Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Latch SwimmingPool Door - Sistema de control de acceso para prevención de incidentes en piscinas

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Manuel Tinoco Escorihuela

Tutor: José Vicente Busquets Mataix

Curso 2016 - 2017

Agradecimientos

En primer lugar, me gustaría agradecer a mis padres su apoyo incondicional durante toda mi trayectoria académica, desde mis inicios hasta la actualidad, siempre ayudando en todo lo posible, sin importar la situación.

En segundo lugar, me gustaría dar las gracias a una persona especial, que ha vivido en primera persona todos estos años de esfuerzo, sufrimiento y felicidad que he pasado en la Universidad. Siempre dando el apoyo necesario en los momentos más difíciles y confiando en mi cuando más lo necesitaba.

Por otro lado, me gustaría agradecer también a mi hermana y mi cuñado, siempre disponibles en caso de necesidad.

Además, quiero agradecer a mi tutor, José Vicente, tanto su completa disposición en cualquier duda surgida durante el desarrollo del proyecto, como las enseñanzas transmitidas desde el primer curso de carrera hasta el final de la misma.

Para no olvidarme de nadie, me gustaría, por último, agradecer a todos esos compañeros y compañeras de batallas que han hecho que el camino hasta aquí haya sido mucho más ameno.

Resumen

El principal objetivo de este proyecto es implementar un sistema lo más económico y sencillo posible que permita tener una seguridad adicional en el acceso a zonas potencialmente peligrosas de piscinas privadas.

Para ello, se utilizará un sistema ya disponible en la red, propiedad de Telefónica, llamado Latch.

Latch es un sistema que proporciona un segundo factor de autenticación, funcionando como un pestillo digital. Este sistema, es configurado desde un dispositivo móvil, pudiendo autorizar o denegar la entrada a las zonas protegidas cuando el propietario lo desee.[1]

Con esta tecnología a nuestra disposición, se analizarán los requisitos necesarios para desarrollar un sistema basado en la plataforma de hardware libre Arduino, así como los módulos complementarios necesarios disponibles para el desarrollo del proyecto.

Una vez implementado e instalado el sistema, será el mismo, el que autorice el acceso a las zonas restringidas cuando alguien pulse el botón de apertura de la puerta, permitiendo o denegando el acceso, y notificándolo mediante un mensaje push al propietario el intento de acceso.

Palabras clave: Seguridad adicional, Arduino, Latch, Segundo factor de autenticación, dispositivo móvil, piscinas privadas.

Resum

L'objectiu principal d'aquest projecte és implementar un sistema el més econòmic i senzill possible que permeta obtindre una seguretat addicional en l'accés a zones potencialment perilloses de piscines privades.

Per a açò, s'utilitzarà un sistema ja disponible en la xarxa, propietat de Telefónica, anomenat Latch.

Latch és un sistema que proporciona un segon factor d'autenticació, funcionant com un pestell digital. Aquest sistema, és configurat des d'un dispositiu mòbil, puguent autoritzar o denegar l'entrada a les zones protegides quan el propietari ho desitge. [1]

Amb aquesta tecnologia a la nostra disposició, s'analitzarán els requisits necessaris per a desenvolupar un sistema basat en la plataforma de hardware lliure Arduino, així com els mòduls complementaris necessaris disponibles per al desenvolupament del projecte.

Una vegada implementat i instal·lat el sistema, serà el mateix, el que autoritzarà l'accés a les zones restringides quan algú prema el botó d'obertura de la porta, permetent o denegant el mateix i notificant-ho mitjançant un missatge push al propietari amb l'intent d'accés.

Paraules clau: Seguretat addicional, Arduino, Latch, Segon factor d'autenticació, dispositiu mòbil, piscines privades.

Abstract

The main objective of this project is to implement a chapest and simplest system to provide additional security in access to potentially dangerous areas of private swimming pools.

For this purpose, will be used a system already available on the network owned by Telefónica, called Latch.

This system is configured from a mobile device, and may authorize or deny entry to protected areas when the owner so desires. [1]

With this available technology , the necessary requirements will be analyzed to develop a system based on the free hardware platform Arduino, as well as the necessary complementary available modules for the development of the project.

Once the system will be implemented and installed, it will manage access to restricted areas when someone presses the door open button, allowing or denying access, and notifying access attempt by a push message to the owner.

Keywords: Additional security, Arduino, Latch, Second authentication factor, mobile device, private pools.

Tabla de contenidos

Índice de tablas

Tabla de figuras

1. Introducción

1.1 Motivación

Dados los crecientes incidentes en piscinas privadas que cada año se traducen en un mayor número de niños/as fallecidos durante la época estival (según la OMS, está entre las tres principales causas de muerte y discapacidad de por vida en niños/as de 5 a 15 años)[10], se ha pensado en una posible solución tecnológica que ofrezca un segundo factor de seguridad ante los posibles descuidos de las personas responsables del cuidado de los mismos.

Este proyecto pretende impedir accesos no deseados a las zonas de las piscinas que supongan un claro riesgo de incidente mediante la autorización o rechazo del acceso al recinto de una forma remota.

Se ha pensado en un proyecto que tenga un coste bajo de adquisición del producto con el objetivo de hacer accesible a todos los públicos la instalación del sistema.

1.2 Objetivos

El principal objetivo de este proyecto es reducir los incidentes de niños/as en piscinas privadas. Para ello, se pretende desarrollar un sistema que permita el control del acceso a las zonas que puedan suponer un incremento del riesgo de accidente.

Para el correcto funcionamiento del sistema, se va a implementar un circuito electrónico, con sus componentes que realizaran las funciones de la cerradura del acceso al recinto. Este circuito, basado en la plataforma Arduino, se encargará de comprobar el estado de autorización de acceso consultando el mismo vía WiFi a un servidor que contiene la configuración realizada por el propietario del sistema en su teléfono móvil.

Por otro lado, se va a implementar un servidor intermedio que recibirá las peticiones del sistema electrónico y realizará la conexión con los servidores de Latch, salvando así los problemas relacionados con los certificados que pueden surgir con el uso de placas de Arduino[3]. Este servidor recibirá el código de identificación de la placa y consultará el estado a los servidores de Latch, que a su vez responderán con el estado configurado al servidor intermedio, tras lo que este, responderá a la placa Arduino, autorizando o denegando el acceso.

1.3 Planificación

Con el objetivo de desarrollar el presente proyecto dentro de los plazos estipulados, se ha planificado el desarrollo siguiendo las siguientes fases:

- Análisis de requisitos y limitaciones teniendo en cuenta las funcionalidades necesarias.
- Estudio de las soluciones similares disponibles en el mercado.
- Instalación, configuración y programación del servidor.
- Adquisición de electrónica necesaria para el montaje del proyecto.
- Análisis de requisitos y necesidades de los módulos electrónicos.
- Implementación de código necesario para la programación de los componentes hardware.
- Configuración de cuenta para comprobar el correcto funcionamiento del proyecto.
- Montaje del circuito.
- Pruebas y validación de resultados.

2. Análisis y diseño

En este apartado, se analizan los requisitos que debe cumplir el proyecto, así como las estrategias a seguir para que el proyecto cumpla con estos requisitos al final del desarrollo del mismo.

2.1 Requisitos del proyecto

El proyecto debe cumplir una serie de requisitos para que funcione correctamente.

Estos requisitos, se pueden dividir entre requisitos funcionales, y no funcionales.

2.1.1 Requisitos funcionales

- Los componentes deben ser lo más pequeños posibles.
- Los componentes deben ser fácilmente sustituibles.
- El sistema debe tener un pulsador fácilmente accesible.
- El sistema controlará el acceso y lo permitirá solamente cuando esté autorizado.
- Debe informar al propietario del intento de acceso cuando se deniegue el acceso.

2.1.2 Requisitos no funcionales

- La instalación debe ser sin cables.
- Se debe poder autorizar o denegar el acceso mediante un dispositivo móvil.
- El sistema debe ofrecer una solución en caso de no estar disponible.
- Debe ser lo más económico posible.
- Debe ser sencillo de instalar en el vallado de protección de la piscina.
- Debe responder lo antes posible cuando se accione.

2.2 Análisis y justificación del diseño

Para el desarrollo de nuestro sistema, necesitamos contar con los siguientes componentes:

- Placa Arduino
- Modulo de conexión WiFi
- Motor de accionamiento de cerradura
- Pulsador

Teniendo en cuenta estos componentes, se han elegido, de entre los disponibles en el mercado, los que cumplen con los requisitos necesarios y son, al mismo tiempo, lo más económicos posibles.

La placa Arduino elegida va en función de la potencia deseada, así como de los pines necesarios para interconectar todos los módulos necesarios para el desarrollo del proyecto.

En primer lugar, el módulo de conexión WiFi elegido ha sido el ESP8266. Este módulo cuenta con lo necesario para dotar de conexión WiFi al sistema y es uno de los más utilizados y económicos dentro de los disponibles del mercado.

Necesita de alimentación externa para su arranque debido a los picos de tensión que utiliza en la inicialización, por lo que se ha optado por utilizar una fuente de alimentación externa de las más asequibles.

El ESP8266 utiliza 3 pines de la placa de Arduino, de los cuales 2 son para la conexión serial y el otro para el pin de selección.

Por otro lado, el pulsador utiliza dos un pin de la placa de Arduino para detectar cuando se cierra el circuito.

Por último, el motor de accionamiento elegido ha sido el 28BYJ-48, un motor unipolar paso a paso de gran precisión y bajo consumo. este utiliza 4 pines de la placa Arduino para excitar las 4 bobinas que controlan la rotación del mismo.

Por tanto tenemos, 3 pines para el módulo ESP8266, 4 pines para el motor unipolar, 1 pin para el pulsador y dos adicionales para el conversor USB-TTL que se va a utilizar durante la implementación, lo que hace un total de 10 pines necesarios para la realización del proyecto.

Teniendo en cuenta el numero de pines necesarios y el objetivo de diseñar un sistema lo más económico posible, se ha optado por la placa Arduino UNO R3, que cuenta con todo lo necesario.

3. Material utilizado

A continuación, se describen los componentes físicos que se han utilizado en el proyecto. Como componente base, tenemos la placa Arduino. El resto de componentes, se han elegido teniendo en cuenta los requisitos analizados en el apartado que precede.

3.1 Arduino Uno R3

Arduino es una plataforma de hardware libre que se enfoca en acercar y facilitar el uso de la electrónica y la programación de los sistemas embebidos al público en general. Esta plataforma está liberada, en su totalidad, con licencia de código abierto. Además dispone de un entorno de desarrollo (IDE) que permite compilar el código para la placa seleccionada.

La plataforma está basada en una placa con un microcontrolador, que junto con el IDE propio, le aporta una versatilidad que la sitúa como una de las mejores soluciones de hardware para proyectos relativamente sencillos.

Esta cuenta con una gran variedad de placas de diferentes tamaños y complejidad, diseñadas con el objetivo de ajustarse al máximo a las necesidades de cada usuario. En este proyecto, se ha optado por utilizar el Arduino Uno R3, una placa basada en el microcontrolador ATmega328 . Se ha optado por esta placa dado que cuenta con la cantidad de pines necesarios para la realización del proyecto y con la suficiente cantidad de memoria para manejar las librerías necesarias y almacenar el código necesario.

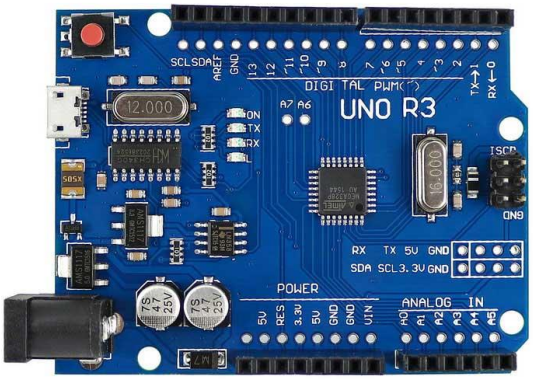


Figura 1: Arduino UNO R3

Como características destacables de nuestra placa, tenemos las siguientes:

Característica	Descripción
Microcontrolador	ATmega 328
Voltaje de trabajo	5 V
Voltaje de entrada	7-12 V
Alimentación de la placa	Mediante cable USB o entrada dedicada
Pines digitales	14 pines de I/O
Pines digitales con soporte PWM	6
Pines de entrada analógica	6
Memoria flash	32 KB
Velocidad de reloj	16 MHz
Tensiones de salida de alimentación	5 y 3.3 V

Tabla 1: Características Arduino UNO R3

3.2 Módulo WiFi

Para poder dotar de conectividad a la red al sistema, evitando el despliegue de cableado, se ha optado por recurrir a el conocido modulo ESP8266 que permite la conectividad WiFi del sistema. Este modulo tiene dos modos de trabajo. Por un lado, puede trabajar como esclavo o como hardware independiente, ya que está dotado de una memoria flash que le permite almacenar y ejecutar código por sí mismo.

Este chip es un SoC con la pila de protocolos TCP/IP integrada. En el mercado, se puede encontrar integrado en diferentes módulos, que le proporcionan pines para su conexión a un dispositivo maestro.[14]

Suele traer preinstalado un firmware que permite la comunicación mediante comandos AT con el mismo. Este firmware dota al dispositivo de diferentes funcionalidades, tales como servir como cliente HTTP o incluso levantar un servidor web. No obstante, para simplificar la utilidad del mismo, se hace necesario sustituir este firmware por otro con que junto con una librería especifica, facilita el uso de las funciones del mismo.

Es necesario tener en cuenta que este chip trabaja con un voltaje de 3.3V, por lo que, si no incluimos elementos de protección al mismo al conectarlo a cualquier sistema electrónico que no trabaje en estos niveles de tensión, podríamos quemarlo y dejarlo inservible.

3.2.1 ESP8266 versión 01

Es el más económico de los disponibles, no obstante, ofrece unas características válidas para el tipo de proyecto que se ha desarrollado. Dispone de dos pines GPIO y de los comunes TX y RX utilizados por la interfaz serial para comunicarse con el mismo.

Es ideal para funcionar como esclavo de otro más potente (en nuestro caso de Arduino). Aunque también es capaz de funcionar de forma autónoma, está bastante limitado por el numero de pines GPIO de los que dispone.

Dadas sus características se ha utilizado como esclavo de la placa Arduino para proporcionar a la misma conectividad de red.

Como aspecto a tener en cuenta, debemos saber que requiere de una fuente de alimentación externa a Arduino. Esto se debe al consumo que necesita el ESP8266 en su arranque. Este chip puede llegar a demandar 300 o incluso 400 mA. en su arranque, y dado que nuestra placa de Arduino tiene una corriente máxima de salida de 200mA, podríamos tener problemas a la hora de inicializar y arrancar el módulo.

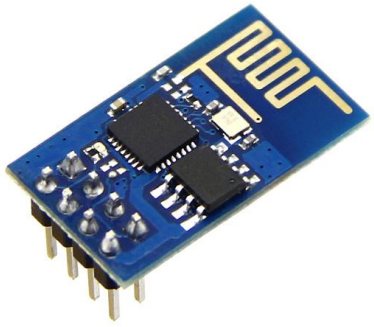


Figura 2: Módulo ESP8266 versión 01

3.3 Motor 28BYJ-48

Para la implementación de la cerradura del proyecto, se ha elegido un motor unipolar de paso a paso, en concreto, el 28BYJ-48.

Estos motores de bajo coste, son muy habituales en proyectos con pequeños robots o posicionadores, aunque no son excesivamente potentes ni rápidos, tienen unas características necesarias para el proyecto que se ha desarrollado.

Estos motores tienen las siguientes características:

- Tensión nominal de entre 5 y 12V (en nuestro caso 5).
- 4 fases o bobinas internas.

- Resistencia de 50 Ohmios.
- Par motor de 34 Newtons (0,34 Kg/cm aproximadamente).
- Consumo de unos 55 mA.
- 8 pasos por vuelta.
- Reductora de 1 / 64.

El motor presenta un conector que se puede, o bien conectar a una protoboard, o conectar directamente a un adaptador (ULN2003A) que haga las funciones de resistencia y simplifique la conexión con la placa que va a controlarlo.

Figura 3: Motor 28BYJ-48

3.4 Adaptador ULN2003A

El adaptador ULN2003A es la típica brakeboard que acompaña al motor 28BYJ-48. Está dotado de un array de transistores Darlington, que soportan hasta 500mA y dispone de un conector para el motor y unos pines (IN1-IN4) que se corresponden con las 4 bobinas, para conectar el motor al Arduino, mediante el cual excitaremos las bobinas correspondientes para producir el giro que deseemos.

Además dispone de 4 LEDs que indican la fase en la que se encuentra el motor, y que van cambiando a medida que se van excitando las diferentes bobinas.

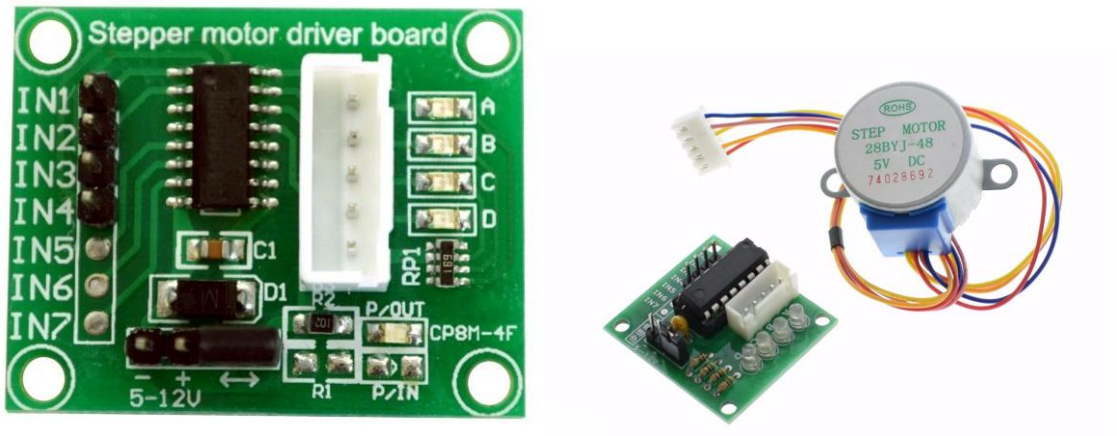


Figura 4: Adaptador ULN2003A y motor 28BYJ-48

3.4 Conversor USB a TTL

Como herramienta imprescindible para el desarrollo del proyecto para, por ejemplo, actualizar el firmware del módulo ESP8266, contamos con un convertidor USB a TTL.

Este dispositivo, simplifica las labores de comunicación con los dispositivos que utilizar las interfaces RS-232, permitiendo una conexión directa entre los mismos y otro dispositivo USB (en este caso un PC).

El conversor transforma los paquetes del protocolo USB a paquetes RS-232 y viceversa, de forma que hace posible una comunicación entre dos dispositivos, que de otra forma no sería posible.

Este incluye, además, dos pines de alimentación a 5 y a 3.3 V, acompañados de otro pin de masa, que nos permiten usar este modulo como alimentación para los módulos WiFi.

Este dispone de dos interfaces. Por un lado, la interfaz USB de la cual recibe alimentación, y por otro la interfaz RS-232. Esta última dispone de 5 pines que se detallan a continuación:

- 5V: Pin con tensión de 5V para alimentación de circuitos.
- 3.3V: Pin con tensión de 3.3V para alimentación de circuitos.
- RX: Pin de recepción de datos del protocolo RS-232.
- TX: Pin de transmisión de datos del protocolo RS-232.
- GND: Pin con tensión 0V.

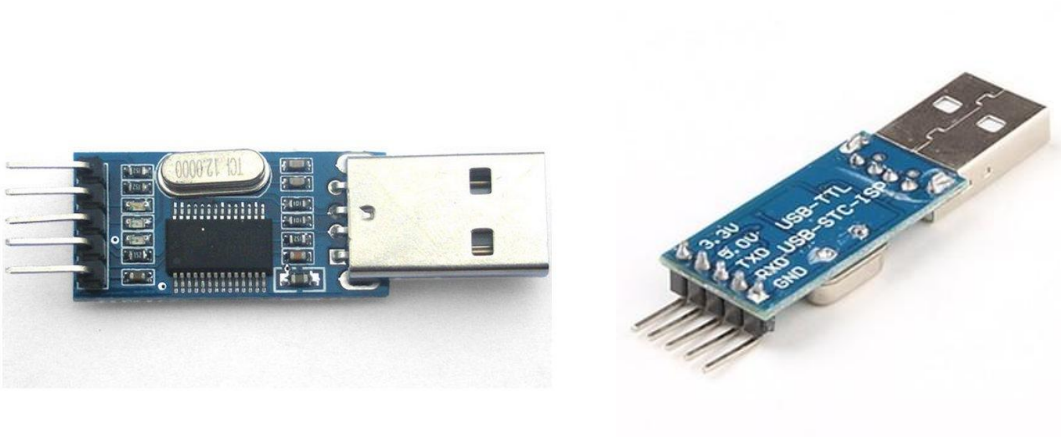


Figura 5: Conversor USB a TTL

3.5 Fuente de alimentación suplementaria

Tal y como hemos indicado anteriormente, nuestro chip ESP8266 necesita de una alimentación adicional para solventar los problemas que pueden surgir durante el arranque y la inicialización del mismo.

Para ello se ha empleado una fuente de alimentación suplementaria, capaz de proporcionar alimentación a tensiones de 3.3 y/o de 5V.

Esta fuente dispone de unos jumpers que se utilizan para seleccionar la tensión de salida deseada.

Figura 6: Fuente de alimentación suplementaria

3.6 Protoboard

En la gran mayoría de proyectos en los que la electrónica juega un papel fundamental, se utilizan las placas de prototipado o protoboards para diseñar e implementar los circuitos electrónicos.

Estas plazas están constituidas por un tablero y una serie de orificios conectados entre sí de manera interna, de forma que simplifican el proceso de prototipado, evitando la necesidad de puntos de soldadura en los casos en los que no dispongamos de el cableado necesario para interconectar dos dispositivos.

Normalmente, los orificios de conexión siguen unos patrones de líneas para simplificar las conexiones.

El patrón típico que se sigue es el siguiente:

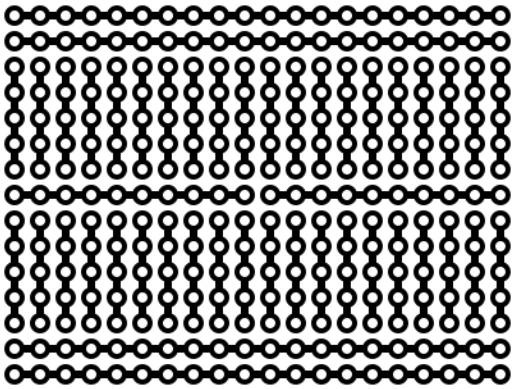


Figura 7: Patrón común protoboard

En este proyecto se ha utilizado una placa de prototipado como la que sigue:

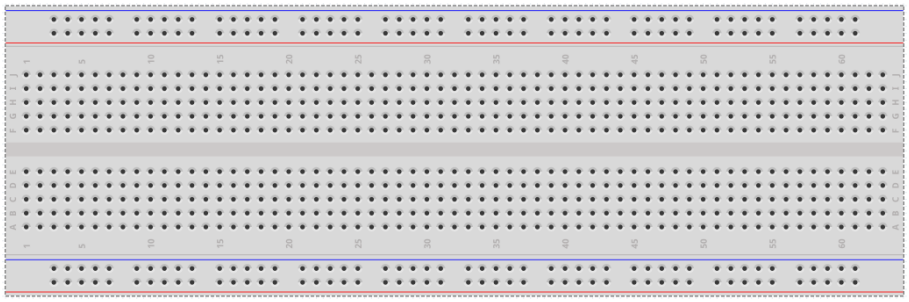


Figura 8: Protoboard

3.8 Pulsador

Para accionar el circuito, se ha optado por utilizar un pulsador de los más habituales en proyectos de electrónica.

Este consta de 4 pines que corresponden a través de los cuales fluye la corriente cuando se acciona el mismo.

Figura 9: Pulsador

4. Herramientas utilizadas

Para poder llevar a cabo el presente proyecto, es necesario contar con ciertas herramientas que nos permitan, tanto programar el hardware, como el software necesario con el fin de poner todo en funcionamiento de forma correcta.

4.1 Firmware adicional para ESP8266

Tal y como se ha indicado anteriormente, para comunicar el proyecto con el servidor intermedio que se encarga de consultar el estado del Latch configurado, se ha utilizado el chip ESP8266, el cual dispone de un firmware por defecto que permite la utilización de comandos AT. Sin embargo, para facilitar la programación del mismo, se ha optado por actualizar este firmware con una solución de un tercero como es ESP_bridge.

Hay que tener en cuenta que este firmware no cumple con los estándares de calidad, ni ha pasado pruebas de calidad para que se considere como una solución totalmente operativa. No obstante, al tratarse de un proyecto de ámbito académico, y de que el firmware es totalmente operativo, se ha utilizado con el fin de simplificar al máximo la complejidad del ESP8266.

4.1.1 ESP_bridge

Este firmware esta desarrollado para trabajar con la librería ESPduino. Este recibe las ordenes desde las funciones con comandos SLIP (estándar de encapsulación de datagramas IP sobre líneas serie - RFC 1055) y realiza las conexiones necesarias. El firmware, junto con la librería indicada, permiten al ESP8266 enviar y recibir información con la arquitectura REST.

La implementación de ESPduino, soporta el método GET y la inclusión de cabeceras personalizadas, suficiente como para realizar la petición al servidor intermedio que se comunicará con los servidores de Latch para obtener el estado actual del mismo.

4.1.2 Actualización del firmware

Para poder actualizar este nuevo firmware disponemos de dos métodos.

En primer lugar, podemos utilizar la placa de Arduino en modo USB-Serial bypass para permitir la conexión entre el módulo WiFi situando el mismo en estado de reset (llevando a LOW su entrada de RESET, conectándola a GND).

Por otro lado, y para evitar el uso de la placa de Arduino para actualizar el firmware, vamos a utilizar el conversor USB-TTL. Para ello, debemos conectar los pines de la siguiente forma:

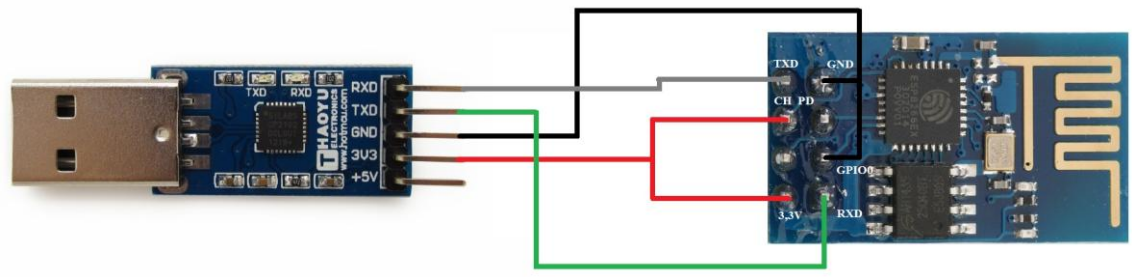


Figura 10: Cableado para sobrescribir el firmware del ESP8266

Para hacer posible la sobrescritura del firmware, es necesario conectar el pin GPIO 0 a GND, de forma que el módulo grabará en las direcciones de memoria reservadas, la información que reciba por la interfaz serie.

Una vez realizada la conexión entre el conversor y el ESP, y el conversor y el PC, es hora de enviar código hasta el ESP para sobrescribir el firmware.

Existen diferentes utilidades para flashear los módulos indicando las direcciones de memoria en las que sobrescribir, no obstante, vamos a utilizar una utilidad desarrollada en Python que está incluida en la librería ESPduino con este fin. Con esta utilidad es posible enviar ficheros binarios a direcciones de memoria determinadas.

Según la documentación de la misma, el formato para la orden es el siguiente:

[nombre utilidad] -p [puerto] write_flash [dirección de memoria] [ruta fichero] ...

En concreto, para la realización del proyecto, se ha utilizado el siguiente comando:

python esptool.py -p COM7 write_flash 0x00000 ../release/0x00000.bin 0x40000 ../release/0x40000.bin

Cuando el proceso ha finalizado, es necesario desconectar el pin GPIO 0 de GND, de lo contrario, el módulo ESP8266 se inicializará de nuevo en modo flash.

Una vez actualizado el firmware, ya podemos comenzar a programar el sketch de Arduino para darle la funcionalidad deseada.

4.2 Herramientas de programación del hardware

A continuación se hablará de las herramientas que nos permiten tanto implementar, como cargar y analizar el código necesario para que funcionen los componentes hardware del sistema. En nuestro caso la placa Arduino, el ESP8266 en modo esclavo y el motor 28BYJ-48.

4.2.1 Plataforma Arduino

La plataforma Arduino está basada en el lenguaje de programación C, por ello, soporta todas las funciones tanto de C, como de C++.

Los programas desarrollados y compilados en la plataforma Arduino, están enlazados con la librería AVR Libc.

Esta librería Open Source, tiene como objetivo proporcionar la posibilidad de programar en lenguajes de alto nivel en los microcontroladores Atmel AVR para que puedan utilizar el compilador GCC.

Las que siguen, son las 3 principales partes de la librería:

- avr-gcc: Compilador de C para microcontroladores con arquitectura AVR.
- avr-libc: Contiene librerías estáticas y los archivos necesarios para el proceso de compilación.
- avr-binutils: Es una colección de programas para manipular objetos binarios desarrollador para microcontroladores con arquitectura Atmel AVR.

4.2.1.1 Librerías

En Arduino, al igual que en el resto de lenguajes de programación, es posible incluir y utilizar librerías propias que proporcionen funcionalidad extra, favoreciendo el desarrollo de aplicaciones modulares y ahorrando la repetición de código.

La plataforma Arduino, dispone de una serie de librerías predeterminadas, no obstante, es posible importar librerías de terceros o desarrollar librerías propias.

4.2.1.2 Programación en Arduino

Arduino está basado en el lenguaje y entorno Wiring. Este entorno está, a su vez, inspirado en la plataforma de programación Processing.

Su estructura básica consta de dos funciones.

La primera de las dos es la función *setup()*. Esta se usa para inicializar los sistemas conectados a la placa de Arduino, los puertos de comunicación, así como las variables etc. Es ejecutada una única vez por la placa tras el arranque o reseteo de la misma.

Por otro lado, tenemos la función *loop().* Esta función se ejecuta constantemente y es la que tiene el código principal de la aplicación desarrollada. Al ejecutarse como un bucle infinito, nos permite responder a los eventos que se van produciendo en el entorno y que va registrando el sistema.

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Figura 11: Funciones principales de la plataforma Arduino

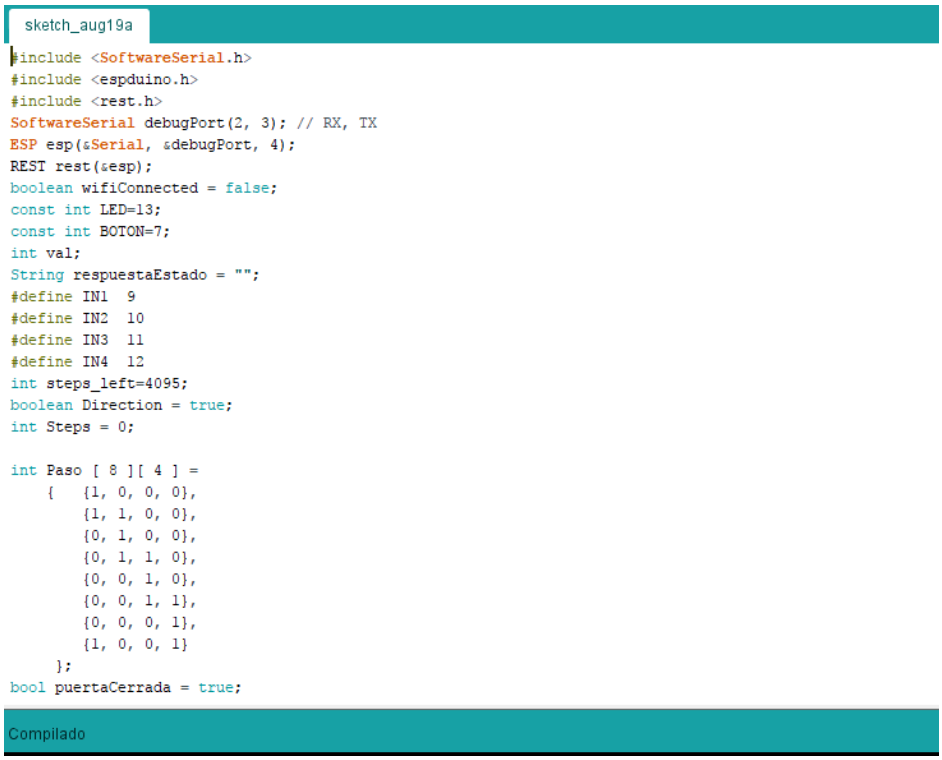
4.2.1.3 Entorno de desarrollo original

El IDE original de Arduino, permite implementar código de programas como sketch, subirlos a la placa Arduino y comunicarse con ellos mediante una consola de comunicación serie.

Está compuesto por diferentes herramientas:

- Editor de texto para implementar código.
- Consola que muestra información del programa en ejecución, y que además, permite enviar información a la placa.
- Barra de herramientas con acceso a funciones comunes.
- Barra superior que incluye todas las funciones disponibles en el entorno.

Además, el entorno integra de forma simplificada, todas las opciones para que el desarrollador pueda implementar, subir y ejecutar un programa, a la placa de Arduino en un tiempo reducido.



```
sketch_aug19a
#include <SoftwareSerial.h>
#include <espduino.h>
#include <rest.h>
SoftwareSerial debugPort(2, 3); // RX, TX
ESP esp(&Serial, &debugPort, 4);
REST rest(&esp);
boolean wifiConnected = false;
const int LED=13;
const int BOTON=7;
int val;
String respuestaEstado = "";
#define IN1  9
#define IN2  10
#define IN3  11
#define IN4  12
int steps_left=4095;
boolean Direction = true;
int Steps = 0;

int Paso [ 8 ][ 4 ] =
    {   {1, 0, 0, 0},
        {1, 1, 0, 0},
        {0, 1, 0, 0},
        {0, 1, 1, 0},
        {0, 0, 1, 0},
        {0, 0, 1, 1},
        {0, 0, 0, 1},
        {1, 0, 0, 1}
    };
bool puertaCerrada = true;

Compilado
```

Figura 12: Entorno de desarrollo(IDE) de Arduino

4.3 Herramientas de programación del servidor

Para que el sistema funcione correctamente, es necesario desarrollar un servidor intermedio que simplifique la comunicación con el servidor de Latch, salvando todos los problemas de implementación relacionados con el certificado que necesita el servidor de Latch, para reconocer y responder correctamente a las peticiones que llegan desde el sistema.

Debido a la necesidad del servidor intermedio, se ha implementado un servidor en PHP, sobre APACHE, en un sistema UBUNTU 16.04 LTS.

4.3.1 Sistema operativo Ubuntu 16.04 LTS

Ubuntu es un sistema operativo basado en Linux, que a su vez, está basado en Unix. Este se distribuye como software libre, está orientado al usuario común, dado que proporciona una interfaz enfocada en la facilidad de uso. Cada vez tiende más a ser utilizado como servidor, ya que con unos pocos comandos se configura un servidor totalmente operativo.

4.3.2 APACHE 2.4

Apache es un servidor web HTTP de código abierto que implementa el protocolo HTTP en su versión 1.1 que se ha utilizado en el presente proyecto como servidor intermedio[9].

Este recibe las peticiones desde el módulo ESP8266, funcionando como esclavo de la placa Arduino, y ejecuta una serie de scripts desarrollados en PHP que consultan el estado de la cuenta en los servidores de Latch.

Para instalar apache sobre Ubuntu se han seguido los siguientes pasos[9]:

En primer lugar instalamos apache2 en el sistema. Para ello ejecutamos en el terminal:

sudo apt-get install apache2

Tras ello, colocamos los scripts que queremos que se ejecuten en la ruta donde se ha instalado el servidor (*/var/www/html*) y les damos permisos de ejecución.

Para completar el correcto funcionamiento del servidor, actualizamos el archivo *index.php* para que ejecute los scripts deseados cuando recibe una petición GET con los parámetros necesarios, y reiniciamos el servicio para que quede totalmente operativo:

sudo /etc/init.d/apache2 restart

4.3.3 PHP

PHP es un lenguaje de programación de alto nivel comúnmente utilizado en el lado del servidor para el desarrollo web.

Mediante PHP se ha implementado la página *index.php* del servidor que ejecuta los scripts de comprobación del estado de la cuenta, y responde a la petición mediante un archivo JSON que es tratado en la placa Arduino.

4.4 API de LATCH

El proyecto trabaja con un servidor propiedad de Telefónica donde se encuentra el estado del Latch que tenemos configurado en nuestra app móvil.

Para poder acceder a estos servidores y consultar el estado de nuestro Latch, existe definida una API REST a la cual se realizan las llamadas para recibir el estado.

Todas las solicitudes a la API de Latch deben estar identificadas y firmadas, de forma que para que todo funcione correctamente, cada solicitud HTTP debe ir acompañada de los encabezados *Authorization* y *Date.*

Una vez configuradas las cabeceras a enviar a la API, podemos realizar diferentes acciones. Entre estas, podemos encontrar las básicas:

- Parear cuenta
- Comprobar Estado
- Desparear Cuenta
- Modificar Estado

Para realizar una de las opciones anteriores, solo tendremos que indicar el *path* al que realizar la petición.

5. Tecnologías empleadas

En este apartado se describen las tecnologías empleadas para el desarrollo del proyecto y su correcto funcionamiento.

5.1 Aplicación Latch

Latch tiene su propia aplicación desarrollada por Telefónica, que podemos encontrar disponible en las tiendas de aplicaciones de los dos principales sistemas operativos móviles (Google Play e App Store).

Figura 13: Aplicación Latch

Está compuesta por un panel de configuración donde se introduce la cuenta que se va a parear con el servicio, un slider principal desde el cual bloquear todos los servicios pareados, y el detalle de los servicios, desde donde podemos activar el bloqueo de alguno de los servicios de forma individual.

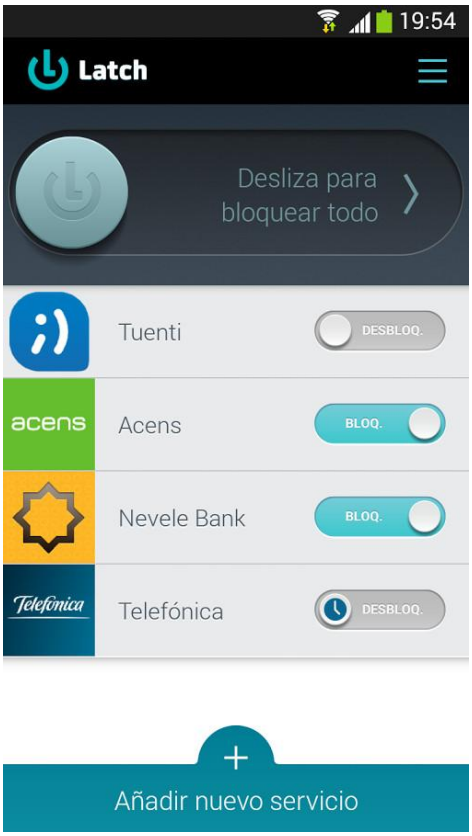


Figura 14: Interfaz aplicación Latch

5.2 Postman

Postman surgió como una extensión para Google Chrome. No obstante, a día de hoy, dispone de aplicaciones nativas para los principales sistemas operativos.

Tiene una versión gratuita que permite la realización de peticiones personalizadas a distintos servidores, facilitando la labor de pruebas y depuración cuando se está desarrollando un sistema en el que interviene un servidor HTTP al que realizar peticiones.[12]

Dispone de una interfaz donde se puede, entre otras cosas:

1. Observar el historial de peticiones realizadas.
2. Seleccionar el método a emplear en la petición (GET, POST, PUT, DELETE, HEAD...).
3. Incluir cabeceras personalizadas.
4. Observar la respuesta recibida.

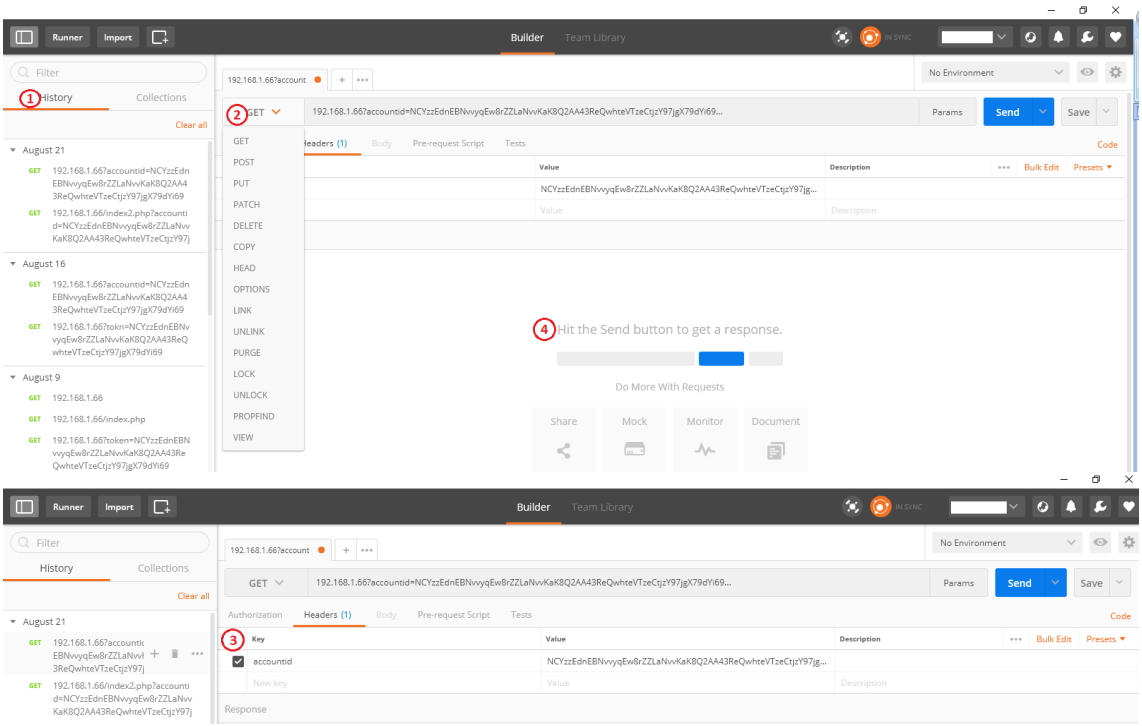


Figura 15: Interfaz Postman

5.3 Wireshark

Wireshark es un analizador de protocolos creado para comprobar el tráfico de una red. Es similar a la herramienta *tcpdump*, sin embargo, este añade una interfaz gráfica que hace que sea más amigable para el usuario.

En el presente proyecto se han realizado capturas espontaneas con el fin de comprobar el correcto uso de las funciones disponibles, tanto en Arduino como en PHP, para el proceso de petición - respuesta en el sistema.

5.4 SublimeText

SublimeText es un IDE que se puede descargar de forma gratuita desde la página web oficial del mismo. Aunque no es un software libre o de código abierto, se puede utilizar la versión de evaluación, ya que no tiene fecha de caducidad.

Se ha utilizado para el desarrollo del código del servidor en PHP.

Figura 16: SublimeText IDE

6. Implementación

En este apartado, se van a describir los pasos seguidos para el desarrollo del código que permitirá que el sistema final funcione conforme a los objetivos definidos.

A continuación, subdividido en dos apartados, se podrán observar los detalles de la implementación, tanto del servidor, como de la placa Arduino.

6.1 Implementación del servidor

Vamos a observar, de una forma mucho más detallada, la implementación del servidor intermedio, que recibe las peticiones desde la placa de Arduino, se comunica con el servidor de Latch, recibe la respuesta del servidor de Latch[2] y responde al sistema Arduino.

6.1.1 Instalación de Sistema Operativo

Con el objetivo de reducir costes y tiempos, se ha optado por instalar el sistema operativo (de ahora en adelante SO), sobre una máquina virtual alojada sobre VirtualBox.

En primer lugar, se ha obtenido la imagen del SO en formato ISO de la página web principal. A continuación, teniendo en cuenta la función del servidor, se ha configurado la máquina virtual, asignándole las siguientes características:

Memoria RAM	1024 MB
Almacenamiento (HDD)	80GB
Tipo de conexión	Adaptador Puente

Tabla 2: Características Servidor Intermedio

Tras ello, se ha instalado el SO y se ha configurado para dejarlo accesible a través de la LAN.

6.1.2 Configuración de red

Para un correcto funcionamiento del sistema, es necesario que el servidor web intermedio, esté configurado con una IP fija dentro de la red. Es a esta ip, a la

que el módulo ESP8266 enviará la petición GET para que el servidor compruebe el estado del Latch.

Para ello, debemos configurar la red del servidor de la siguiente manera:

Abrimos la ventana de configuración de redes:

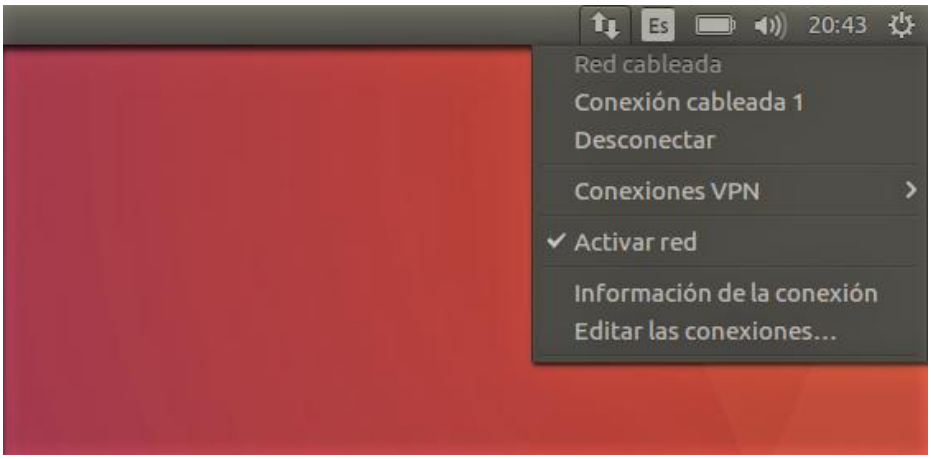


Figura 17: Editar conexiones en Ubuntu

Tras ello, editamos los detalles de conexión de la red actual en la pestaña IPV4:

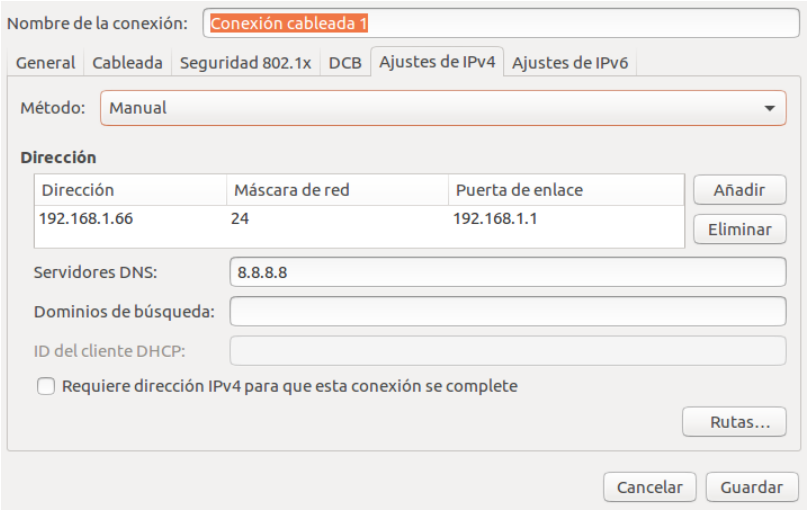
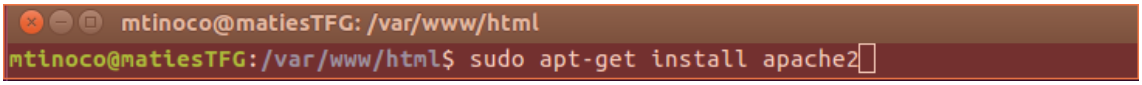


Figura 18: Configurar conexión

6.1.3 Instalación de Apache2

Con el objetivo de que el servidor intermedio, trabaje realmente como un servidor web, se ha optado por instalar Apache sobre el SO Ubuntu. Para ello, se han instalado el paquete *apache2* mediante la ejecución del comando:



6.1.4 Instalación de PHP

Dado que el servidor está programado en PHP, es necesario instalar en el sistema, los requisitos necesarios para que se ejecute el código de la forma deseada. Para ello, ha sido necesario instalar en el SO Ubuntu, que hace las funciones de servidor intermedio, los paquetes necesarios para que gestione, de forma correcta, el código desarrollado en PHP.

Para ello, se ha instalado el modulo que habilita la ejecución de código PHP en el servidor Ubuntu mediante el comando:

```
mtinoco@matiesTFG: /var/www/html
mtinoco@matiesTFG:/var/www/html$ sudo apt-get install libapache2-mod-php
```

6.1.5 Creación de Scripts

Siguiendo el API de Latch, definido en la página web del proyecto, se han desarrollado dos scripts que sirven para, por un lado, parear una cuenta de usuario en el sistema de Latch, y por otro, consultar el estado del mismo definido en la app móvil.

6.1.5.1 Script de pareado de cuenta

En el sistema, el primer paso consiste en parear la cuenta a la placa de Arduino para obtener un *accountid*, el cual será utilizado para comprobar el estado del Latch.

Para ello, existe un API que debemos utilizar para la función de pareado:

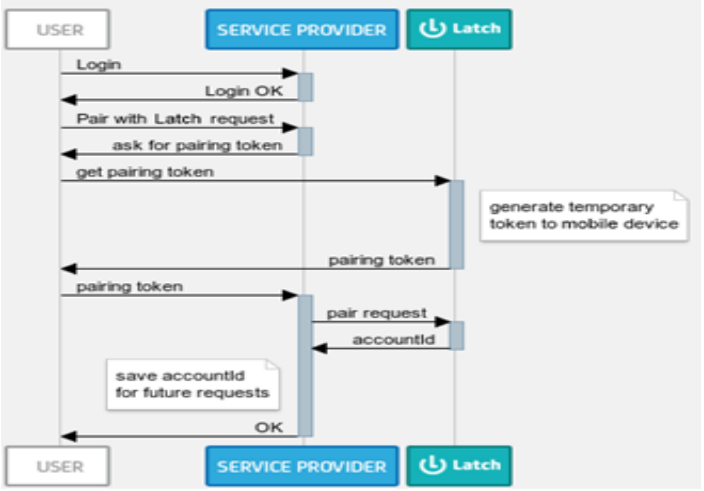


Figura 19: API pareado Latch

Como medida de seguridad, los servidores de Latch implementan un sistema de autenticación. Este sistema requiere de una cabecera *Authorization* compuesta por la constante *11PATHS*, un *applicationId* que se obtiene al registrar la aplicación en el sistema oficial de Telefónica y una *requestSignature,* la cual se obtiene de una firma derivada de la URL, los parámetros, los encabezados y la fecha actual, todo ello empleando un *hash* que también obtenemos, como en el caso del *applicationId,* al registrar la aplicación de forma oficial.

Todo este proceso de autenticación, está implementado en el script que ejecuta el servidor intermedio llamado *pair.sh*:

```
ApplicationId="yjY293nZr7i63NUJtH3j"
SecretKey="PQTazKmFiHXYvwMTmVc28d6eHbd8kdavgtjq4eUs"

Server="https://latch.elevenpaths.com"
URL="/api/1.1/pair/$1"

requestSignature+="GET\n"
date=`date -u '+%Y-%m-%d %H:%M:%S'`
requestSignature+="$date\n\n$URL"
signed=`echo -en "$requestSignature" | openssl dgst -sha1 -hmac "$SecretKey" -binary`
b64signed=`echo -n "$signed"|base64`

auth_header="Authorization:11PATHS $ApplicationId $b64signed"
date_header="X-11Paths-Date: $date"
```

Tras ello, se realiza la petición con el comando siguiente:

```
response=`curl -q -s -N --header "$auth_header" --header "$date_header" "$Server$URL"`
```

Con lo que, en la variable *response* tendremos la respuesta de los servidores de Latch, donde vendrá especificado el *AccountId*, el cual utilizará la placa de Arduino para consultar el estado configurado mediante la ejecución del script *status.sh*.

```
#!/bin/bash

if [ -z "$1" ]; then
 echo -e "\nUsage: $0 \n"
 exit 0
fi

ApplicationId="yjY293nZr7i63NUJtH3j"
SecretKey="PQTazKmFiHXYvwMTmVc28d6eHbd8kdavgtjq4eUs"

Server="https://latch.elevenpaths.com"
URL="/api/1.1/pair/$1"

requestSignature+="GET\n"
date=`date -u '+%Y-%m-%d %H:%M:%S'`
requestSignature+="$date\n\n$URL"
signed=`echo -en "$requestSignature" | openssl dgst -sha1 -hmac "$SecretKey" -binary`
b64signed=`echo -n "$signed"|base64`

auth_header="Authorization:11PATHS $ApplicationId $b64signed"
date_header="X-11Paths-Date: $date"

response=`curl -q -s -N --header "$auth_header" --header "$date_header" "$Server$URL"`

echo $response
```

Figura 20: script pair.sh

6.1.5.1 Script de consulta de estado de Latch

Una vez obtenido el *AccountId* mediante el proceso de pareado, es posible consultar el estado del Latch configurado.

Para ello, se ha implementado un script llamado *status.sh*, el cual utiliza la misma API que el script de pareado:

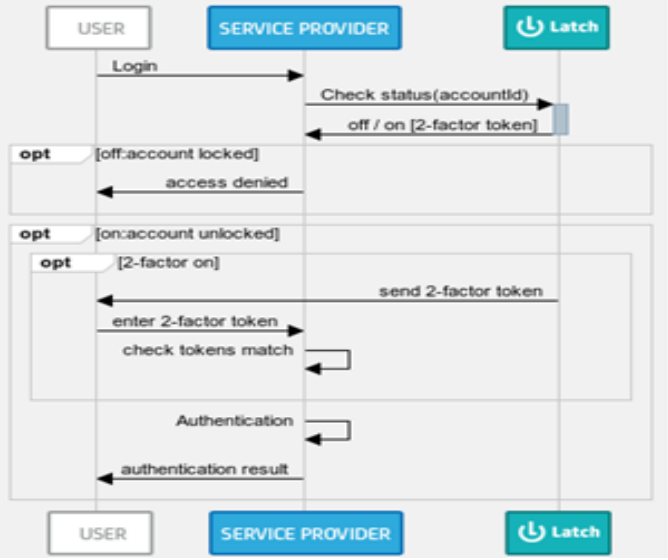


Figura 21: API comprobación de estado Latch

Utilizando el mismo proceso que el script de pareado, únicamente tenemos que cambiar la ruta de la petición y utilizar el *AccountId*:

```
Server="https://latch.elevenpaths.com"
URL="/api/1.1/status/$1"

requestSignature+="GET\n"
date=`date -u '+%Y-%m-%d %H:%M:%S'`
requestSignature+="$date\n\n$URL"
signed=`echo -en "$requestSignature" | openssl dgst -sha1 -hmac "$SecretKey" -binary`
b64signed=`echo -n "$signed"|base64`

auth_header="Authorization:11PATHS $ApplicationId $b64signed"
date_header="X-11Paths-Date: $date"
```

Podemos observar, como en este caso, la variable *URL* ya no apunta a la ruta */api/1.1/pair/,* sino que en este caso, apunta a la ruta */api/1.1/status/.*

Una vez realizada la petición, del mismo modo que en el script de pareado, tendremos en la variable *response* la respuesta de los servidores de Latch, en este caso, indicando, en vez del *accountId,* el estado de la cuenta:

```
response=`curl -q -s -N --header "$auth_header" --header "$date_header" "$Server$URL"`
```

De esta forma, el script para comprobar el estado configurado, queda implementado de la siguiente manera:

```
#!/bin/bash

if [ -z "$1" ]; then
 echo -e "\nUsage: $0 \n"
 exit 0
fi

ApplicationId="yjY293nZr7i63NUJtH3j"
SecretKey="PQTazKmFiHXYvwMTmVc28d6eHbd8kdavgtjq4eUs"

Server="https://latch.elevenpaths.com"
URL="/api/1.1/status/$1"

requestSignature+="GET\n"
date=`date -u '+%Y-%m-%d %H:%M:%S'`
requestSignature+="$date\n\n$URL"
signed=`echo -en "$requestSignature" | openssl dgst -sha1 -hmac "$SecretKey" -binary`
b64signed=`echo -n "$signed"|base64`

auth_header="Authorization:11PATHS $ApplicationId $b64signed"
date_header="X-11Paths-Date: $date"

response=`curl -q -s -N --header "$auth_header" --header "$date_header" "$Server$URL"`

echo $response
```

Figura 22: script status.sh

Para comprobar el correcto funcionamiento, se ha realizado una captura del dialogo entre el ESP8266 y el servidor intermedio mediante la herramienta Wireshark[11], donde podemos observar como el servidor intermedio responde en formato JSON con la respuesta obtenida de los servidores de Latch:

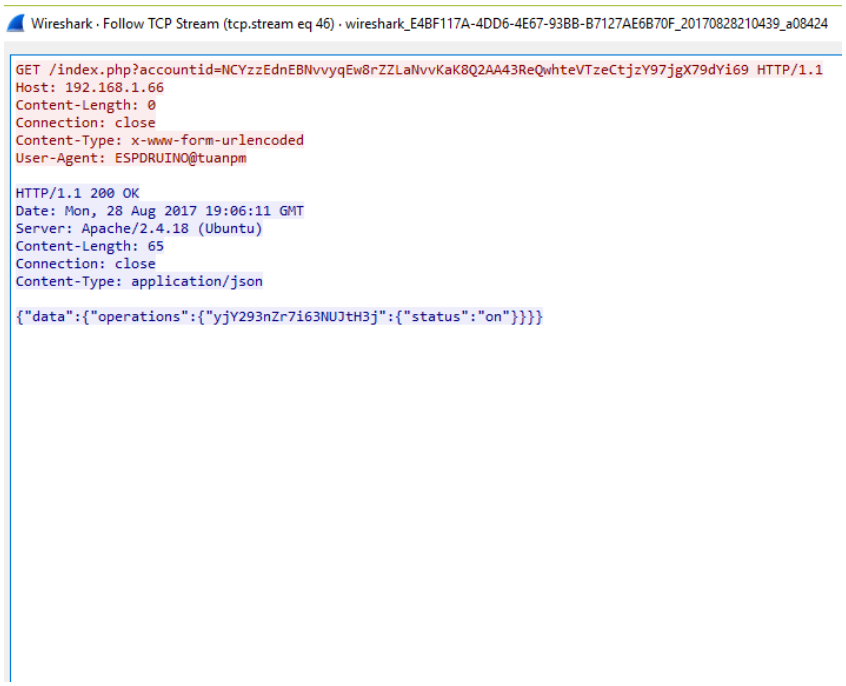


Figura 23: Comunicación entre ESP8266 y Servidor intermedio

6.1.6 Creación de página principal

Un HTTP, normalmente, establece por defecto que, si no se especifica la ruta a la que se desea enviar la petición, esta irá destinada al archivo *index.html* o en su defecto al archivo *index.php* disponible en la ruta principal del servidor (en nuestro caso (*/var/www/html/*).

Por ello, se ha creado un archivo *index.php* que recoge los parámetros pasados en la petición GET y ejecuta como CGI los scripts que comprueban el estado del Latch o realizan el pareado de la cuenta, dependiendo del parámetro que reciba en la petición.

```php
<?php
try{
    $url = "$_SERVER[REQUEST_URI]";
    $url = filter_var($url,FILTER_SANITIZE_URL);
    $parts = parse_url($url);
    parse_str($parts['query'], $query);

    if(isset($query['accountid'])){

        $accountid = $query['accountid'];
        $comando = './status.sh ' .$accountid;
        $resultado = shell_exec($comando);
        $data = $resultado;
        header('Content-Type: application/json');
        echo $data;

    }elseif(isset($query['token'])){

        $token = $query['token'];
        $comando = './pair.sh ' .$token;
        $resultado = shell_exec($comando);
        $data = $resultado;
        header('Content-Type: application/json');
        echo $data;

    }else{

        echo "Ni ACCOUNTID ni TOKEN definidos en la petición";

    }
} catch (Exception $e){

    echo "Excepcion capturada: " , $e->getMessage(), "\n";

}
?>
```

Figura 24: código index.php

6.2 Implementación del sistema Arduino

En este subapartado se va a detallar el diseño del sistema Arduino, junto con las conexiones de todos sus módulos y la programación del sketch que va a cargarse y ejecutarse continuamente en la placa a la espera de eventos a los que responder.

6.2.1 Conexionado de los componentes

Siguiendo los requisitos de cada uno de los componentes, se ha construido el circuito necesario para el correcto funcionamiento de los mismos.

En este prototipo, se han utilizado los siguientes componentes:

- Placa Arduino UNO R3.
- Motor Unipolar 28BYJ-48.
- Conversor USB-TTL.
- Módulo WiFi ESP8266.
- Fuente de alimentación suplementaria.
- Driver ULN2003A.
- Pulsador electrónico.
- Diodo LED.
- Resistencias.
- Cables Dupont.
- Placa de prototipado.

En primer lugar, se ha conectado el módulo WiFi ESP8266 a la placa Arduino y a la fuente de alimentación suplementaria. De esta forma, obtiene los mA. necesarios para el correcto arranque, inicialización y dotado de conexión inalámbrica a la placa de Arduino.

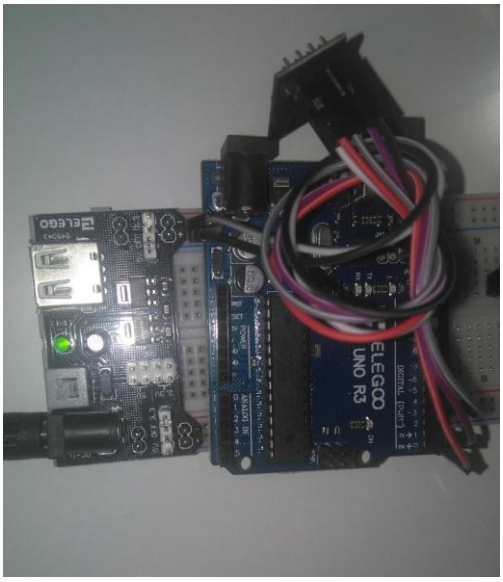


Figura 25: Arduino + ESP8266

Por otro lado, se ha conectado el conversor USB-TTL a la placa Arduino para observar en el monitor serial los mensajes de control implementados en el sketch.

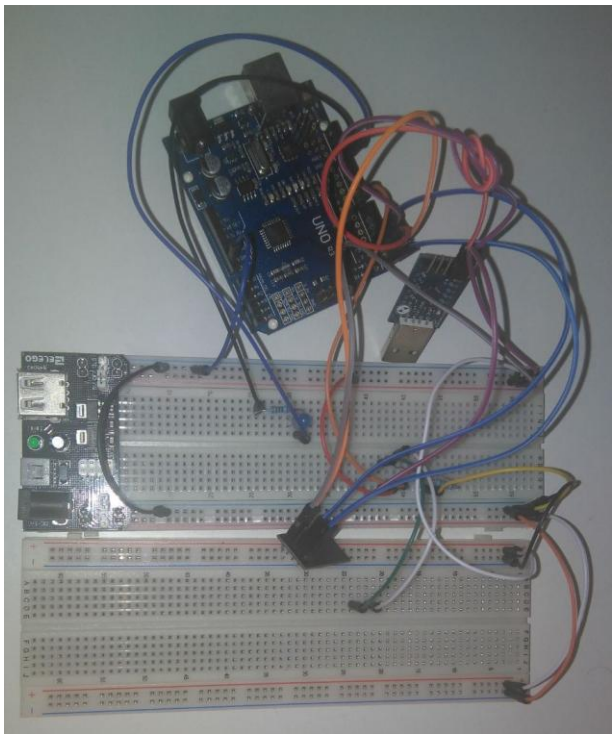


Figura 26: Arduino + ESP8266 + USB-TTL

Además, se ha conectado el motor unipolar a la placa de Arduino a través del driver ULN2003A que nos evita el uso de resistencias para la utilización del mismo.

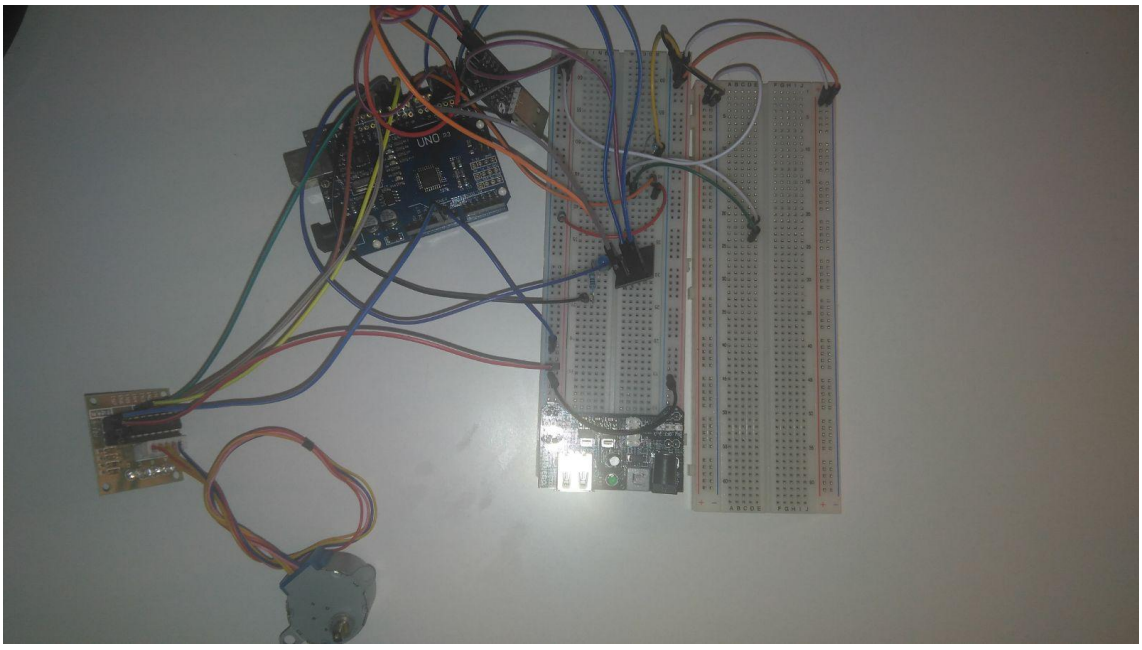


Figura 27: Arduino + ESP8266 + 28BYJ-48 + USB-TTL

Por último, para el accionamiento del sistema, se ha utilizado un pulsador que hace a los efectos, del botón de apertura de la puerta. A este se le ha instalado

un LED que nos indica cuando el pulsador es accionado y el sistema se pone en marcha.

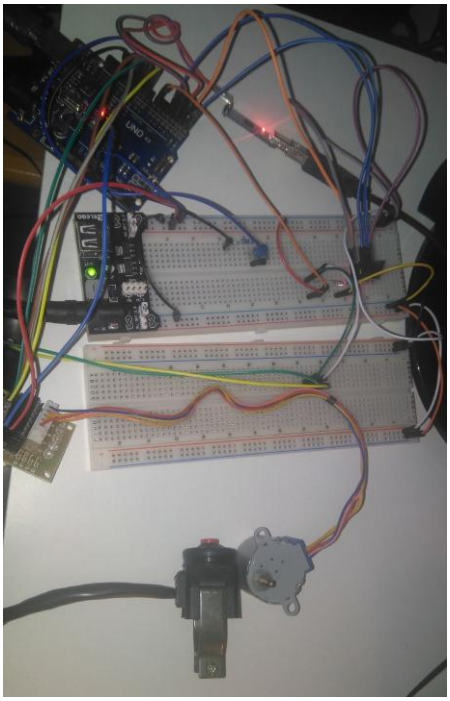


Figura 28: Sistema completo a la espera

6.2.2 Creación del sketch

La parte fundamental del proyecto es la implementación del sketch que ejecuta la placa Arduino para gestionar los eventos.

Este sketch, cuenta con diversas funciones que se van a detallar a continuación.

En primer lugar, importamos las librerías necesarias y definimos las variables que vamos a utilizar en la implementación:

```
#include <SoftwareSerial.h>
#include <espduino.h>
#include <rest.h>
SoftwareSerial debugPort(2, 3); // RX, TX
ESP esp(&Serial, &debugPort, 4);
REST rest(&esp);
boolean wifiConnected = false;
const int LED=13;
const int BOTON=7;
int val;
String respuestaEstado = "";
#define IN1  9
#define IN2  10
#define IN3  11
#define IN4  12
int steps_left=4095;
boolean Direction = true;
int Steps = 0;

int Paso [ 8 ][ 4 ] =
    {   {1, 0, 0, 0},
        {1, 1, 0, 0},
        {0, 1, 0, 0},
        {0, 1, 1, 0},
        {0, 0, 1, 0},
        {0, 0, 1, 1},
        {0, 0, 0, 1},
        {1, 0, 0, 1}
     };
bool puertaCerrada = true;
```

Figura 29: Variables sketch TFG.ino

Aquí podemos ver que definimos variables para el funcionamiento del ESP8266, del pulsador y el LED que indica su accionamiento y de las variables relacionadas con el funcionamiento del motor unipolar.

Tras ello, utilizando parte del código disponible en la librería ESPduino, definimos la función *WifiCb()* que funcionará como *callback* para obtener el resultado de la conexión del ESP8266 a la red WiFi[15]:

```
void wifiCb(void* response){
  uint32_t status;
  RESPONSE res(response);
  if(res.getArgc() == 1) {
    res.popArgs((uint8_t*)&status, 4);
    if(status == STATION_GOT_IP) {
      debugPort.println("WIFI CONNECTED");

      wifiConnected = true;
    } else {
      wifiConnected = false;
    }
  }
}
```

Figura 30: Función WifiCb()

Una vez definida esta función, se han utilizado diversas funciones para controlar el movimiento del motor unipolar[6][7][8]:

```
void giraIzqMotor(){     // ABRIR
  while(steps_left>0)
        {
          stepper() ;    // Avanza un paso
          steps_left-- ;  // Un paso menos
          delay (1) ;
        }
      delay(300);
      Direction=!Direction;
      steps_left=4095;
      puertaCerrada = false;
}

void giraDerechaMotor(){  //CERRAR
  while(steps_left>0)
        {
          stepper() ;    // Avanza un paso
          steps_left-- ;  // Un paso menos
          delay (1) ;
        }
      delay(300);
      Direction=!Direction;
      steps_left=4095;
      puertaCerrada = true;
}
```

```
void stepper()            //Avanza un paso
{
  digitalWrite( IN1, Paso[Steps][ 0] );
  digitalWrite( IN2, Paso[Steps][ 1] );
  digitalWrite( IN3, Paso[Steps][ 2] );
  digitalWrite( IN4, Paso[Steps][ 3] );

  SetDirection();
}


void SetDirection(){
    if(Direction)
        Steps++;
    else
        Steps--;

    Steps = ( Steps + 7 ) % 7 ;
}
```

Figura 31: Funciones de control del motor unipolar

Ya tenemos definidas las funciones de control del sistema, por lo que es el momento de implementar las funciones de *setup()* y de *loop()*.

En la primera de estas inicializamos los componentes que va a utilizar nuestro sistema. En primer lugar, definimos el tipo de utilidad que van a tener los pines a los que están conectados los módulos:

```
void setup() {
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);
  pinMode(LED,OUTPUT);
  pinMode(BOTON,INPUT);
```

Figura 32: Inicialización de los pines utilizados

Tras ello, conectamos el módulo WiFi a la red[5]:

```
  debugPort.begin(19200);
  esp.enable();
  delay(500);
  esp.reset();
  delay(500);
  while(!esp.ready());

  debugPort.println("ARDUINO: setup rest client");
  if(!rest.begin("192.168.1.66")) {
    debugPort.println("ARDUINO: failed to setup rest client");
    while(1);
  }
  debugPort.println("ARDUINO: setup wifi");
  esp.wifiCb.attach(&wifiCb);
  esp.wifiConnect("CALOCA","2301L0412C");
  debugPort.println("ARDUINO: system started");
}
```

Figura 33: Conexionado del modulo WiFi a la red

Una vez inicializados los componentes, es el momento de implementar la función *loop().* Esta ejecutará de forma cíclica, la función *checkstatus()* encargada de registrar los eventos de pulsación del botón de accionamiento y realizar la petición al servidor intermedio[4]:

```
void checkStatus(){
  val=digitalRead(BOTON);
  char response[266];
  esp.process();
  if(wifiConnected && val==HIGH) {
    digitalWrite(LED,HIGH);
    rest.request("/index.php?accountid=NCYzzEdnEBNvvyqEw8rZZLaNvvKaK8Q2AA43ReQwhteVTzeCtjzY97jgX79dYi69","GET","");
    if(rest.getResponse(response, 266) == HTTP_STATUS_OK){
      //motor();
      debugPort.println("RESPONSE: ");
      debugPort.println(response);
      respuestaEstado= response;
      Serial.println("\n\n");
      Serial.println(respuestaEstado);
  }
  val = LOW;
  digitalWrite(LED,LOW);
  delay(1000);
  }
}
```

Figura 34: Función checkstatus()

Esta función sobrescribirá el valor de la variable *respuestaEstado* en caso de obtener una respuesta del servidor intermedio. Tras ello, la función *loop()* ejecutará la sección de código que se encarga de parsear la respuesta y accionar el motor unipolar o no:

```
if(respuestaEstado != ""){  //TRATAMIENTO DE RESPUESTA DEL LATCH
  char myString[266];
  for(int i = 0; i < 265; i++){
    myString[i] = respuestaEstado.charAt(i);
  }
  Serial.println("myString => ");
  Serial.println(myString);

  if(respuestaEstado.indexOf("\"on\"") != -1){
    estadoLatch = "abrir";
  }else{
    if(respuestaEstado.indexOf("\"off\"") != -1){
      estadoLatch = "cerrado";
    }else{
      estadoLatch = "NO FUNCIONA";
    }
  }
    Serial.println("\n\n");
    Serial.println("ESTADO LATCH =>  ");
    Serial.println(estadoLatch);
    if(estadoLatch == "abrir"){
      if(puertaCerrada == true){
        giraIzqMotor();
      }
    }else{
      if(puertaCerrada == false){
        giraDerechaMotor();
      }
    }
respuestaEstado = "";
}
```

Figura 35: Comprobación de variable respuestaEstado y accionamiento del motor

6.2.3 Arranque del sistema

Para el arranque del sistema, es necesario que el ESP8266 se conecte a la red de forma correcta, por lo que, como se ha definido anteriormente en el apartado donde se detalla la implementación del sketch, la red WiFi a la que se va a conectar, debe estar operativa en el momento del arranque del sistema.

Tras ello, el sistema necesita de aproximadamente unos 5 segundos para inicializarse correctamente, que es el tiempo que necesita el ESP8266 para conectarse a la red WiFi.

Tras ello, ya está totalmente operativo y queda a la espera que se produzca el evento de solicitud de apertura de la puerta para ponerse en marcha.

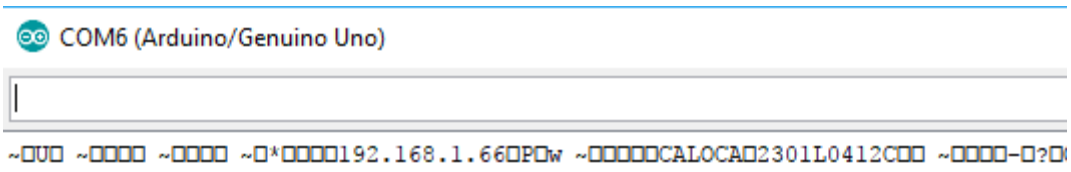


Figura 36: Sistema a la espera de recepción de evento

6.2.4 Comprobación del estado de las peticiones y respuestas

Para comprobar el correcto uso de las peticiones HTTP y que la respuesta recibida por los servidores es correcta, se ha utilizado Wireshark para comprobar la petición emitida por el ESP8266 con destino al servidor intermedio, como por el servidor intermedio con destino a los servidores de Latch.

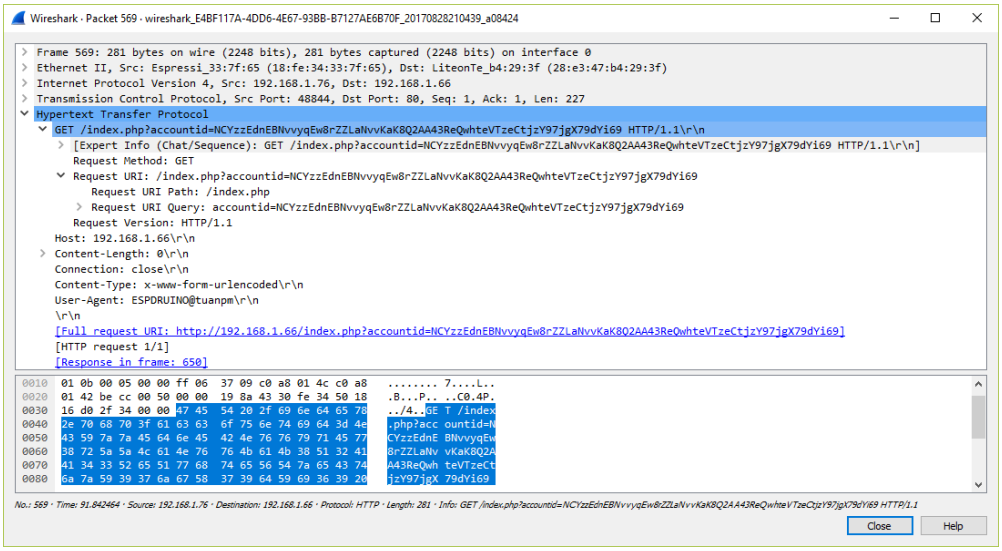


Figura 37: Petición ESP8266 a Servidor Intermedio

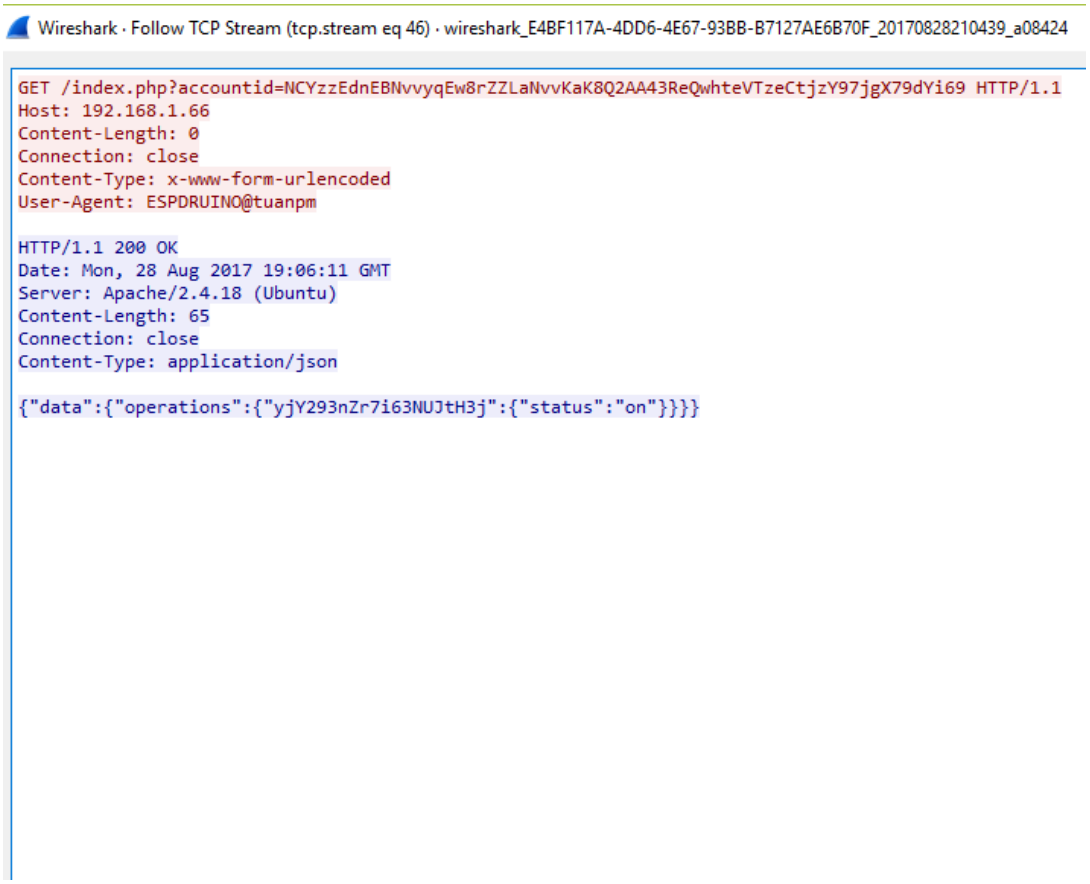


Figura 38: Dialogo TCP entre ESP8266 y Servidor Intermedio

7. Pruebas

En primer lugar, indicar que el correcto funcionamiento de los componentes, se ha comprobado de forma individual durante la fase de desarrollo, con lo que sabemos que cada componente, realiza su función cumpliendo con los requisitos establecidos en la fase inicial del proyecto.

Tras ello, con todo el sistema con conectado, los servicios implementados y los componentes del sistema correctamente inicializados y en funcionamiento, se han realizado una serie de pruebas para comprobar el funcionamiento global de la solución.

Durante las pruebas, se han ido observando en el monitor serie que incorpora el IDE de Arduino, los resultados que se iban obteniendo.

7.1 Pruebas del servidor

Para comprobar el correcto funcionamiento del servidor, se ha utilizado la aplicación Postman.[13]

Con esta aplicación, se han simulado las peticiones que va a realizar el ESP8266, con destino a nuestro servidor, y se ha observado la respuesta que devuelve, comprobando así, que nuestro servidor responde al ESP8266 los datos que nosotros deseamos.

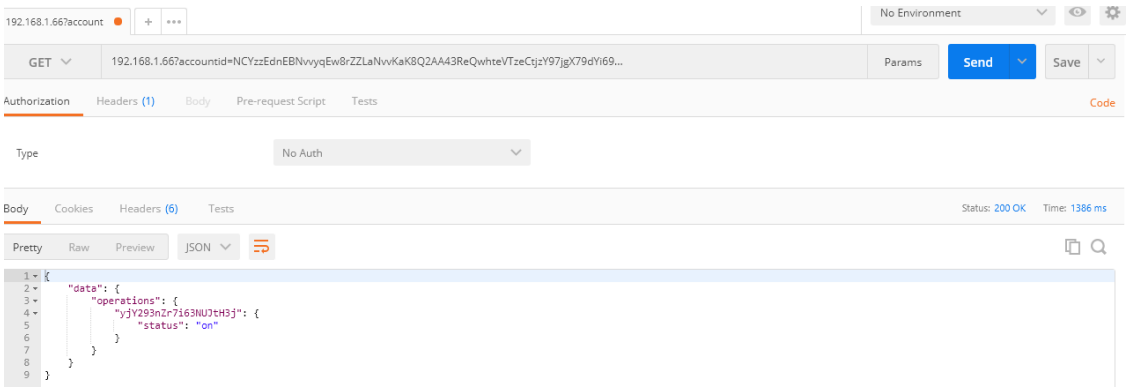


Figura 39: Recepción respuesta servidor con Postman

7.2 Pruebas del sistema

En primer lugar, se ha realizado un intento de acceso con Latch desbloqueado, comprobando como, efectivamente, se autoriza la apertura de la puerta:

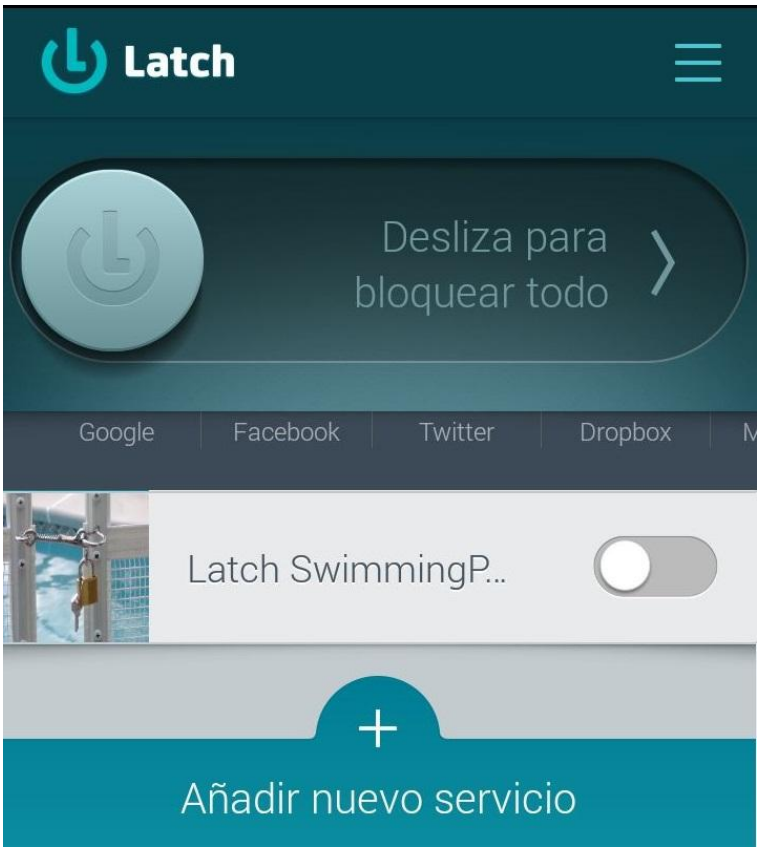


Figura 40: Interfaz móvil Latch desbloqueado

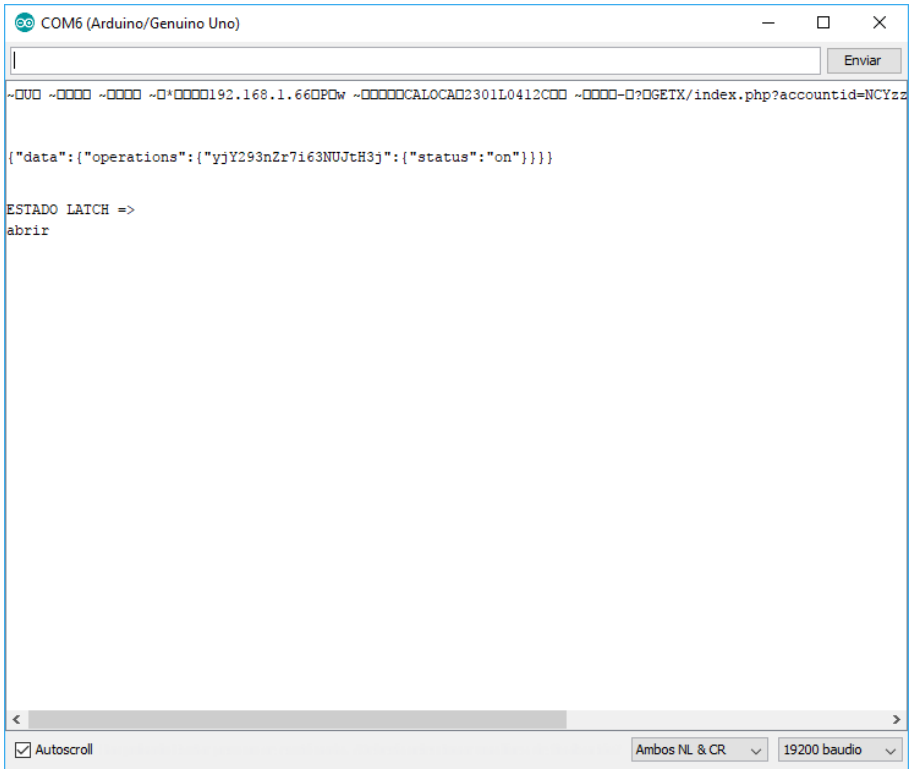


Figura 41: Autorización de apertura

Por contra, cuando Latch se encuentra bloqueado, se ha comprobado como el sistema rechaza la apertura de la cerradura, impidiendo el acceso:

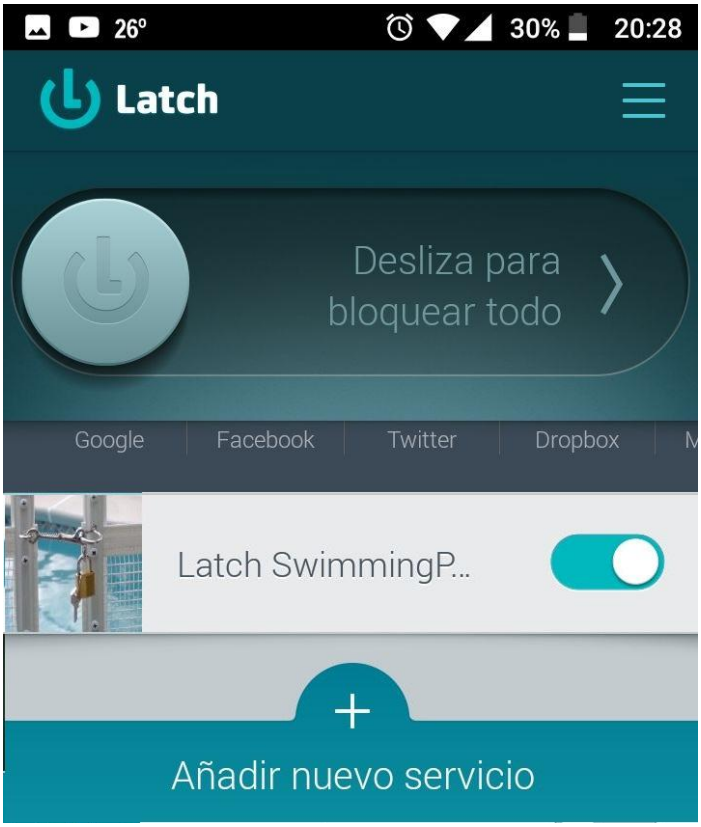


Figura 42: Interfaz móvil Latch bloqueado

Podemos observar como con Latch bloqueado, el monitor serie de Arduino nos muestra que se debe mantener en estado "cerrado" el sistema, impidiendo el acceso:

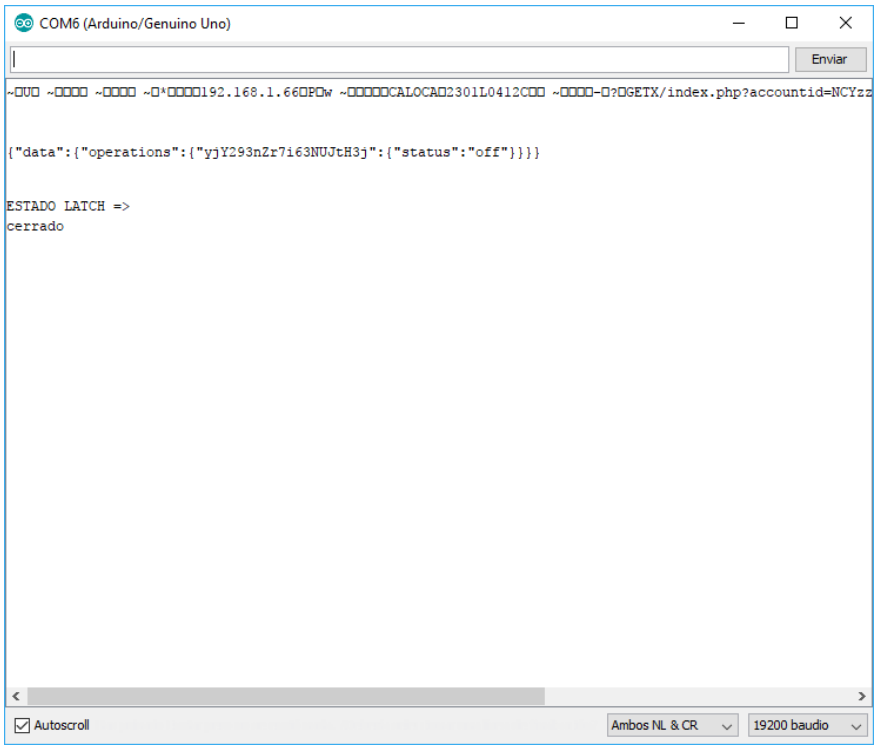


Figura 43: Denegación de apertura

Además, conforme se solicita la apertura estando bloqueado, se recibe la notificación de acceso en el dispositivo móvil donde se encuentra instalada la aplicación de Latch:

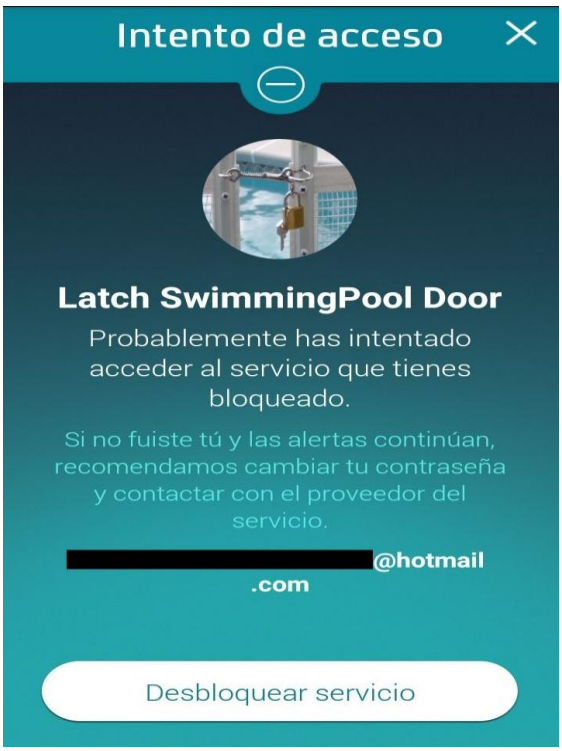


Figura 44: Alerta móvil de intento de acceso

8. Conclusiones y trabajo futuro

Actualmente, es posible implementar sistemas de una forma económica y sencilla. Arduino, sigue siendo una de las plataformas más extendidas para la implementación de este tipo de sistemas.

No obstante, ante la gran variedad de componentes disponibles en el mercado, se hace necesario un análisis de requisitos para elegir los componentes adecuados, que cumplan los requisitos, tanto de costes, como de características necesarias para el correcto desarrollo del proyecto.

Una vez realizados los análisis pertinentes, se ha optado por utilizar los componentes antes mencionados, para el desarrollo del sistema, comprobando la total viabilidad del proyecto dentro de un presupuesto reducido.

Teniendo siempre presente el objetivo inicial, que motivó al desarrollo del proyecto, se ha demostrado que con un sistema Open Source como Arduino, es posible desarrollar un sistema totalmente funcional.

Debido a la creciente uso del mundo IoT (Internet of Things), se debe estudiar aspectos relacionados con la seguridad de los mismos, tales como Integridad, Confidencialidad y Disponibilidad.

Con este problema como base, algunos de los posibles trabajos futuros, que darían continuidad al proyecto, podrían ser:

- Comunicación cifrada entre Arduino y Servidor intermedio.
- Ahorro del servidor intermedio mediante el uso de un microcontrolador que permita la utilización de certificados mayores a 1024 bits[3].
- Recopilación de logs en el servidor intermedio.
- Creación de base de datos para la recolección de estadísticas y desarrollo de front-end que permita analizar el uso del sistema.
- Integración del sistema en distintos ámbitos como puede ser, entre muchos otros:
    - Apertura de cajas fuertes[8].
    - Autorización de apertura de garajes.
    - Autorización de uso de sistemas electrónicos dentro de un domicilio.

9. Bibliografía

[1] Latch. El interruptor de seguridad para tu vida digital: **https://latch.elevenpaths.com**

[2] Latch. Como proteger las identidades digitales: **http://www.elladodelmal.com/2013/12/como-proteger-las-identidades-digitales.html**

[3] Latch y el Internet de las Cosas. Integración con Arduino: **http://blog.elevenpaths.com/2015/10/latch-y-el-internet-de-las-cosas.html**

[4] Sending GET request from ESP8266+Arduino: **https://forum.arduino.cc/index.php?topic=350678.0**

[5] ESP8266: HTTP GET Requests: **https://techtutorialsx.com/2016/07/17/esp8266-http-get-requests/**

[6] Motores paso a paso: 28BYJ-48: **http://www.prometec.net/motor-28byj-48/**

[7] Motor paso a paso 28BYJ-48 con Arduino: **https://programarfacil.com/blog/motor-paso-a-paso/**

[8] Emulador de cerradura de caja fuerte con motor 28BYJ-48: **https://puntoflotante.net/CONTROL-MOTOR-DE-PASOS-28BYJ-48-BOLT-18F2550.htm**

[9] Instalar Apache 2 en Ubuntu: **http://www.linuxhispano.net/2009/11/01/instalar-apache-ubuntu/**

[10] Reducción de la mortalidad en la niñez. Organización mundial de la salud. **http://www.who.int/mediacentre/factsheets/fs178/es/**

[11] Análisis de tráfico con Wireshark: **https://www.incibe.es/extfrontinteco/img/File/intecocert/EstudiosInformes/cert_inf_seguridad_analisis_trafico_wireshark.pdf**

[12] Postman: gestiona y construye tus APIs rápidamente: **https://www.paradigmadigital.com/dev/postman-gestiona-construye-tus-apis-rapidamente/**

[13] Postman. Request Builder: **https://www.getpostman.com/docs/postman/sending_api_requests/requests**

[14] Internet de las cosas. Sistema electrónico de control basado en Arduino: **https://riunet.upv.es/bitstream/handle/10251/55869/MARTINEZ%20-%20Internet%20de%20las%20cosas.%20Sistema%20electr%C3%B3nico%20de%20control%20basado%20en%20Arduino..pdf?sequence=2&isAllowed=y**

[15] Github - tuanmpt/espduino: ESP8266 network client(mqtt,restful) for Arduino: **https://github.com/tuanpmt/espduino**

10. Anexos

10.1 Script pair.sh

```
#!/bin/bash

if [ -z "$1" ]; then
 echo -e "\nUsage: $0 \n"
 exit 0
fi

ApplicationId="yjY293nZr7i63NUJtH3j"
SecretKey="PQTazKmFiHXYvwMTmVc28d6eHbd8kdavgtjq4eUs"

Server="https://latch.elevenpaths.com"
URL="/api/1.1/pair/$1"

requestSignature+="GET\n"
date=`date -u '+%Y-%m-%d %H:%M:%S'`
requestSignature+="$date\n\n$URL"
signed=`echo -en "$requestSignature" | openssl dgst -sha1 -hmac "$SecretKey" -binary`
b64signed=`echo -n "$signed"|base64`

auth_header="Authorization:11PATHS $ApplicationId $b64signed"
date_header="X-11Paths-Date: $date"

response=`curl -q -s -N --header "$auth_header" --header "$date_header" "$Server$URL"`

echo $response
```

10.2 Script status.sh

```
#!/bin/bash

if [ -z "$1" ]; then
 echo -e "\nUsage: $0 \n"
 exit 0
fi

ApplicationId="yjY293nZr7i63NUJtH3j"
SecretKey="PQTazKmFiHXYvwMTmVc28d6eHbd8kdavgtjq4eUs"

Server="https://latch.elevenpaths.com"
URL="/api/1.1/status/$1"

requestSignature+="GET\n"
date=`date -u '+%Y-%m-%d %H:%M:%S'`
requestSignature+="$date\n\n$URL"
signed=`echo -en "$requestSignature" | openssl dgst -sha1 -hmac "$SecretKey" -binary`
b64signed=`echo -n "$signed"|base64`

auth_header="Authorization:11PATHS $ApplicationId $b64signed"
date_header="X-11Paths-Date: $date"

response=`curl -q -s -N --header "$auth_header" --header "$date_header" "$Server$URL"`

echo $response
```

10.3 index.php

```php
<?php
try{
    $url = "$_SERVER[REQUEST_URI]";
    $url = filter_var($url,FILTER_SANITIZE_URL);
    $parts = parse_url($url);
    parse_str($parts['query'], $query);

    if(isset($query['accountid'])){

        $accountid = $query['accountid'];
        $comando = './status.sh ' .$accountid;
        $resultado = shell_exec($comando);
        $data = $resultado;
        header('Content-Type: application/json');
        echo $data;
    }elseif(isset($query['token'])){

        $token = $query['token'];
        $comando = './pair.sh ' .$token;
        $resultado = shell_exec($comando);
        $data = $resultado;
        header('Content-Type: application/json');
        echo $data;
    }else{

        echo "Ni ACCOUNTID ni TOKEN definidos en la petición";
    }
} catch (Exception $e){
}
?>
```

10.4 TFG.ino

```
#include <SoftwareSerial.h>
#include <espduino.h>
#include <rest.h>
SoftwareSerial debugPort(2, 3); // RX, TX
ESP esp(&Serial, &debugPort, 4);
REST rest(&esp);
boolean wifiConnected = false;
const int LED=13;
const int BOTON=7;
int val;
String respuestaEstado = "";
#define IN1  9
#define IN2  10
#define IN3  11
```

```
#define IN4  12
int steps_left=4095;
boolean Direction = true;
int Steps = 0;

int Paso [ 8 ][ 4 ] =
   {  {1, 0, 0, 0},
      {1, 1, 0, 0},
      {0, 1, 0, 0},
      {0, 1, 1, 0},
      {0, 0, 1, 0},
      {0, 0, 1, 1},
      {0, 0, 0, 1},
      {1, 0, 0, 1}
    };
bool puertaCerrada = true;

void wifiCb(void* response){
  uint32_t status;
  RESPONSE res(response);
  if(res.getArgc() == 1) {
    res.popArgs((uint8_t*)&status, 4);
    if(status == STATION_GOT_IP) {
      debugPort.println("WIFI CONNECTED");

      wifiConnected = true;
    } else {
      wifiConnected = false;
    }
  }
}

void checkStatus(){
  val=digitalRead(BOTON);
  char response[266];
  esp.process();
  if(wifiConnected && val==HIGH) {
    digitalWrite(LED,HIGH);
    rest.request("/index.php?accountid=NCYzzEdnEBNvvyqEw8rZZLaNvvKaK8Q2AA43ReQwhteVTzeCtjzY97jgX79dYi69","GET","");
    if(rest.getResponse(response, 266) == HTTP_STATUS_OK){
      //motor();
      debugPort.println("RESPONSE: ");
      debugPort.println(response);
      respuestaEstado= response;
      Serial.println("\n\n");
      Serial.println(respuestaEstado);
  }
  val = LOW;
  digitalWrite(LED,LOW);
  delay(1000);
  }
```

```
66. }
67.
68. void setup() {
69.   pinMode(IN1, OUTPUT);
70.   pinMode(IN2, OUTPUT);
71.   pinMode(IN3, OUTPUT);
72.   pinMode(IN4, OUTPUT);
73.   pinMode(LED,OUTPUT);
74.   pinMode(BOTON,INPUT);
75.   Serial.begin(19200);
76.   debugPort.begin(19200);
77.   esp.enable();
78.   delay(500);
79.   esp.reset();
80.   delay(500);
81.   while(!esp.ready());
82.   debugPort.println("ARDUINO: setup rest client");
83.   if(!rest.begin("192.168.1.66")) {
84.     debugPort.println("ARDUINO: failed to setup rest client");
85.     while(1);
86.   }
87.   debugPort.println("ARDUINO: setup wifi");
88.   esp.wifiCb.attach(&wifiCb);
89.   esp.wifiConnect("CALOCA","2301L0412C");
90.   debugPort.println("ARDUINO: system started");
91. }
92. void loop() {
93.   String estadoLatch;
94.   checkStatus();
95.   if(respuestaEstado != ""){  //TRATAMIENTO DE RESPUESTA DEL LATCH
96.     char myString[266];
97.     for(int i = 0; i < 265; i++){
98.       myString[i] = respuestaEstado.charAt(i);
99.     }
100.            Serial.println("myString => ");
101.            Serial.println(myString);
102.
103.            if(respuestaEstado.indexOf("\"on\"") != -1){
104.              estadoLatch = "abrir";
105.            }else{
106.              if(respuestaEstado.indexOf("\"off\"") != -1){
107.                estadoLatch = "cerrado";
108.              }else{
109.                estadoLatch = "NO FUNCIONA";
110.              }
111.            }
112.              Serial.println("\n\n");
113.              Serial.println("ESTADO LATCH =>  ");
114.              Serial.println(estadoLatch);
115.              if(estadoLatch == "abrir"){
116.                if(puertaCerrada == true){
117.                  giraIzqMotor();
```

```
118.              }
119.            }else{
120.              if(puertaCerrada == false){
121.                giraDerechaMotor();
122.              }
123.            }
124.          respuestaEstado = "";
125.          }
126.        }
127.
128.        void giraIzqMotor(){     // ABRIR
129.          while(steps_left>0)
130.                {
131.                  stepper() ;    // Avanza un paso
132.                  steps_left-- ;  // Un paso menos
133.                  delay (1) ;
134.                }
135.              delay(300);
136.              Direction=!Direction;
137.              steps_left=4095;
138.              puertaCerrada = false;
139.        }
140.
141.        void giraDerechaMotor(){  //CERRAR
142.          while(steps_left>0)
143.                {
144.                  stepper() ;    // Avanza un paso
145.                  steps_left-- ;  // Un paso menos
146.                  delay (1) ;
147.                }
148.              delay(300);
149.              Direction=!Direction;
150.              steps_left=4095;
151.              puertaCerrada = true;
152.        }
153.        void stepper()           //Avanza un paso
154.        {
155.          digitalWrite( IN1, Paso[Steps][ 0] );
156.          digitalWrite( IN2, Paso[Steps][ 1] );
157.          digitalWrite( IN3, Paso[Steps][ 2] );
158.          digitalWrite( IN4, Paso[Steps][ 3] );
159.
160.          SetDirection();
161.        }
162.        void SetDirection(){
163.            if(Direction)
164.               Steps++;
165.            else
166.               Steps--;
167.
168.            Steps = ( Steps + 7 ) % 7 ;
169.        }
```