$$

$$T = (Current\ Unix\ time - T0) / X$$

Siendo T0 normalmente 0, y X el tamaño del paso usado.

Esta contraseña, por su propia naturaleza, caduca cada X segundos, y es importante utilizar un valor balanceado entre seguridad y facilidad de uso. Es recomendado utilizar 30 segundos por los creadores del algoritmo.

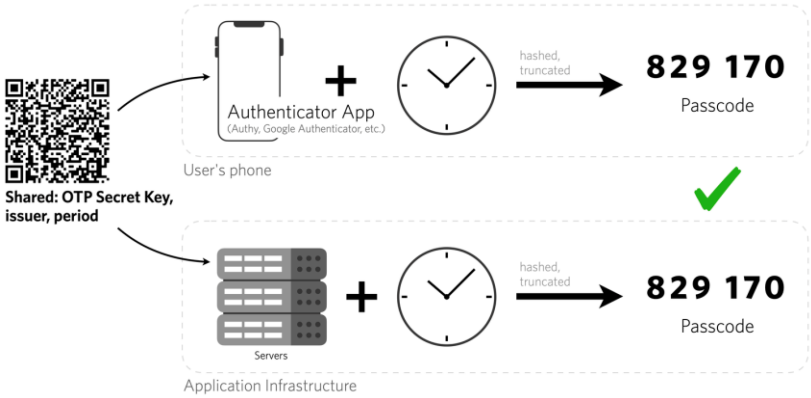

Figura 4.1: Esquema sobre TOTP[35]

Como es evidente, al ser un algoritmo que basa su seguridad en una clave o token de seguridad, que es compartido al inicio, este token debe ser generado aleatoriamente y guardado de forma segura, pues de este depende el secretismo del sistema.[16]

4.2 Implementación del TOTP

Realmente, como el algoritmo de TOTP se basa en una adaptación al tiempo del HOTP, es necesario implementar este algoritmo también y poner como entrada los datos correspondientes. A su vez, el HOTP usa algoritmos de encriptación como el HMAC y el HASH, siendo necesario el uso de estos para el funcionamiento del sistema.

En este caso, se ha utilizado una librería encontrada en GitHub de un proyecto (Picoth, AngainorDev[18]), la cual es una adaptación a CircuitPython que implementa TOTP. Esta adaptación se ha realizado con librerías del lote de librerías de Adafruit[19].

El funcionamiento básico es el siguiente:

- Se crea un objeto de tipo TOTP al que se le pasa el token.
- Se llama al método totp() que devuelve el valor del código de verificación a utilizar.

Siendo realmente sencillo, la parte más compleja viene a la hora de preparar el sistema para que funcione correctamente.

4.2.1 Sincronización con el tiempo UNIX

El tiempo UNIX actual es un valor importante a la hora de calcular el doble factor de autenticación necesario para la manipulación del simulador de presencia[16]. Por suerte, el mCU implementa un contador que establece dicho tiempo[4].

Pero, este contador se establece a 1 de enero de 2020 cada vez que se inicia[2], siendo desde aquí cuando empieza a contar. Esto no es funcional, ya que las aplicaciones que generan TOTP están sincronizadas con el tiempo UNIX actual, no el del mCU.

Por suerte, AIO ofrece un servicio por el cual devuelve el tiempo UNIX en el formato en el que se solicite[20]. En este caso, nos interesa el valor en segundos. Tras recibir esta información, se actualiza el valor del contador para sincronizarlo con el tiempo UNIX actual.

Tras esto, nos desuscribimos del topic correspondiente, pues no necesitaremos más esta información. Finalmente, es el propio método el que hace la llamada al módulo time para obtener dicho valor cuando lo requiera.

El único problema de este método de sincronización es que el sistema de tiempo UNIX está representado en la zona horaria GMT (UTC +0), distinta de la existente en España (UTC +1 o UTC +2). Esto no sería un problema si la zona horaria española fuese constante, ya que simplemente se tendría en cuenta a la hora de realizar el cálculo de la hora actual. Pero desde 1981, en España se cambia la hora 2 veces al año[21].

Para poder conocer la zona horaria o, mejor dicho, el desfase de la hora española con la GMT, se ha hecho uso de la API de *WorldTimeAPI* que, dado un país, se devuelve dicha información. Por supuesto, esta información es actualizada dependiendo de la época del año[22]. Simplemente hay que guardar este desfase en memoria para que pueda ser utilizado cuando sea necesario realizar el cálculo de la hora actual.

4.2.2 Generación del token de seguridad

Como se ha comentado antes, la generación aleatoria del token compartido es importante pues es donde reside la identificación y conexión del usuario con su acceso o transacción segura en internet.

Una opción inicial fue usar el módulo *random* incluido, pero este utiliza algoritmos pseudoaleatorios para generar sus números, pero no está recomendado para su uso en aplicaciones de seguridad.

Afortunadamente, CircuitPython incluye el módulo os, con el método *urandom()*, que genera una cadena de n bytes aleatorios haciendo uso de un generador de verdaderos números aleatorios implementado en hardware[23]. Gracias a esos valores aleatorios, se ha generado una contraseña con los caracteres de la Base 32, interpretando los bytes generados como números, obteniendo el módulo con 32 y utilizando este como índice en la lista con los posible valores. Finalmente, se une todo en una misma cadena y se devuelve.

Generación de token para TOTP

```
def Random_32_Key_generator(length: int):
    from os import urandom

    dict = "ABCDEFGHIJKLMNOPQRSTUVWXYZ234567"
    passw = []
    for i in range(length):
        random_number = int.from_bytes(urandom(1),"little")%32
        passw.append(dict[random_number])

    final_pass = ''.join(passw)
    return final_pass
```

[2] El valor original del contador del microcontrolador es 1577836800

La cadena devuelta es la que se terminará utilizando como token de seguridad en el TOTP.

Es importante que se cree un token seguro vy este debe de ser compartido con los usuarios que van a utilizar el simulador. Este proceso se realiza mediante la generación de un código QR que se muestra en la pantalla del Pico Display Pack[24]. Este QR codifica una url donde se introduce la información necesaria para que aplicaciones dedicadas a guardar estos tokens TOTP, como Google Authenticator, puedan leerlos y registrarlos[25]. Por suerte, este formato se ha convertido en un estándar de facto, es decir, que la mayoría de las aplicaciones lo aceptan.

El QR es generado en el propio microcontrolador, pues si se dependiese de servidores externos, no se garantizaría que el token se mantuviese secreto. Se ha hecho uso de la librería *Adafruit miniQR*[26], incorporada en el lote de módulos de Adafruit. Tras obtener la matriz correspondiente al QR, solamente se dibuja en pantalla con los colores blanco y negro, para que pueda ser leído.

El último aspecto para tener en cuenta en la seguridad de la clave compartida es dónde y cómo se guarda dicha clave. Este puede parecer uno de los puntos flojos del sistema, pues realmente, el token se acaba guardado en un archivo .json que se almacena en la memoria del mCU, en texto plano. El caso está en que esta clave se almacena localmente, y si se puede acceder físicamente al dispositivo para leer la clave, es que se ha accedido al interior de la vivienda, y ya no sería útil esconder dicha información.

5 MEMORIA

Como ya se ha especificado con anterioridad, el objetivo del simulador es poder programar el encendido y apagado de una serie de dispositivos a distancia. Y el mCU debe conocer cuándo debe o no encender el dispositivo, es decir, que debe de almacenar estos datos. A su vez, debe conocer cuál es la red wifi a la que se debe conectar cuando se encienda, con su contraseña. Por otro lado, es necesario que recuerde a que cuenta de AIO debe conectarse, incluyendo su API key, para poder autenticarse en el broker.

Todos estos datos pueden ser introducidos en el microcontrolador, pero si ocurre una pérdida de alimentación y se termina apagando el simulador, es incómodo que todos estos datos se perdieran, así como los propios horarios almacenados.

Para solucionar este problema, se almacenan los valores de las variables correspondientes en la memoria del mCU, para que pueda ser recuperada en caso de que sea necesario.

Este almacenaje se realiza por medio de .json, formato de texto sencillo creado con la finalidad de intercambiar datos entre programas[27]. Y estos pueden ser un programa y él mismo en el futuro. Gracias al módulo de Python e implementado en CircuitPython de json, la escritura y lectura de estos objetos en memoria es realmente sencilla.

5.1 Datos que almacenar

Los datos que son necesarios de almacenar son los siguientes:

- Horarios establecidos por el usuario.
- Credenciales de la conexión WiFi.
- Token generado para la autenticación en 2 Pasos
- Credenciales de conexión a Adafruit IO

Nada más iniciar el mCU, el sistema comprueba la existencia de los archivos correspondientes y, si existen, comprueba que los datos que contiene están completos. Si todo esto es correcto, sigue con su funcionamiento sin problemas.

Todas estas comprobaciones se realizan al inicio. Por suerte, en los microcontroladores con CircuitPython, el primer programa que se ejecuta cuando es iniciado es específicamente el llamado boot.py[28]. Por este motivo, todas esta parte de comprobar la existencia o no de la información necesaria para el funcionamiento del simulador se ha incluido en dicho boot.py. Según el resultado de las comprobaciones, se le indica al programa principal, main.py[28], que cargue el programa normal o el de lectura de datos por el usuario.

En caso de que falle alguno de los datos, se procede de la siguiente manera:

5.1.1 Horarios

Si el archivo .json que almacena esta información no esté presente (porque sea la primera vez que se ha iniciado, por ejemplo), o los datos que incluye no están completos, se crea el archivo correspondiente (si ya existe, se borra automáticamente y se queda en blanco) y se rellena con unos datos iniciales de prueba:

- Horario 1: 1-2 y 3-4
- Horario 2: 5-6 y 7-8

Tras esto, se guarda el archivo y se pasa a con la siguiente comprobación.

5.1.2 Secretos de conexión WiFi

Al igual que antes, se comprueba la existencia del archivo y la integridad de sus datos.

En caso de que no existan o los datos no sean correctos, se pasa a un modo donde, tras el arranque, se ejecuta un programa donde se le pide al usuario que introduzca dichas credenciales.

Esto se realiza a través de una web hecha en HTML que el propio micro expone en su ip cuando entra en modo Access Point, es decir, que es el propio mCU el que emite una señal WiFi. El usuario debe de conectarse a dicha señal y entrar en la ip específica.

Tras introducir los datos y enviarlos, el mCU los recibe y almacena en el archivo .json correspondiente y reinicia todo el sistema, ya con los datos almacenados.

Una particularidad de las credenciales de WiFi es que, en el programa principal, el microcontrolador trata de conectarse a la red indicada. Si tras 5 intentos, no es capaz de realizar la conexión, se borran el archivo con dichas credenciales y se reinicia el mCU para que vuelva a comprobar su existencia y vuelva a solicitarlas al usuario.

Figura 5.1: Seguridad WiFi

5.1.3 Token de seguridad 2FA

Al igual que anteriormente, se comprueba la existencia del archivo que contiene esta información y la integridad de la misma.

En caso de que esto no sea correcto, se genera una clave aleatoria como ya se ha explicado. Tras esto, el mCU sigue sus comprobaciones.

5.1.4 Credenciales de Adafruit IO

Finalmente, se comprueba la existencia de la información necesaria para el acceso a la plataforma de AIO.

En caso de no existir dicha información, el programa entra en modo de petición de datos al usuario, explicado en el apartado de las credenciales WiFi, solo que se solicita explícitamente la información relacionada con AIO.

Al igual que con los datos de WiFi, tras reunir la información necesaria, esta se almacena en su archivo y se reinicia el microcontrolador, para que vuelva a realizar todas las comprobaciones.

Es necesario recalcar que, en caso de que las credenciales sean erróneas o el mCU no pueda realizar la conexión con el broker de AIO, el archivo correspondiente a esta información será eliminado y el microcontrolador reiniciado, para que, cuando realice las comprobaciones de nuevo, solicite nuevamente las credenciales al usuario.

5.2 Reset

También se ha añadido una opción para poder reiniciar el Simulador y dejarlo como si fuese la primera vez que se inicia.

Para realizar el reset, hay que pulsar un botón de los que incluye el Pico Display (Y) durante uno 15-17 segundos. Tras esto, se muestra por pantalla que se va a reiniciar el simulador y, tras unos segundos, se borran todos los datos guardados y se reinicia el microcontrolador.

Al volver a iniciarse, volverá a pedir las credenciales nuevamente y creará una nueva key para el TOTP. En caso de perder la Key secreta del TOTP, este sería la solución a seguir.

6 IMPLEMENTACIÓN FÍSICA

La implementación del simulador de presencia se puede dividir en 2 circuitos separados: uno correspondiente a la tensión alterna de 220V, utilizada tanto para alimentar el circuito como para encender los dispositivos conectados, y otro de baja tensión que se encarga tanto de manejar los actuadores como de tomar las medidas necesarias de los sensores para los sistemas de seguridad.

6.1 Esquema general

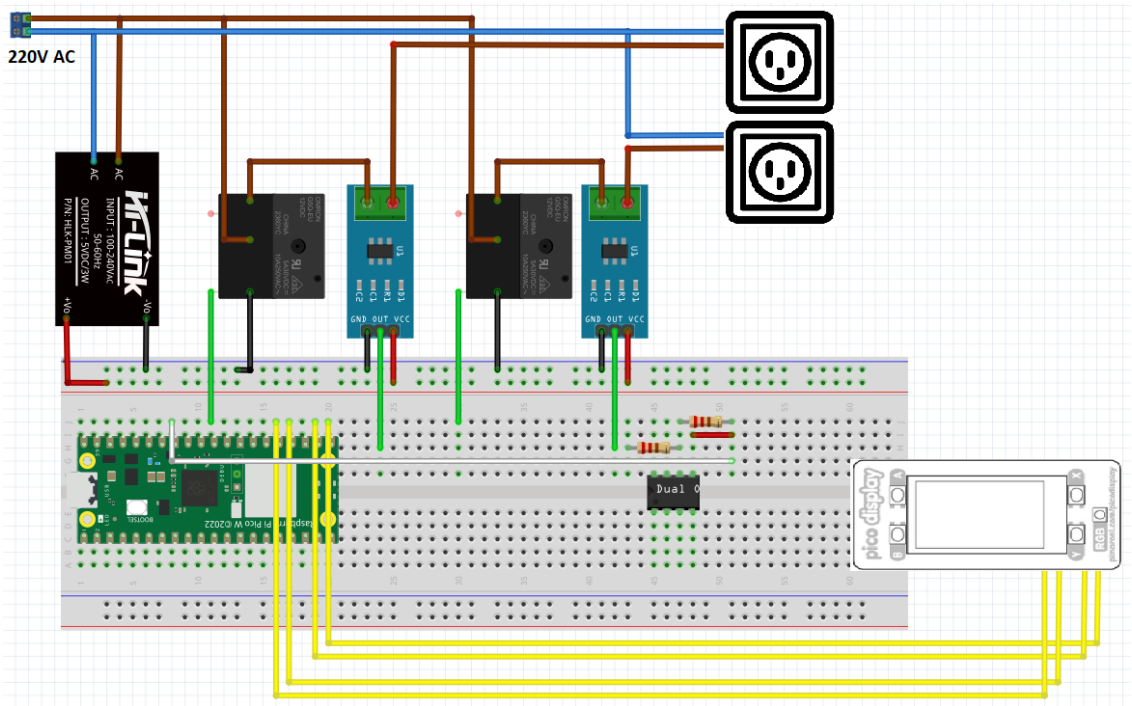

Figura 6.1: Esquema General Aproximado del Proyecto

En la imagen se puede observar de manera general las conexiones del sistema. La parte de 220V se encarga de alimentar los enchufes donde se conectan los dispositivos a controlar. Además, pasa por los relés, para que puedan ser controlados por el mCU.

Por otro lado, estos 220v se utilizan también para alimentar todo el circuito de baja tensión, gracias a un transformador AC/DC. De este transformador salen 5V que alimenta tanto al microcontrolador como a los relés y sensores.

El propio mCU es capaz de generar los 3.3V necesarios para alimentar el Display, con el que se conecta gracias a 4 pines que establecen una comunicación SPI con la pantalla. Además, en la placa de pruebas donde se coloca la Raspberry, se encuentra montado la etapa amplificadora para poder mejorar las medidas de los sensores.

6.2 Parte Analógica

La estructura física que soporta el simulador de presencia hace uso de relés, un dispositivo electromecánico

que permite accionar a voluntad una corriente, sirviendo de interruptor accionado electrónicamente. La gracia de los relés es que tiene un circuito aislado de activación y de carga. Esto resulta útil para manejar tensiones que un dispositivo pequeño como un microcontrolador no puede usar, como es el caso de los 220V AC de la corriente doméstica[29].

Uno de los problemas de los relés es que, debido a su uso y desgaste, puede originar que los terminales que hacen contacto para dar paso o no a la corriente se queden sujetos, soldados debido al calor que se puede generar en estos casos. Para detectar estos casos, se ha incluido un sensor capaz de medir la corriente alterna que circula por el circuito de carga. Gracias a la medida de este sensor, se puede detectar cuando hay corriente en el circuito y cuando no y, sobre todo, cuando la hay si no debería de haberla, saltando un error.

Esta medida de seguridad es importante debido a que, muchas veces, dejar un aparato eléctrico funcionando puede llegar a provocar percances indeseados. Pero realizar la medición de la intensidad acarrea una serie de problemas.

6.2.1 Lectura de Intensidad AC con ACS712

Para este caso, se ha utilizado el sensor de intensidades alternas ACS712, concretamente, el modelo de 5A, pues es un rango suficiente para medir intensidades típicas de, por ejemplo, lámparas y luces. el problema surge a la hora de medir la salida del sensor. El modelo de 5A tiene una sensibilidad de 185mV/A, es decir, que la tensión en el pin de salida varía 0.185V cada amperio que mida el sensor[30].

Por tanto, la ecuación del sensor sería la siguiente:

$$V_o = 2{,}5 + 0{,}185 \times I$$

Si tenemos en cuenta una bombilla incandescente de unos 100W, uno de los casos de mayor potencia que se puede encontrar, aunque ya no se fabriquen, tiene una intensidad de:

$$I = \frac{P}{V} = \frac{100}{220} = 0{,}45A$$

Es decir que, en un caso extremo, el sensor solamente variaría su tensión de salida en $\frac{0{,}185}{2} = 0{,}093A$, algo muy complicado de medir por el ADC del microcontrolador, cuyo valor de referencia son 3,3V y tiene una resolución de 12 bits[31].

$$V_o = 2{,}5 + 0{,}185 \cdot 0{,}45 = 2{,}583V$$

Por tanto, el sensor mCU notaría la diferencia de 2,5V a 2,583V, es decir, de 0,0853V

$$\mathrm{D} = 0{,}0853 \cdot \frac{2^{12}}{3{,}3} \cong 105$$

Parece un valor relativamente factible para detectarlo en el programa, pero esto es teniendo en cuenta una bombilla de unos 100W, algo que hoy en día ya no se usa ni se fabrica. Siendo realista, y poniendo una potencia de una bombilla LED, de aproximadamente 8W, el resultado es el siguiente:

$$I = \frac{P}{V} = \frac{8}{220} = 0{,}036A$$

$$V_o = 2{,}5 + 0{,}185 \cdot 0{,}036 = 2{,}5066V$$

$$\mathrm{D} = 0{,}0066 \cdot \frac{2^{12}}{3{,}3} \cong 8$$

Si además tenemos en cuenta los ruidos que aparecen en las mediciones, detectar una diferencia de 8 unidades en el programa ya es algo bastante más complejo.

6.2.2 Amplificación de la señal

Por este motivo, y para poder centrar la tensión de salida del sensor a la mitad del rango de medida del ADC, se ha decidido montar una etapa amplificadora con un OPAMP para aumentar la sensibilidad del sensor y que sea más sencillo poder medir esta intensidad.

Se ha decidido montar esta estructura pues permite, además de mejorar la ganancia, modificar el punto de funcionamiento, llevándolo al valor que nos interesa.

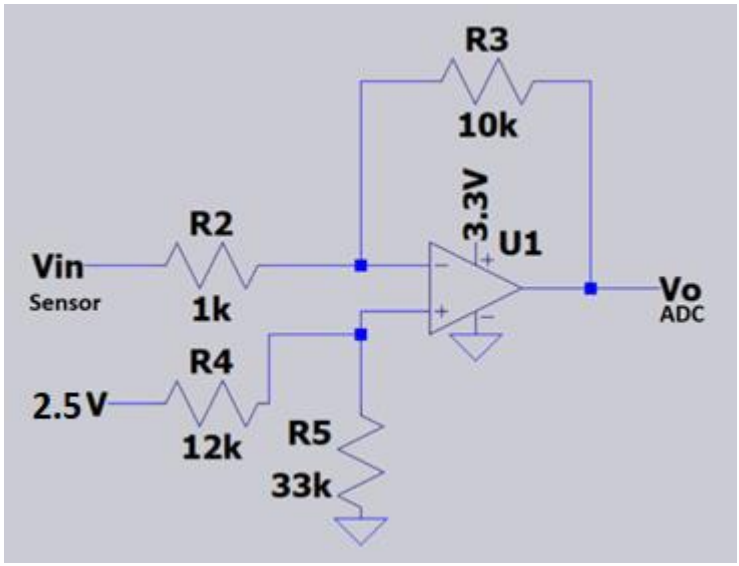

Figura 6.2: Circuito del Amplificador de Señal

Analizando el circuito presentado, podemos llegar a la siguiente ecuación:

$$V_o = \left(2{,}5 \cdot \frac{(R_2 + R_3) \cdot R_5}{(R_4 + R_5) \cdot R_2}\right) - \frac{R_3}{R_2} \cdot V_{in}$$

3,3V es la alimentación del amplificador operacional y para la tensión de referencia del circuito se ha usado 2.5V. Por otro lado, la relación existente entre R3 y R2 marcan la ganancia de la etapa amplificadora. Si se elige ganancia de 10, se puede establecer unos valores para R2 y R3, siendo uno R3 10 veces más grande que R2.

Teniendo ya estos valores, solamente queda buscar cuales son los valores de R4 y R5 que satisfagan una segunda condición, que la tensión media sea de $\frac{3{,}3}{2} = 1{,}65V$. Tras el análisis del circuito, se puede llegar a los siguientes valores:

Tabla 2: Valores de Resistencias de etapa amplificadora

Nombre de Resistencia	Valor
R2	1 K Ω
R3	10 K Ω
R4	3,3 K Ω
R5	100 K Ω

La tensión de 2,5V utilizada de referencia se obtiene gracias a un TL431C, un generador de referencia que devuelve una tensión constante utilizada para mejorar la precisión de circuitos[32].

Tras incluir la etapa amplificadora a la salida del sensor, la ecuación final es la siguiente:

$$V_o = 1{,}62 - 1{,}85 \times I$$

Como se puede observar, la tensión de referencia se ha modificado y la sensibilidad ha aumentado 10 veces. Por otro lado, hay que destacar que ahora la sensibilidad es negativa. Esto implica que, un aumento en la intensidad medida se refleja en una disminución de la tensión de salida. Esto es algo que no influye en el funcionamiento general del programa, pues se puede especificar en este para que funcione sin problemas.

Si repetimos las cuentas anteriores, este es el resultado:

$$V_o = 2{,}5 - 1{,}85 \cdot 0{,}036 = 2{,}566V$$

$$D = 0{,}066 \cdot \frac{2^{12}}{3{,}3} \cong 82$$

Como se puede observar, la variación de la medida del ADC ahora es notablemente superior a la anterior, lo que se puede medir con mejor seguridad que antes.

6.2.3 Detección de intensidad

No obstante, no es suficiente con solo tener un sensor que sea capaz de medir la corriente alterna que circula por el circuito, pues justamente por ser alterna, ocurre que esta oscila. Si tomamos un valor estático no nos sirve para nada, pues la intensidad está cambiando constantemente, y si hacemos la media de un numero de muestras tampoco, pues la oscilación de la corriente tiene de media 0, y no notaríamos la variación.

Por este motivo, para detectar la intensidad es necesario medir la amplitud de la onda que lee nuestro sensor tras pasar por el amplificador. Al ser una onda senoidal, es equivalente medir la amplitud que la amplitud de pico a pico.

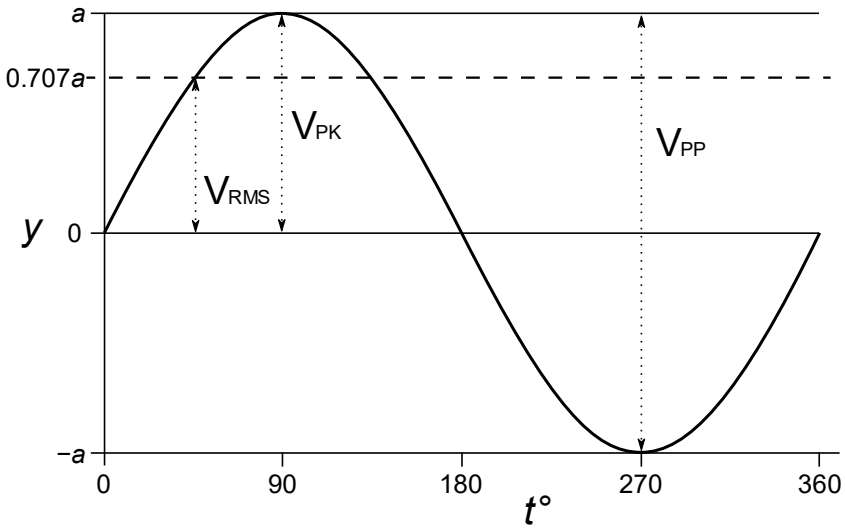

Figura 6.3: Onda Senoidal

Para la detección en nuestro simulador, realizaremos 3 pruebas de 0.5S con muestras cada 1ms. De estas muestras, solamente es necesario calcular la amplitud pico a pico restando el valor máximo obtenido al valor mínimo. Técnicamente, como la frecuencia de la red es de 50Hz, un ciclo completo s realiza cada 20ms, y con tomar muestras en este intervalo, sería suficiente. Pero se toman muestras de 25 oscilaciones para asegurarnos de la correcta medición. Si la mayoría de estas 3 pruebas tiene una amplitud de pico a pico superior a un umbral preestablecido, se considera como detección positiva.

En caso de detectar que el simulador no funciona correctamente, se escribirá un código en el feed correspondiente al dispositivo que no funciona. Adafruit IO mandará un correo electrónico en el momento de que esto ocurra, para alertar al usuario de que algo no funciona correctamente. Cuando el error pasa, el feed vuelve a su estado normal.

6.3 Entradas y Salidas

Una vez analizada la parte analógica, vamos a explicar el funcionamiento de las entradas y salidas del microcontrolador que se utilizan. Para ello, primero hay que conocer los elementos con los que se va a interactuar: Los relés y el display.

Los relés funcionan de manera sencilla. Se les alimenta con 5V y se activan cuando se les da una señal alta (high) en su pin de control. A este pin se le conecta una de las salidas digitales del mCU para poder manejarlo.

La mayoría de los relés traen 3 entradas de su circuito de potencia: la común, que siempre está conectada, la normalmente cerrada, que se conecta con la común cuando no se está actuando sobre el relé, y la normalmente abierta, que se conecta cuando se activa este. Como nosotros queremos encender el aparato conectado cuando enviemos la señal, los enchufes donde conectamos los aparatos están cableados en la entrada normalmente abierta del relé. Como controlamos 2 dispositivos, serán necesarios 2 relés independientes.

Por otro lado, el display se comunica con el SPI. En la comunicación SPI, son necesarios 4 conexiones entre el microcontrolador y el periférico. Estas líneas son: SCLK (Señal de reloj), SS (Chip Select), MISO (Master In, Slave Out) y MOSI (Master Out, Slave In). La Raspberry Pi pico W cuenta con 2 busses SPI para utilizar. En este caso. A Continuación, se hará un recuento de todos los pines utilizados:

Tabla 3: Nº de Pines de la RPPico W usados

Uso	Número del Pin de RP Pico W
Salida Relé 1 (D)	6
Salida Relé 2 (D)	7
Botón RO (D)	16
Botón Reset (D)	20
MISO	21
SS	22
SCLK	24
MOSI	25
Entrada Sensor 2 (A)	31
Entrada sensor 1 (A)	32

7 INFORMACIÓN AL USUARIO

Una parte importante para el uso de cualquier dispositivo es que este mismo devuelva algún tipo de retroalimentación o información para que el usuario sea capaz de distinguir que el simulador está haciendo algo, es decir, que no se ha quedado colgado, por ejemplo. Además, también es importante que el usuario pueda consultar la configuración del dispositivo, para poder actuar en consecuencia, es decir, poder modificar cuales son los horarios de encendido y apagado de cada aparato.

Se ha llevado este apartado por dos ramas: una es la comunicación online, que permite conocer el estado del simulador cuando se está manejando de esta forma; y a través de una pantalla IPS LCD de 1,14 pulgadas, el Pico Display Pack[24], que permite mostrar información directamente al usuario de manera presencial. Esta pantalla también es una parte importante de la seguridad de simulador, pues sirve para mostrar el QR que se debe escanear para modificar las configuraciones del simulador.

7.1 El panel de control

Antes de entrar en las maneras de comunicarnos con el usuario, es importante mostrar cuál es la interfaz que verá dicho usuario cuando quiera modificar la configuración del simulador. Gracias a AIO, que permite mostrar y manejar información online de manera sencilla, esta tarea es sencilla.

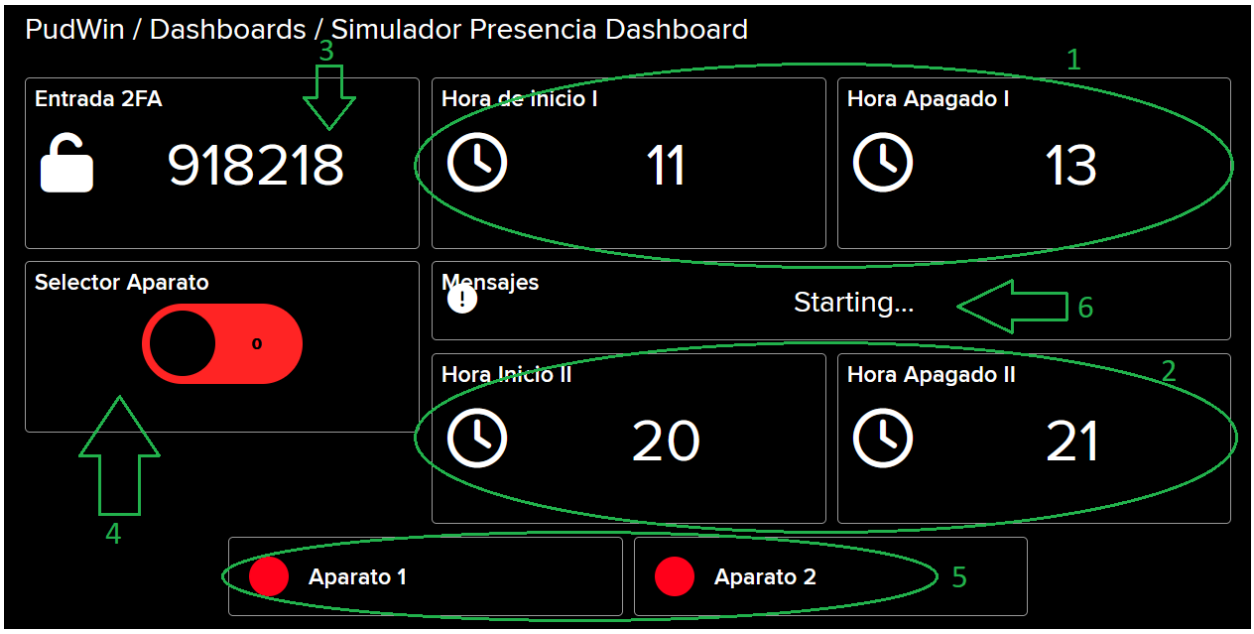

Figura 7.1: Dashboard del Simulador de Presencia

Se pueden distinguir 6 apartados, con sus utilidades:

1. Configuración de horas del periodo 1
2. Configuración de horas del periodo 2
3. Entrada del doble factor de autenticación (TOTP)
4. Selector de dispositivo a configurar
5. Información del estado de los aparatos conectados.
6. Información relativa a la configuración que emite el mCU

Los elementos del 1 al 4 son entradas del usuario, es decir, que es donde este debe de modificar las configuraciones. El elemento 4, el selector, permite multiplexar la conexión y poder modificar ambos dispositivos con las mismas entradas del dashboard.

Para poder modificar una configuración, el usuario debe introducir la clave TOTP en el elemento 3. Mientras esta clave sea válida, unos 30 segundos, puede realizar todas las modificaciones que desee en los horarios de los dispositivos.

El elemento 5 indica el estado de los dispositivos conectados. Si este está encendido, la luz pasa a estado verde. Finalmente, el elemento 6 corresponde con los mensajes de información que el mCU muestra al usuario

7.2 Comunicación Online

Como ya se ha comentado en el apartado 3, Conexión a Internet, el simulador se conecta al broker MQTT de Adafruit IO para poder comunicarse con este y poder escribir en los topics disponible o suscribirse a los topics interesados.

A través de este canal se puede enviar cualquier información posible, desde números, enteros o decimales, pasando por booleanos y, por supuesto, cadenas de texto. Se ha aprovechado de esta comunicación para crear un canal de comunicación entre los usuarios y el microcontrolador.

En primer lugar, se ha habilitado un par de topics que almacenan información booleana de si alguno de los dispositivos está encendido en ese momento.

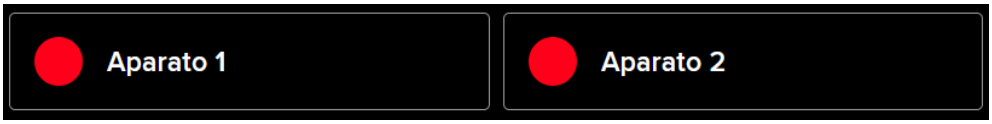

Figura 7.2: Indicadores de Estado de aparatos manejados

Esos topics se representan en el dashboard en el elemento 5.

El estado de los dispositivos es, realmente, una variable interna del programa del simulador. Simplemente, se escribe en el topic correspondiente si se realiza un cambio de encendido a apagado o viceversa.

Figura 7.3: Dashboard cuándo el aparato 1 está encendido

Como la comprobación de los horarios se realiza una vez cada minuto, la actualización del estado tiene esta misma periodicidad, en el caso extremo de que se encienda y se apague seguido. Esto es importante pues hay que tener en cuenta la limitación de la tasa de datos mencionada en el apartado 3.1.1, Limitaciones de Adafruit IO.

Por otro lado, el elemento 6 del dashboard corresponde a la ventana de mensajes del mCU.

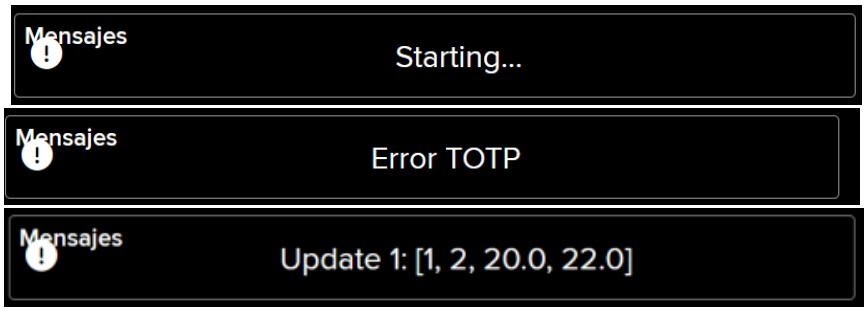

Figura 7.4: Ejemplo de Mensajes enviados por el microcontrolador

Cuando se realiza una acción, esta se ve reflejada en un mensaje que el mCU envía una vez cada minuto a esta ventana. En esta se muestra, por ejemplo, el valor de los horarios actuales, en función del dispositivo seleccionado, si el código TOTP es erróneo o simplemente si se está iniciando.

7.3 Uso del Pico Display

La pantallita del Pico Display es muy útil para mostrar información al usuario de manera local. Además, como esta no tiene la limitación de datos, se puede actualizar casi de manera inmediata.

En primer lugar, al iniciar el microcontrolador, se muestra una primera imagen donde se puede leer el código QR para poder guardarlo en la aplicación preferida por el usuario. El simulador se queda en esta pantalla en espera a que el usuario introduzca una vez, de manera correcta, el código TOTP a través del dashboard.

Figura 7.5: Estado de espera a TOTP

Tras introducir correctamente la verificación y una pequeña pantalla de inicio, se muestran los horarios de ambos dispositivos en la pantalla. Se pueden ver las horas de inicio y fin de ambos periodos de ambos dispositivos.

Figura 7.6: Ambos dispositivos apagados

En este caso, el estado de los dispositivos se indica con el color del texto del dispositivo en concreto. En caso de encontrarse encendido, las letras se dibujan de color azul.

Figura 7.7: Dispositivo 1 Encendido

La gran ventaja del uso de este Display es que, además de mejorar la seguridad, pues se necesita estar delante del simulador cuando se inicia para poder manejarlo de manera remota, se puede obtener la retroalimentación necesaria para el manejo de manera más inmediata y cómoda para el usuario.

8 MÁQUINA DE ESTADOS

En este capítulo, se explora de manera general el funcionamiento del simulador de presencia en casa. Se muestra las posibles situaciones que se pueden encontrar y se indica como se resuelve.

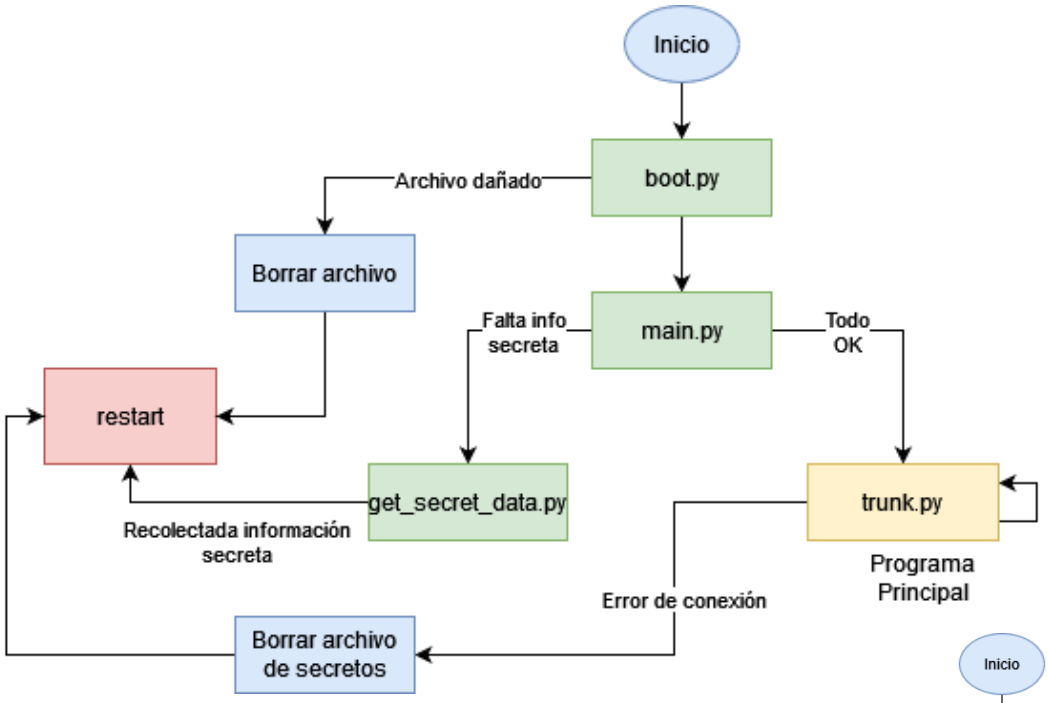

Figura 8.1: Máquina de Estados General

A continuación, se muestra el funcionamiento del modo get_secret_data, donde se recibe información del usuario a través de modo de acceso del microcontrolador.

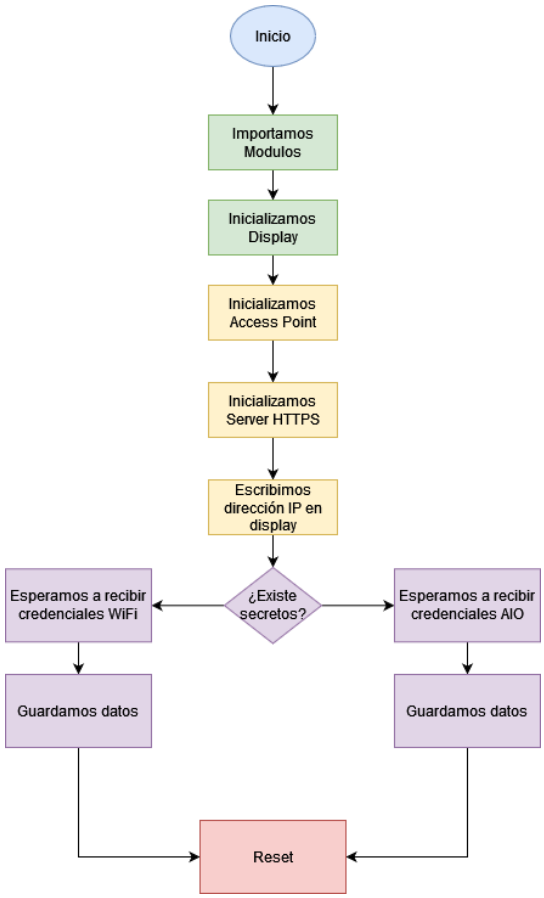

Figura 8.2: Máquina de Estados del modo obtención de datos

En boot.py se comprueba la existencia y la integridad de todos los archivos que el simulador necesita para su correcto funcionamiento. Estos archivos almacenan información que persiste cuando se reinicia el mCU.

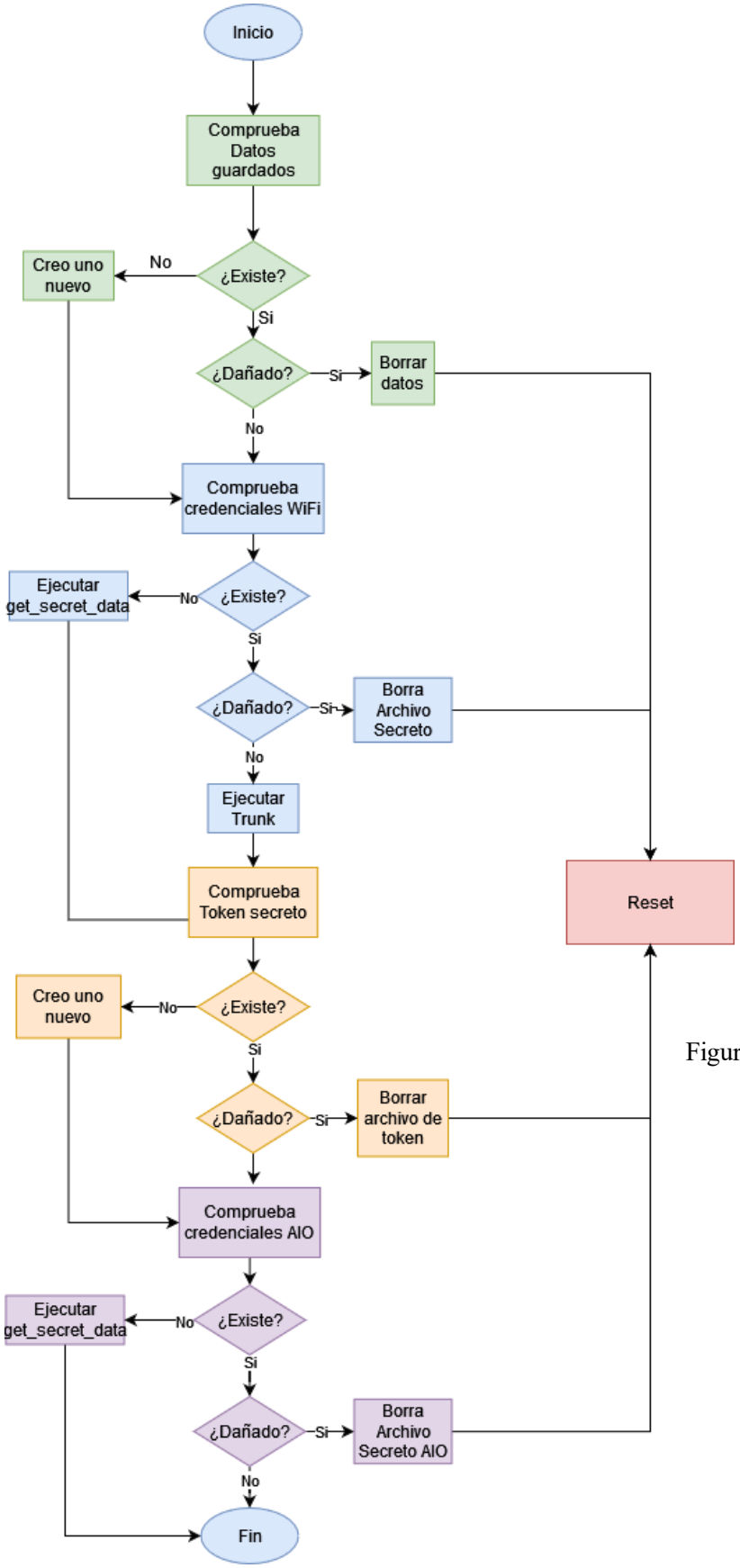

Figura 8.3: Máquina de Estados de lanzamiento (boot)

Finalmente, se muestra el funcionamiento del programa principal.

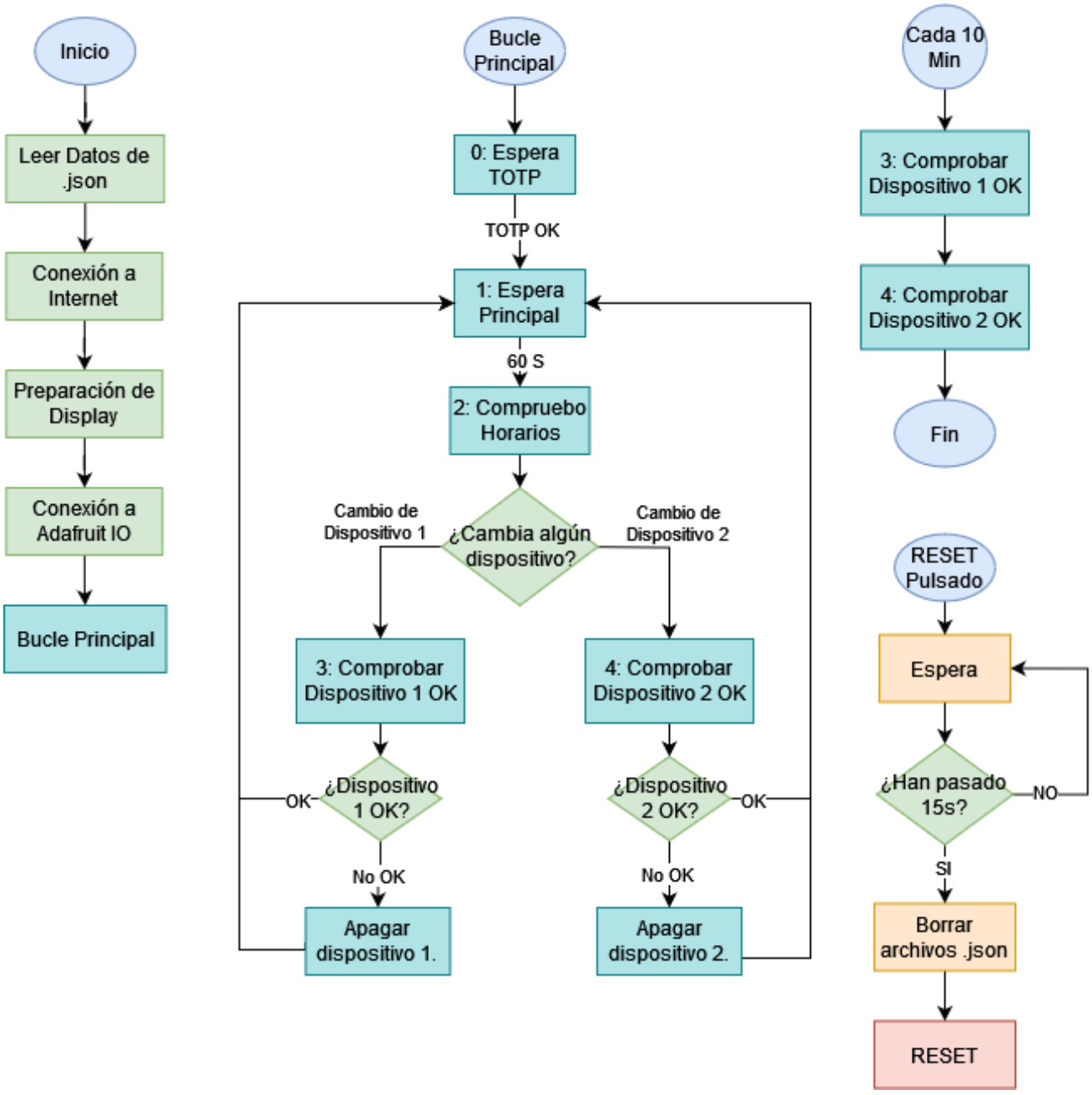

Figura 8.4: Máquina de Estados de programa Principal

9 PRESUPUESTO

En este punto se va a detallar de manera aproximada el presupuesto, contando con todos los elementos utilizados en el proyecto.

Tabla 4: Resumen de elementos usados y precios

Elemento	Precio individual [€]	Cantidad	Precio Total [€]
RP Pico W	8,4	1	8,4
PicoDisplay pack	17,54	1	17,54
Enchufes	4	2	8
Relés	1,58	2	3,16
Sensores ACS 712	4,1	2	8,2
Resistencias	0,065	8	0,52
Opamp	0,26	1	0,26
Tecla	1	1	1
Menaje Vario	5	1	5
TL431c	1	1	1
Transformador AC/DC	5,345	1	5,345
Total			58,425

10 CONCLUSIONES

El objetivo de este trabajo es el de Realizar un Simulador de presencia en Casa, que sea capaz de manejar un par de dispositivos electicos o electrónicos para aumentar la seguridad de las casas, sobre todo en tiempos de vacaciones y cuando las casas se encuentran vacías.

Además, también se pretendía añadir una capa más de seguridad, tanto a nivel de Software y comunicaciones, con el TOTP, como a nivel de Hardware, con la detección de corriente. Estas características ayudan a aumentar la confianza de los usuarios en dejar un dispositivo conectado a la electricidad solo en casa, sin supervisión.

Haciendo un repaso a las Especificaciones señaladas al principio, se puede ver como el simulador final cumple con todas estas, y que, por lo tanto, el desarrollo ha sido un éxito.

REFERENCIAS

[1] «Ministerio del Interior | La Policía Nacional facilita consejos para prevenir los robos en domicilio de cara a las vacaciones veraniegas», 30 de junio de 2020. https://www.interior.gob.es/opencms/es/detalle/articulo/La-Policia-Nacional-facilita-consejos-para-prevenir-los-robos-en-domicilio-de-cara-a-las-vacaciones-veraniegas/ (accedido 15 de junio de 2023).

[2] «Microcontrolador - Wikipedia, la enciclopedia libre». https://es.wikipedia.org/wiki/Microcontrolador (accedido 19 de junio de 2023).

[3] «Buy a Raspberry Pi Pico – Raspberry Pi». https://www.raspberrypi.com/products/raspberry-pi-pico/ (accedido 5 de junio de 2023).

[4] «Raspberry Pi Documentation - RP2040». https://www.raspberrypi.com/documentation/microcontrollers/rp2040.html (accedido 5 de junio de 2023).

[5] «Microcontroladores Raspberry Pi PICO, H, W». https://www.kubii.com/es/las-tarjetas-raspberry-pi/3205-1641-raspberry-pi-pico-w-h-wh-3272496311589.html#/version_pico-pico_w (accedido 19 de junio de 2023).

[6] «Amazon.es : esp32». https://www.amazon.es/esp32/s?k=esp32 (accedido 19 de junio de 2023).

[7] «ARDUINO UNO WiFi REV2 — Arduino Official Store». https://store.arduino.cc/products/arduino-uno-wifi-rev2 (accedido 19 de junio de 2023).

[8] D. George, «MicroPython - Python for microcontrollers». https://micropython.org/ (accedido 19 de junio de 2023).

[9] K. Rembor, lady ada, J. Doscher, Á. Figueroa, y M. Schroeder, «¡Bienvenido a CircuitPython! | Adafruit Learning System», 20 de marzo de 2020. https://learn.adafruit.com/bienvenido-a-circuitpython-2?view=all (accedido 5 de junio de 2023).

[10] B. Rubell, T. Cooper, J. Cooper, A. Bachman, y K. Rembor, «Welcome to Adafruit IO | Adafruit Learning System», 13 de junio de 2018. https://learn.adafruit.com/welcome-to-adafruit-io?view=all (accedido 5 de junio de 2023).

[11] «Adafruit IO Frequently Asked Questions». https://io.adafruit.com/faq (accedido 5 de junio de 2023).

[12] L. Llamas, «¿Qué es MQTT? Su importancia como protocolo IoT», 18 de abril de 2019. https://www.luisllamas.es/que-es-mqtt-su-importancia-como-protocolo-iot/ (accedido 5 de junio de 2023).

[13] Colaboradores de Wikipedia, «Autenticación con contraseña de un solo uso», *Autenticación con contraseña de un solo uso*, 20 de abril de 2022. https://es.wikipedia.org/w/index.php?title=Autenticaci%C3%B3n_con_contrase%C3%B1a_de_un_solo_uso&oldid=143013053 (accedido 5 de junio de 2023).

[14] «RFC 4226 - HOTP: An HMAC-Based One-Time Password Algorithm». https://datatracker.ietf.org/doc/html/rfc4226 (accedido 5 de junio de 2023).

[15] D. Hellmann, «PyMOTW-3 HMAC», 30 de julio de 2017. https://pymotw.com/3/hmac/index.html (accedido 5 de junio de 2023).

[16] D. M'Raihi, S. Machani, M. Pei, y J. Rydell, «RFC 6238 - TOTP: Time-Based One-Time Password Algorithm», marzo de 2011. https://datatracker.ietf.org/doc/html/rfc6238 (accedido 5 de junio de 2023).

[17] *The Open Group Base Specifications Issue 7, section 4.16 Seconds Since the Epoch*. The Open Group,

2017. Accedido: 5 de junio de 2023. [En línea]. Disponible en: http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap04.html#tag_04_16

[18] AngainorDev, «Picoth: An OTP enabled macro keyboard based upon a Rapberry Pico and RGB keypad base from Pimoroni.», enero de 2021. https://github.com/AngainorDev/Picoth (accedido 5 de junio de 2023).

[19] «Libraries». https://circuitpython.org/libraries (accedido 5 de junio de 2023).

[20] «Adafruit IO - Time Power-Up». https://io.adafruit.com/PudWin/services/time (accedido 5 de junio de 2023).

[21] M. P. Villatoro, «¿Desde cuándo se cambia la hora en España? De los caprichos de Franco y el Reich a la adaptación a Europa», *ABC*, 31 de octubre de 2022. Accedido: 14 de junio de 2023. [En línea]. Disponible en: https://www.abc.es/historia/cambia-hora-espana-caprichos-franco-reich-adaptacion-20221027182531-nt.html

[22] «World Time API». http://worldtimeapi.org/ (accedido 14 de junio de 2023).

[23] Colaboradores de CircuitPthon y MicroPython, «os – functions that an OS normally provides — Adafruit CircuitPython 8.2.0-beta.0 documentation». https://docs.circuitpython.org/en/latest/shared-bindings/os/index.html (accedido 5 de junio de 2023).

[24] «Pico Display Pack». https://shop.pimoroni.com/products/pico-display-pack?variant=32368664215635 (accedido 13 de junio de 2023).

[25] T. Habets, «Key Uri Format · google/google-authenticator Wiki», 26 de noviembre de 2018. https://github.com/google/google-authenticator/wiki/Key-Uri-Format (accedido 14 de junio de 2023).

[26] «Introduction — Adafruit miniQR Library 1.0 documentation». https://docs.circuitpython.org/projects/miniqr/en/latest/ (accedido 14 de junio de 2023).

[27] «JSON». https://www.json.org/json-es.html (accedido 7 de junio de 2023).

[28] MicroPython & CircuitPython contributors, «CircuitPython — Adafruit CircuitPython 8.2.0-beta.0 documentation», 2023. https://docs.circuitpython.org/en/latest/README.html#behavior (accedido 7 de junio de 2023).

[29] Colaboradores de Wikipedia, «Relé - Wikipedia, la enciclopedia libre». https://es.wikipedia.org/wiki/Rel%C3%A9 (accedido 7 de junio de 2023).

[30] «Fully Integrated, Hall Effect-Based Linear Current Sensor with 2.1 kVRMS Voltage Isolation and a Low-Resistance Current Conductor ACS712». [En línea]. Disponible en: www.allegromicro.com

[31] «Raspberry Pi Pico W Datasheet | Enhanced Reader», *Raspberry Pi L*, 2 de marzo de 2023. https://datasheets.raspberrypi.com/picow/pico-w-datasheet.pdf (accedido 7 de junio de 2023).

[32] «TL431, TL432 Precision Programmable Reference», 2022, Accedido: 29 de junio de 2023. [En línea]. Disponible en: www.ti.com

[33] «ANTELA Bombilla Inteligente LED E27 Wifi 9W Compatible Con Google Home/Alexa, Bombilla RGB(2700K-6500K) Luces Colores Regulable, Control Remoto,Control De Voz, Ahorro Energético, Paquete 2 : Amazon.es: Iluminación». https://www.amazon.es/dp/B08R8PP7Q9?ref_=cm_sw_r_cp_ud_dp_8VA53878N3ZEXBS52TKD&th (accedido 30 de junio de 2023).

[34] «MOES ZigBee Interruptor de luz inteligente No requiere cable neutro, No necesita condensador, Control app Smart Life Tuya, Funciona Alexa Google Home, requiere Tuya ZigBee Hub, 2 Gang, Blanco : Amazon.es: Bricolaje y herramientas». https://www.amazon.es/dp/B096W1J72D?ref_=cm_sw_r_cp_ud_dp_VMDF9NSVEPJFB5JYSBAK (accedido 30 de junio de 2023).

[35] «What is a Time-based One-time Password (TOTP)? | Twilio». https://www.twilio.com/docs/glossary/totp (accedido 30 de junio de 2023).

Anexo A: Código

En este anexo se adjuntará el código desarrollado.

boot.py

Programa ejecutado al inicio.

```
import digitalio
import board
import storage
import os

button = digitalio.DigitalInOut(board.GP12)
button.direction = digitalio.Direction.INPUT
button.pull = digitalio.Pull.UP

storage.remount("/",not button.value) # si se pulsa el botón A del pico
Display, el dispositivo no read-only
file = "datos.json"
secret_file_name = "secretos.json"
secret_key_filename = "secret_key.json"
adafruit_secret = "Adafruit_secrets.json"




try:
    from json import load, dump
    os.listdir().index(file)
    print("El archivo",file,"ya existe")
    with open(file,"r") as data:
        json_data = load(data)

    try:    # if the data in datos.json aren't correct -> Delete
        a = json_data["timetable1"]
        b = json_data["timetable2"]
    except:
        import os,microcontroller
        os.remove(file)
        microcontroller.reset()

except ValueError as err:   # If File not Found
```

```
    print("El archivo no existe")

    try:
        json_data = {
            "timetable1": [1,2,3,4],
            "timetable2": [5,6,7,8]
        }
        with open(file,"w") as data_file:  # Try to create a new file with
default data
            dump(json_data,data_file)

        print("Creado el archivo:", file)
    except OSError as error:
        if error.args[0] == 30: # Read only File System
            print("Read only File system")
        else:
            print("Error creando el archivo")
except:
    print("Error inesperado")




try:    # look for secretos.json
    os.listdir("dependencies").index(secret_file_name)
    print("El archivo",secret_file_name, "ya existe")
    import json
    program = {"actual":"trunk"}
    with open("run.json","w") as run_file:
        json.dump(program,run_file)

except ValueError as err:   # If secretos.json doesn't exist
    print("El archivo",secret_file_name,"no existe")
    try:
        import json     # Try to create a new file
        program = {"actual":"get_secret_data"}
        with open("run.json","w") as run_file:
            json.dump(program,run_file)

    except OSError as error:
        if error.args[0] == 30: # Read only File System
            print("Read only File system")
        else:
            print("Error creando el archivo")
except OSError as err:
    if err.args[0] == 30:
        print("Read only File System")
    else:
        print("Error creando el archivo secreto")
```

```
except:
    print("Error inesperado")



try:
    from json import load
    os.listdir("dependencies").index(secret_key_filename)
    print("El archivo",secret_key_filename,"ya existe")
    with open("dependencies/"+secret_key_filename,"r") as data:
        json_data = load(data)

    try:    # if the data in secret_key.json aren't correct -> Delete
        a = json_data["TOTP_key"]
    except:
        import os,microcontroller
        os.remove("dependencies/"+secret_key_filename)
        microcontroller.reset()

except ValueError as err:   # If the file doesn't exist
    print("El archivo",secret_key_filename,"no existe")
    from dependencies.AIO_Callbacks import Random_32_Key_generator as RKG
    from json import dump

    try:
        secret_key = {"TOTP_key" : RKG(20)}
        with open("dependencies/"+secret_key_filename,"w") as
secret_key_file:
            dump(secret_key,secret_key_file)

        print("Creado:",secret_key_filename)
    except OSError:
        if error.args[0] == 30: # Read only File System
            print("Read only File system")
        else:
            print("Error creando el archivo:",secret_key_filename)




try: # Look for Adafruit_secrets.json
    from json import load
    os.listdir("dependencies").index(adafruit_secret)
    print("El archivo",adafruit_secret,"ya existe")
    with open("dependencies/"+adafruit_secret,"r") as data:
        json_data = load(data)

    try:    # if the data in Adafruit_secrets.json aren't correct -> Delete
        a = json_data["broker"]
        b = json_data["io_key"]
```

```
        c = json_data["port"]
        d = json_data["io_user"]

    except:
        import os,microcontroller,json
        os.remove("dependencies/"+adafruit_secret)
        microcontroller.reset()

except ValueError as err:   # If the file doesn't exist
    print("El archivo",adafruit_secret,"no existe")
    from json import dump

    try:
        program = {"actual":"get_secret_data"}
        with open("run.json","w") as run_file:
            json.dump(program,run_file)

    except OSError:
        if error.args[0] == 30: # Read only File System
            print("Read only File system")
        else:
            print("Error creando el archivo:",adafruit_secret)
except OSError as err:
    if err.args[0] == 30:
        print("Read only File System")
    else:
        print("Error creando el archivo secreto")
except:
    print("Error inesperado")
```

main.py

Elección de modo a ejecutar

```
import json

run_program = open("run.json","r")
to_run = json.load(run_program)
run_program.close()

if to_run["actual"] == "get_secret_data":
    import get_secret_data
elif to_run["actual"] == "trunk":
    import trunk
else:
    import time
    while True:
        print("Error")
        time.sleep(10)
```

get_secret_data.py

Modo de Recolección de Datos por Wifi

```
# Import Modules
import wifi
import socketpool
import time
import displayio, vectorio
from terminalio import FONT
    # Import adafruit bundle modules
from adafruit_httpserver.mime_type import MIMEType
from adafruit_httpserver.request import HTTPRequest
from adafruit_httpserver.response import HTTPResponse
from adafruit_httpserver.server import HTTPServer
from adafruit_httpserver.methods import HTTPMethod
from adafruit_display_text import bitmap_label
from dependencies.AIO_Callbacks import initialize_display
import os

time.sleep(3)   # Sleep time to wait the complete boot of the wifi chip

## Initialize the display
display = initialize_display(rotation=270)
    # Create a Background
back_color = displayio.Palette(1)
back_color[0] = 0x2EC97C
Background =
vectorio.Rectangle(pixel_shader=back_color,x=0,y=0,width=display.width,height=di
splay.height)

    # Create a Splash Group
splash = displayio.Group()
splash.append(Background)

    ## Start the Acces Point
pico_ssid = "Pico_W"
password = "12345678"
wifi.radio.stop_ap()
wifi.radio.start_ap(pico_ssid,password)

pool = socketpool.SocketPool(wifi.radio)
server = HTTPServer(pool)

# The HTML
def webpage():
    html = f"""
    <!DOCTYPE html>
    <html>
    <head>
```

```
        <title>Simulador de Presencia</title>
    </head>
    <body>
    <div id="cabecera">
        <h1>Simulador de Presencia</h1>
        <h2>Introduzca credenciales WIFI</h2>
    </div>

    <div id="texto">
        <p>Página creada con la función de obtener información de necesaria para
la conexión a
        la red y el funcionamiento correcto del simulador de presencia
        </p>
        <p>Este Proyecto se corresponde con el Trabajo de Fin de Grado de Pedro
Tomás Martínez Flores,
        alumno de 4º curso de GIERM en la universidad de Sevilla
        </p>
        <p>A Continuación, introduzc los datos requeridos la la conexión:
        SSID y Contraseña del WIFI</p>
    </div>

    <form method="POST">
        <label for="ssid">SSID</label><br>
        <input type="tex" id="ssid" name="SSID"><br>
        <label for="pass">Password</label><br>
        <input type="tex" id="pass" name="Password"><br>
        <input type="submit">
    </form>

    </body>
    </html>
    """
    return html

def webpage_ada():
    html = f"""
    <!DOCTYPE html>
    <html>
    <head>
        <title>Simulador de Presencia</title>
    </head>
    <body>
    <div id="cabecera">
        <h1>Simulador de Presencia</h1>
        <h2>Introduzca credenciales de Adafruit IO</h2>
    </div>

    <div id="texto">
```

```
        <p>Página creada con la función de obtener información de necesaria para
la conexión a
        la red y el funcionamiento correcto del simulador de presencia
        </p>
        <p>Este Proyecto se corresponde con el Trabajo de Fin de Grado de Pedro
Tomás Martínez Flores,
        alumno de 4º curso de GIERM en la universidad de Sevilla
        </p>
        <p>A Continuación, introduzc los datos requeridos la la conexión:
        Usuario y key de Adafruit IO</p>
    </div>

    <form method="POST">
        <label for="io_user">Adafruit User</label><br>
        <input type="text" id="io_user" name="Usuario AIO"><br>
        <label for="io_key">Adafruit Key</label><br>
        <input type="text" id="io_key" name="Key AIO"><br>
        <input type="submit">
    </form>

    </body>
    </html>
    """
    return html

try:
    exist_secrets = os.listdir("dependencies").index("secretos.json")   # If
secretos.json exists -> adafruit_secrets.json doesn't

    @server.route("/")  #Standard Action
    def base(request: HTTPRequest):
        """
        Serve the default index.html file.
        """
        with HTTPResponse(request, content_type=MIMEType.TYPE_HTML) as response:
            response.send(f"{webpage_ada()}")

    @server.route("/", method=HTTPMethod.POST)  # Action when recive a POST
request
    def buttonpress(request: HTTPRequest):
        #  get the raw text
        raw_text = request.raw_request.decode("utf8")
        # Tread the data from the raw text
        io_user = raw_text.split("\n")[-1].split("&")[0].split("=")[1]
        io_key = raw_text.split("\n")[-1].split("&")[1].split("=")[1]
        # Hide the display
        splash.hidden = True
        display.show(splash)
        # Make the dictonary to put in the secretos file
```

```
        secrets = {
            "io_user" : io_user,
            "io_key" : io_key,
            "broker": "io.adafruit.com",
            "port": 1883
        }

        # Create and write the secretos File
        import json
        with open("dependencies/Adafruit_secrets.json","w") as secret_file:
            json.dump(secrets,secret_file)

        # Reset the microcontroller
        import microcontroller
        microcontroller.reset()

except: # secretos.json daesn't exists
    @server.route("/")  #Standard Action
    def base(request: HTTPRequest):
        """
        Serve the default index.html file.
        """
        with HTTPResponse(request, content_type=MIMEType.TYPE_HTML) as response:
            response.send(f"{webpage()}")

    @server.route("/", method=HTTPMethod.POST)  # Action when recive a POST
request
    def buttonpress(request: HTTPRequest):
        #  get the raw text
        raw_text = request.raw_request.decode("utf8")
        # Tread the data from the raw text
        SSID= raw_text.split("\n")[-1].split("&")[0].split("=")[1]
        PASS= raw_text.split("\n")[-1].split("&")[1].split("=")[1]
        # Hide the display
        splash.hidden = True
        display.show(splash)
        # Make the dictonary to put in the secretos file
        secrets = {
            "ssid" : SSID,
            "password" : PASS
        }

        # Create and write the secretos File
        import json
        with open("dependencies/secretos.json","w") as secret_file:
            json.dump(secrets,secret_file)

        # Reset the microcontroller
        import microcontroller
```

```
        microcontroller.reset()

print_text = f"\nListening on http://{wifi.radio.ipv4_address_ap}:80"
print(print_text)

# Make the label to display
display_text = "SSID:"+pico_ssid+"\nWaiting
wifi/AIO\ndata:\nhttp://"+str(wifi.radio.ipv4_address_ap)

Text = bitmap_label.Label(FONT,text = display_text, color=0x000000)
Text.anchor_point = (0,0.5)
Text.anchored_position=(0,0)
text_group = displayio.Group(
    scale=2,
    x=3,
    y=display.height//2
)
text_group.append(Text)
splash.append(text_group) # Add the text to the group

# Start the server.
server.start(str(wifi.radio.ipv4_address_ap))   # Start the server

display.show(splash)

while True: # Infinite loop
    try:
        # Process any waiting requests
        server.poll()
    except OSError as error:
        print(error)
        continue
```

trunk.py

Programa principal

```
# Programa principal de Simulador de Presencia.
# TFG Pedro Tomás Martínez flores

## Libraries
import time
from ssl import create_default_context
from wifi import radio
from socketpool import SocketPool
import board
import digitalio
from analogio import AnalogIn
from json import load,dump
from terminalio import FONT
import displayio
from vectorio import Rectangle
import gc
import alarm
# Adafruit Bundle libraries
import adafruit_minimqtt.adafruit_minimqtt as MQTT
from adafruit_io.adafruit_io import IO_MQTT
from adafruit_requests import Session
from adafruit_display_text import bitmap_label
# Other dependencies
import dependencies.AIO_Callbacks as Callback   # Custom file with Callbaks,
classes and functions

try:
    print()
    file_name = "datos.json"
    with open(file_name,"a") as file:  # Try to open the file and check the
write system file option
        pass
    with open(file_name,"r") as file:
        datos = load(file)


    with open("dependencies/secretos.json","r") as secret_file:
        secrets = load(secret_file)

    with open("dependencies/Adafruit_secrets.json","r") as secret_file:
        A_secrets = load(secret_file)

    with open("dependencies/secret_key.json","r") as TOTP_key_file:
        TOTP_data = load(TOTP_key_file)
```

```
    ## Sleep time to wait the complete boot of the wifi chip
    wifi_setup_alarm = alarm.time.TimeAlarm(monotonic_time=time.monotonic()+3)
    alarm.light_sleep_until_alarms(wifi_setup_alarm)


    ## Global Variables
    timetable1 = datos["timetable1"]    # Initial device 1 Schedule
    timetable2 = datos["timetable2"]    # Initial device 2 Schedule
    TOTP_key = TOTP_data["TOTP_key"]

    actual_tt = [0,0]
    Error1=0
    Error2=0

    running_check_time = 10 # Time between checks in minutes
    RESET_TIME = 15  # Time in seconds the reset button must to be pressed to
iniciate the RESET

        # The main object
    schedule = Callback.Schedule(0,timetable1,timetable2,TOTP_key)    # Create
schedule object whith both timetables
    gc.collect()
        #Devices state machines
    device1=Callback.state_machine(0,2)
    device2=Callback.state_machine(0,2)
        # State of the program
    state = 0
        #0: Starting/1: Wait a  minute/2: Check the timetables/3-4: Check if
running OK

        # Data to get the timezone
    base_url = "http://worldtimeapi.org/api/timezone/{zone}/{area}"
    zone = "Europe"
    area = "Madrid"
        # Data to make the QR
    QR_usser = "Hola"
    QR_account = "Prueba_2"
    QR_data = TOTP_key
        # Defines
    FONTSCALE = 2
    ON_TEXT_COLOR = 0x0700c9
    OFF_TEXT_COLOR = 0x000000


    # Configurate board outputs
    output1 = digitalio.DigitalInOut(board.GP5)
    output1.direction = digitalio.Direction.OUTPUT
    output2 = digitalio.DigitalInOut(board.GP4)
    output2.direction = digitalio.Direction.OUTPUT
```

```
    led = digitalio.DigitalInOut(board.LED)
    led.direction = digitalio.Direction.OUTPUT
    led.value = False

    # Configure Reset Input
    RESET_INPUT = digitalio.DigitalInOut(board.GP15)
    RESET_INPUT.direction = digitalio.Direction.INPUT
    RESET_INPUT.pull = digitalio.Pull.UP

    # Configurate sensor inputs
    input1 = AnalogIn(board.A1)
    input2 = AnalogIn(board.A0)

    ## Initialize the pico Display
    display = Callback.initialize_display(rotation=270)

    ## Connect to WiFi
    print("Conecting to internet....")
    trys = 0
    for i in range(5):
        try:
            radio.connect(secrets["ssid"], secrets["password"])
            print("\nConectado a Red ",secrets["ssid"]," con ip:
",radio.ipv4_address)
            break
        except:
            trys += 1
            print("Tries:",trys,"-> Retrying the conexion...")
    if trys > 4:
        raise ConnectionError

    pool = SocketPool(radio)


    # API Request to obtain the utc_offset of the actual timezone
    requests = Session(pool, create_default_context())

    final_url=base_url.format(zone=zone,area=area)
    time_tz=requests.get(final_url) # get petitions
    time_offset = time_tz.json()['utc_offset']  # extract the data from the json
    offset_hour,offset_minute = Callback.get_hmoffset(time_offset)  # obtain the
offset in hours and minutes

    print ("Off_hour", offset_hour,"Off_min",offset_minute)
    gc.collect()

    # Make display images
        # Create the QR bitmap
    qr_bitmap = Callback.bitmap_QR(QR_usser,QR_account,QR_data)
```

```
    pallete = displayio.Palette(2)
    pallete[0] = 0xFFFFFF
    pallete[1] = 0x000000
    qr_tile = displayio.TileGrid(qr_bitmap,pixel_shader=pallete,x=0,y=0)

        # Add a backgruound Color
    Bkgrd = displayio.Bitmap(display.width,display.height,1)
    bkgrd_color = displayio.Palette(1)
    bkgrd_color[0]=0x2EC97C
    bkgrd_tile = displayio.TileGrid(Bkgrd,pixel_shader=bkgrd_color,x=0,y=0)
        # Add two rectangles to the backgraund
    Bck_rect_color = displayio.Palette(1)
    Bck_rect_color[0] = 0xbfc204
    left_rect = Rectangle(pixel_shader=Bck_rect_color,width=display.width//2-
9,height=display.height -4,x=2,y=2)
    right_rect = Rectangle(pixel_shader=Bck_rect_color,width=display.width//2-
9,height=display.height -4,x=display.width//2+2,y=2)


        # Make a text for the QR screen
    TEXT_COLOR = 0x000000
    QR_text = bitmap_label.Label(FONT, text = "Read the\nQR to add\nthe TOTP\nto
your\napp", color = TEXT_COLOR)
    QR_text.anchor_point = (0,0.5)
    QR_text.anchored_position = (0,0)
    QR_text_group = displayio.Group(
        scale=FONTSCALE,
        x=display.width//2 + 8,
        y=display.height//2

    )
    QR_text_group.append(QR_text)

        # Make the display context
    QR_group = displayio.Group(scale=3,x=3,y=(display.height//2)-
(qr_bitmap.height*3//2))
    QR_group.append(qr_tile)

    splash = displayio.Group()
    splash.append(bkgrd_tile)
    splash.append(QR_group)
    splash.append(QR_text_group)

    display.show(splash)

        # Text in the display
    display_text = "Start"
    line1_label = bitmap_label.Label(FONT,text = display_text, color =
TEXT_COLOR)
    line1_label.anchor_point = (0,0.5)
```

```
    line1_label.anchored_position =(0,0)
    line_1_group = displayio.Group(
        scale=FONTSCALE,
        x=FONTSCALE*2,
        y=display.height//2
    )
    line_1_group.append(line1_label)

    line2_label = bitmap_label.Label(FONT,text = display_text, color =
TEXT_COLOR)
    line2_label.anchor_point = (0,0.5)
    line2_label.anchored_position =(0,0)
    line_2_group = displayio.Group(
        scale=FONTSCALE,
        x=display.width //2 + FONTSCALE*2,
        y=display.height//2
    )
    line_2_group.append(line2_label)

    text_group = displayio.Group()
    text_group.append(line_1_group)
    text_group.append(line_2_group)

    gc.collect()

    ## Conncet to Adafruit IO
    mqtt_client = MQTT.MQTT(
        broker=A_secrets["broker"],
        port=A_secrets["port"],
        username=A_secrets["io_user"],
        password=A_secrets["io_key"],
        socket_pool=pool,
        ssl_context=create_default_context()
    )
    A_io = IO_MQTT(mqtt_client)

        # Connect base callbacks
    A_io.on_connect = Callback.conected
    A_io.on_disconnect = Callback.disconnected
    A_io.on_subscribe = Callback.subscribe
    A_io.on_unsubscribe = Callback.unsubscribe
    A_io.on_message = Callback.on_message

    try:
        # The Connection
        print("Conectandose a Adafruit IO...")
        A_io.connect()
    except:
        import os,alarm,time
```

```
        import microcontroller
        print("Error de conexión a Adafruit IO.")
        os.remove("dependencies/Adafruit_secrets.json")
        sleep_10=alarm.time.TimeAlarm(monotonic_time = time.monotonic()+5)
        alarm.light_sleep_until_alarms(sleep_10)
        microcontroller.reset()

        # Add callbaks to the feeds
    A_io.add_feed_callback("selector-aparato", schedule.toggle_selector)
    A_io.add_feed_callback("hora-inicio",schedule.new_init1)
    A_io.add_feed_callback("hora-apagado",schedule.new_end1)
    A_io.add_feed_callback("hora-inicio-2",schedule.new_init2)
    A_io.add_feed_callback("hora-apagado-2",schedule.new_end2)
    A_io.add_feed_callback("totp-input",schedule.new_code)
        # Subscrite to the feeds
    A_io.subscribe("selector-aparato")
    A_io.subscribe("hora-inicio")
    A_io.subscribe("hora-apagado")
    A_io.subscribe("hora-inicio-2")
    A_io.subscribe("hora-apagado-2")
    A_io.subscribe("totp-input")
    A_io.subscribe("estado-aparato-1")
    A_io.subscribe("estado-aparato-2")
    A_io.subscribe("text-messages")
        # Get the actual TOTP data
    schedule.code = A_io.get("totp-input")
        # Sincronize the states feeds with the initials
    A_io.publish("estado-aparato-1",device1.state)
    A_io.publish("estado-aparato-2",device2.state)

        # Publish the first message
    A_io.publish("text-messages", schedule.global_message)
        # Sincronize the Selector
    A_io.publish("selector-aparato",schedule.value)

        # Sincronize the rtc
    mqtt_client.add_topic_callback("time/seconds",Callback.update_rtc)
    A_io.subscribe_to_time('seconds')
    time.sleep(2)   # Wait to obtains almost one message
    mqtt_client.unsubscribe("time/seconds")
    gc.collect()

    ## Main loop
    ref_time = time.monotonic()
    running_check_reference = time.monotonic()
    RESET_TIME_REFERENCE = time.monotonic()
    while True:
        try:
            A_io.loop()
```

```
            interval = time.monotonic() - ref_time   # Update time
            running_check_interval = time.monotonic() - running_check_reference

            if RESET_INPUT.value:   # RESET Count
                RESET_TIME_REFERENCE = time.monotonic()
            RESET_INTERVAL = time.monotonic() - RESET_TIME_REFERENCE

            if RESET_INTERVAL > RESET_TIME:
                state = 10
                line1_label.text = "RESET!!!!"
                line2_label.text = "RESET!!!!"
                try:
                    splash.remove(QR_group)
                    splash.remove(QR_text_group)
                    splash.append(left_rect)
                    splash.append(right_rect)
                    splash.append(text_group)

                    del QR_group
                    del QR_text_group
                    del QR_text
                    del qr_tile
                    del qr_bitmap
                    del pallete
                except:
                    pass
                gc.collect()

            print("RESET waiter:",RESET_INTERVAL)
            print("State:",state)
            print("===============")
            gc.collect()


            if not state:   #State = 0 -> Starting: Show the QR until the TOTP
key was in
                if schedule.code == schedule.TOTP_2FA.totp():
                    state = 1
                    splash.remove(QR_group)
                    splash.remove(QR_text_group)
                    splash.append(left_rect)
                    splash.append(right_rect)
                    splash.append(text_group)

                    del QR_group
                    del QR_text_group
                    del QR_text
                    del qr_tile
                    del qr_bitmap
```

```
                    del pallete
                    gc.collect()



            elif state == 1:    #State = 1 -> Main state

                if actual_tt[0] != schedule.timetable1:
                    gc.collect()
                    line1_label.text = "Device 1:\nI1:" +
str(schedule.timetable1[0]) + "\nF1:" + str(schedule.timetable1[1]) + "\nI2:" +
str(schedule.timetable1[2]) + "\nF2:" + str(schedule.timetable1[3])
                    actual_tt[0] = schedule.timetable1.copy()

                elif actual_tt[1] != schedule.timetable2:
                    gc.collect()
                    line2_label.text = "Device 2:\nI1:" +
str(schedule.timetable2[0]) + "\nF1:" + str(schedule.timetable2[1]) + "\nI2:" +
str(schedule.timetable2[2]) + "\nF2:" + str(schedule.timetable2[3])
                    actual_tt[1] = schedule.timetable2.copy()
                gc.collect()

                # if is time to running check
                if running_check_interval > running_check_time*60:
                    state = 3   # output1 chach state

                # if a minute has passed
                if interval>=60:
                    state = 2   # Check the timetables



            elif state == 2:   # Check the timetables
                led.value = False
                ref_time = int(time.monotonic())  # New time reference
                now = time.time()
                # Publish the text message
                A_io.publish("text-messages", schedule.global_message)

                # check if it's time to do something (Device 1)
                is_ON=
Callback.update_device(schedule.timetable1,now,offset_hour,offset_minute)
                if is_ON:
                    if Error1 and device1.state:
                        device1.update()
                    if not device1.state:
                        output1.value = True
                        line1_label.color=ON_TEXT_COLOR
                        device1.update()
                        state = 3
```

```
                else:
                    if Error1 and (not device1.state):
                        device1.update()
                    if device1.state:
                        output1.value = False
                        line1_label.color=OFF_TEXT_COLOR
                        device1.update()
                        state = 3

                # check if it's time to do something (Device 2)
                is_ON=
Callback.update_device(schedule.timetable2,now,offset_hour,offset_minute)
                if is_ON:
                    if Error2 and device2.state:
                        device2.update()
                    if not device2.state:
                        output2.value = True
                        line2_label.color=ON_TEXT_COLOR
                        device2.update()
                        state = 4
                else:
                    if Error2 and (not device2.state):
                        device2.update()
                    if device2.state:
                        output2.value = False
                        line2_label.color=OFF_TEXT_COLOR
                        device2.update()
                        state = 4

                if state == 2:  # if nothing change -> Back to wait
                    state = 1
                gc.collect()



            elif state == 3:    # Check the output1 is working
                rele_state =
Callback.detect_device_iterative(input1,3,0.7,500,0.5)
                if rele_state == device1.state:
                    Error1 = 0
                    led.value = True
                    A_io.publish("estado-aparato-1",device1.state)
                else:
                    led.value = False
                    Error1=1
                    A_io.publish("estado-aparato-1","E")
                    output1.value = False
                state = 1
                if running_check_interval > running_check_time*60:
```

```
                    state = 4



            elif state == 4:    # Check the output2 is working
                rele_state =
Callback.detect_device_iterative(input2,3,0.7,500,0.5)
                if rele_state == device2.state:
                    led.value = True
                    Error2=0
                    A_io.publish("estado-aparato-2",device2.state)
                else:
                    led.value = False
                    Error2=1
                    A_io.publish("estado-aparato-2","E")
                    output2.value = False
                state = 1
                if running_check_interval > running_check_time*60:
                    running_check_reference = time.monotonic()

            elif state == 10:
                from os import remove
                from microcontroller import reset
                time.sleep(5)
                remove(file_name)
                remove("run.json")
                remove("dependencies/secretos.json")
                remove("dependencies/Adafruit_secrets.json")
                remove("dependencies/secret_key.json")
                reset()

            # update the display
            display.show(splash)

            # Save the actual data in a file
            load = {
                "timetable1":schedule.timetable1,
                "timetable2":schedule.timetable2,
                "TOTP_key":schedule.TOTP_key
            }
            with open("datos.json","w") as file:
                dump(load,file)

            gc.collect()
            # Time to sleep
            sleep_1  =alarm.time.TimeAlarm(monotonic_time=time.monotonic()+1)
            alarm.light_sleep_until_alarms(sleep_1)

        except (ValueError, RuntimeError) as e:
            print("Failed to get data, retrying\n", e)
```

```
            radio.connect(secrets["ssid"], secrets["password"])
            A_io.reconnect()
            continue

except ConnectionError:
    import os,time
    import microcontroller

    sleep_10=alarm.time.TimeAlarm(monotonic_time = time.monotonic()+10)
    alarm.light_sleep_until_alarms(sleep_10)

    print("Error de conexión.")
    os.remove("dependencies/secretos.json")
    microcontroller.reset()

except OSError as ERROR:
    while True:
        if ERROR.args[0] == 30:
            print("Read Only File System")
        else:
        # print("Error. Sistema en modo de solo lectura")
            print(ERROR.strerror,ERROR.args[0])
            print("ErrorRRRRRRRRRRRRR")
        sleep_10=alarm.time.TimeAlarm(monotonic_time = time.monotonic()+10)
        alarm.light_sleep_until_alarms(sleep_10)
    # microcontroller.reset()
except Exception as err:
    print("Error inesperado: ",err.args)
    sleep_10=alarm.time.TimeAlarm(monotonic_time = time.monotonic()+10)
    alarm.light_sleep_until_alarms(sleep_10)
```

AIO_Callbacks

Pequeña librería con funciones usadas en el código

```
# Callbacks
def conected(client):
    print("Conectado a Adafruit IO!")

def disconnected(client):
    # Disconnected function will be called when the client disconnects.
    print("Desconectado de Adafruit IO!")

def subscribe(client, userdata, topic, granted_qos):
    # This method is called when the client subscribes to a new feed.
    print("Suscrito al feed {0}".format(topic))
def unsubscribe(client, userdata, topic, pid):
    print("Desuscrito del feed {0}".format(topic))

def on_message(client, feed_id, payload):
    # Message function will be called when a subscribed feed has a new
value.
    # The feed_id parameter identifies the feed, and the payload parameter
has
    # the new value.
    print("Feed {0} received new value: {1}".format(feed_id, payload))

# This class store the timetables of both devices and the metohds to update
them
# when it is possible, as well as store the value of the switch and make
the
# multiplexation of feeds
# Also control the TOTP 2FA
class Schedule():
    def
__init__(self,initial_value,timetable1:list,timetable2:list,TOTP_key: str)
-> None:
        from dependencies.totp import TOTP
        self.value = initial_value
        self.timetable1=timetable1.copy()
        self.timetable2=timetable2.copy()
        self.TOTP_key = TOTP_key
        self.TOTP_2FA = TOTP(TOTP_key)
        self.code=0
        self.message1="Start(1): " + str(self.timetable1)
        self.message2="Start(2): " + str(self.timetable1)
        self.global_message = "Starting..."

    # Toggle the switch variable when recive a message of certain feed
    def toggle_selector(self,client,topic,message):
        self.value = int(message)
```

```
        if not self.value:
            self.global_message = self.message1
        else:
            self.global_message = self.message2

    # update the firts value of the timetable if it is possible
    def new_init1(self,client,topic,message: str):

        if self.code==self.TOTP_2FA.totp():
            from math import modf
            message = message.replace(",",".")
            new = float(message)

            new_minute,new_hour = modf(new)
            new_minute = round(new_minute*100,2)
            new_hour = round(new_hour,2)
            minute = new_minute%60
            hour = (new_hour + (new_minute//60))%24

            new = hour+minute/100
            if not self.value:
                if new < self.timetable1[1]:
                    update_timetable(self.timetable1,0,new)
                    self.message1 = "Update 1: "+str(self.timetable1)
                    self.global_message = self.message1
                else:
                    self.message1 = "Time error: "+str(self.timetable1)
                    self.global_message = self.message1
            else:
                if new < self.timetable2[1]:
                    update_timetable(self.timetable2,0,new)
                    self.message2 = "Update 2: "+str(self.timetable2)
                    self.global_message = self.message2
                else:
                    self.message2 = "Time error: "+str(self.timetable2)
                    self.global_message = self.message2
        else:
            self.global_message = "Error TOTP"

    # update the second value of the timetable if it is possible
    def new_end1(self,client,topic,message: str):

        if self.code==self.TOTP_2FA.totp():
            from math import modf
            message = message.replace(",",".")
            new = float(message)

            new_minute,new_hour = modf(new)
            new_minute = round(new_minute*100,2)
```

```
            new_hour = round(new_hour,2)
            minute = new_minute%60
            hour = (new_hour + (new_minute//60))%24

            new = hour+minute/100
            if not self.value:
                if new > self.timetable1[0]:
                    update_timetable(self.timetable1,1,new)
                    self.message1 = "Update 1: "+str(self.timetable1)
                    self.global_message = self.message1
                else:
                    self.message1 = "Time error: "+str(self.timetable1)
                    self.global_message = self.message1
            else:
                if new > self.timetable2[0]:
                    update_timetable(self.timetable2,1,new)
                    self.message2 = "Update 2: "+str(self.timetable2)
                    self.global_message = self.message2
                else:
                    self.message2 = "Time error: "+str(self.timetable2)
                    self.global_message = self.message2
        else:
            self.global_message = "Error TOTP"

    # update the third value of the timetable if it is possible
    def new_init2(self,client,topic,message: str):

        if self.code==self.TOTP_2FA.totp():
            from math import modf
            message = message.replace(",",".")
            new = float(message)

            new_minute,new_hour = modf(new)
            new_minute = round(new_minute*100,2)
            new_hour = round(new_hour,2)
            minute = new_minute%60
            hour = (new_hour + (new_minute//60))%24

            new = hour+minute/100
            if not self.value:
                if new < self.timetable1[3]:
                    update_timetable(self.timetable1,2,new)
                    self.message1 = "Update 1: "+str(self.timetable1)
                    self.global_message = self.message1
                else:
                    self.message1 = "Time error: "+str(self.timetable1)
                    self.global_message = self.message1
            else:
                if new < self.timetable2[3]:
```

```
                    update_timetable(self.timetable2,2,new)
                    self.message2 = "Update 2: "+str(self.timetable2)
                    self.global_message = self.message2
                else:
                    self.message2 = "Time error: "+str(self.timetable2)
                    self.global_message = self.message2
        else:
            self.global_message = "Error TOTP"

    # update the fourth value of the timetable if it is possible
    def new_end2(self,client,topic,message: str):

        if self.code==self.TOTP_2FA.totp():
            from math import modf
            message = message.replace(",",".")
            new = float(message)

            new_minute,new_hour = modf(new)
            new_minute = round(new_minute*100,2)
            new_hour = round(new_hour,2)
            minute = new_minute%60
            hour = (new_hour + (new_minute//60))%24

            new = hour+minute/100
            if not self.value:
                if new > self.timetable1[2]:
                    update_timetable(self.timetable1,3,new)
                    self.message1 = "Update 1: "+str(self.timetable1)
                    self.global_message = self.message1
                else:
                    self.message1 = "Time error: "+str(self.timetable1)
                    self.global_message = self.message1
            else:
                if new > self.timetable2[2]:
                    update_timetable(self.timetable2,3,new)
                    self.message2 = "Update 2: "+str(self.timetable2)
                    self.global_message = self.message2
                else:
                    self.message2 = "Time error: "+str(self.timetable2)
                    self.global_message = self.message2
        else:
            self.global_message = "Error TOTP"

    # updathe the code for the TOTP 2FA
    def new_code(self,client,topic,message):
        self.code=message

# Help with the update in schedules
def update_timetable(timetable,updated_time,new_hour):
```

```
    timetable[updated_time]=new_hour

# Update the rtc in function of the message received
def update_rtc(client,topic,message):
    from time import localtime
    from rtc import RTC
    rtc = RTC()
    rtc.datetime = localtime(int(message))


# This Class store and manage the state of a device
class state_machine:
    def __init__(self,initial_state,n_states) -> None:
        self.n_states = n_states
        self.state = initial_state

    def update(self):
        self.state=(self.state+1)%self.n_states

# Get the hous and minutes of a specific time offset
def get_hmoffset(time_offset:str):
    from math import modf
    time_offset = time_offset.replace(":",".")
    if time_offset[0]=="+":
        offset = float(time_offset[1:-1])
        offset_minute,offset_hour = modf(offset)
    else:
        offset = float(time_offset)
        offset_minute,offset_hour = modf(offset)
    # print ("Off_hour", offset_hour,"Off_min",offset_minute)
    return offset_hour,offset_minute


# Make changes in state machines if it is necesary in function of schedules
def update_device(timetable:list ,UTC_time:
int,UTC_offset_hour,UTC_offset_minute):
    from time import localtime
    UTC_hour = localtime(UTC_time)[3]
    UTC_minute = localtime(UTC_time)[4]

    # Make local time in function of actual UTC time and a offset
    local_minute = (UTC_minute + UTC_offset_minute)%60
    local_hour = (UTC_hour + UTC_offset_hour + (UTC_minute +
UTC_offset_minute)//60)%24
    local_time = local_hour + (local_minute/100)

    # Print some stuff
    print("Local time:",local_time)
    print("Limits: ",timetable)
```

```
    if ((local_time >= timetable[0]) and (local_time <= timetable[1])): #
If it is in the 1st ON time
        print("Led ON")
        return True             # return if I can switch the led and
publish in the feed
    elif ((local_time >= timetable[2]) and (local_time <= timetable[3])):#
If it is in the 2nd ON time
        print("Led ON")
        return True
    else:   # Else, it is on a OFF time
        return False

# Make bitmap object that encode the bits in a matrix os QR
def bitmap_QR(usser,account,data):
    from displayio import Bitmap
    from adafruit_miniqr import QRCode

        # making the QR code of the TOTP
    uri =
"otpauth://totp/{0}%20({1})?secret={2}&iusser={0}&algorithm=SHA1&digits=6&p
eriod=30"
    qr_uri = uri.format(usser,account,data)
    TOTP_qr = QRCode()
    TOTP_qr.add_data(bytearray(map(ord,qr_uri)))
    TOTP_qr.make()
    matrix = TOTP_qr.matrix
    # monochome (2 color) palette
    BORDER_PIXELS = 2

    # bitmap the size of the screen, monochrome (2 colors)
    bitmap = Bitmap(
        matrix.width + 2 * BORDER_PIXELS, matrix.height + 2 *
BORDER_PIXELS, 2
    )
    # raster the QR code
    for y in range(matrix.height):  # each scanline in the height
        for x in range(matrix.width):
            if matrix[x, y]:
                bitmap[x + BORDER_PIXELS, y + BORDER_PIXELS] = 1
            else:
                bitmap[x + BORDER_PIXELS, y + BORDER_PIXELS] = 0
    return bitmap

def initialize_display(rotation:int):
    import board
    from displayio import release_displays,FourWire
    from busio import SPI
    from adafruit_st7789 import ST7789
```

```
    release_displays()

    tft_cs = board.GP17
    tft_dc = board.GP16
    spi_mosi = board.GP19
    spi_clk = board.GP18
    spi = SPI(spi_clk, spi_mosi)

    display_bus = FourWire(spi, command=tft_dc, chip_select=tft_cs)
    display = ST7789(
        display_bus, rotation=rotation, width=240, height=135, rowstart=40,
colstart=53
    )
    return display


def Random_32_Key_generator(length: int):
    from os import urandom

    dict = "ABCDEFGHIJKLMNOPQRSTUVWXYZ234567"
    passw = []
    for i in range(length):
        random_number = int.from_bytes(urandom(1),"little")%32
        passw.append(dict[random_number])

    final_pass = ''.join(passw)
    return final_pass


def detect_device(input,num_samples: int =500,time:int = 0.5, threshold:int
= 10250):
    from time import sleep

    values = [] # list of samples

    for i in range(num_samples):    # Take the sample in the time
        values.append(input.value)
        if len(values)>2:
            values.sort()   # Order the values from smallest to largest
            values.pop(1)   # Delate the value of the center
        sleep(time/num_samples)

    # values.sort()

    Max = values[-1]    # Select the max value
    Min = values[0]     # Select the min value

    Diff = Max-Min      # Calculates the amplitude
    if Diff > threshold:
        return True
```

```
    else:
        return False

def detect_device_iterative(input,iterations : int =
3,positives_tries_percentage:float = 0.7,num_samples: int =500,time:int =
0.5, threshold:int = 10250):
    # Ensure that psitives_tries_percentage is between 0 and 1
    if positives_tries_percentage<0 or positives_tries_percentage>1:
        raise SyntaxError("positives_tries_percentage must be between 0 and
1")

    list = []
    for interation in range(iterations):    # Run the detect function
iterations times
        list.append(detect_device(input,num_samples,time,threshold))

    if list.count(True) >= int(iterations*positives_tries_percentage):  #
If a percentage is True, return True
        return True
    else:
        return False
```