Universidad de Alcalá
Escuela Politécnica Superior

Grado en ingeniería de computadores

Trabajo Fin de Grado

Demostrador IoT interactivo mediante redes sociales

Autor: María Anguita García

Tutor/es: Antonio García Herraiz

TRIBUNAL:

Presidente: Bernardo Alarcos Alcáraz

Vocal 1º: Juan Antonio Rodrigo Yanes

Vocal 2º: Antonio García Herraiz

2021-2022

Contenido

Índice de ilustraciones

Resumen

Debido a los avances y cambios que ha sufrido la tecnología en los últimos años, actualmente vivimos rodeados de sistemas inteligentes y herramientas, como las redes sociales, que nos hacen la vida más sencilla. Sin embargo, la mayoría de la gente que los utiliza, no sería capaz de explicar cómo funcionan dichos objetos.

En este proyecto se plantea el funcionamiento básico de los sistemas que nos rodean hoy en día, los dispositivos IoT, y nos ayudan a automatizar muchas de las tareas que hacemos cotidianamente.

Summary

Due to the advances and changes that technology has gone throught in the last few years, we now live surrounded by intelligent systems and tools, like social network, that make our lives easier. However, most people who used them, couldn’t be capable to explain how they work.

In this proyect we set out the basic operation of the systems that surround us these days, IoT devices, and help us to automate a lot of tasks that we use in our daily routine.

Resumen extendido

Es normal ver a gente utilizando dispositivos inteligentes, como teléfonos móviles, asistentes virtuales o incluso electrodomésticos. La mayoría de estos dispositivos se basan en sensores. Un ejemplo simple es un horno, este incorpora sensores de temperatura para que podamos controlarla o darnos cuenta si algo no está funcionando de manera correcta.

A todos estos objetos físicos que hemos nombrado anteriormente, se les denomina IoT. Los IoT llevan incorporados sensores que permiten a estos objetos conectarse a internet además de conectarse e intercambiar datos entre ellos. También, nos permite controlar de manera remota cualquier dispositivo. Podemos desde, controlar la calefacción de nuestro hogar desde cualquier lugar del mundo hasta encender las luces de una habitación utilizando nuestra voz.

Esta tecnología se ha convertido en una de las más importantes del siglo XXI y es por eso que en los últimos años, hemos notado un incremento en la cantidad de sistemas inteligentes que nos rodean. Sobre todo, las nuevas generaciones, que son incapaces de vivir sin un teléfono móvil con el que conectarse a cualquier red social. Debido a todo esto nos hemos planteado acercarnos a esta nueva tendencia, para enseñar a los jóvenes el funcionamiento más básico de estos dispositivos.

Las redes sociales son herramientas que nos permiten comunicarnos a través de internet con otros usuarios. Se han hecho tan populares, debido a lo sencillas e intuitivas que son sus interfaces. Además, es muy fácil crearse una cuenta en cualquiera de ellas ya que apenas nos piden datos para registrarnos.

Todas estas redes sociales están llenas de *bots*, esto son programas informáticos capaces de realizar tareas automáticamente mediante internet. Cada uno de ellos es programado para realizar una función concreta. Por ejemplo, responder a mensajes de los usuarios de manera automática, publicar información o recopilar datos.

El objetivo de nuestro proyecto es explicar a través de las redes sociales como un dato recogido por un sensor de temperatura es capaz de llegar hasta nuestros dispositivos. Todo esto lo haremos creando *bots* en las distintas redes sociales y programándolos para que realicen cada uno una función diferente.

Introducción

1.1 Objetivos

La finalidad de este trabajo es conseguir que los jóvenes sean capaces de entender la tecnología básica de los sistemas inteligentes que nos rodean. Para ello, vamos a aprovechar la repercusión que tienen las distintas redes sociales.

La mayoría de estos dispositivos se basan en sensores. Un ejemplo simple es un horno, este incorpora sensores de temperatura para que podamos controlarla o darnos cuenta si algo no está funcionando de manera correcta. Por este motivo el objetivo de nuestro trabajo consiste en crear un sistema sencillo de IoT que nos muestre la información de los sensores a través de las redes sociales.

1.2 Planteamiento

El proyecto se ha dividido en varias partes, comenzando por el estudio de arduino y los distintos sensores con los que vamos a trabajar, continuando con un análisis del funcionamiento del protocolo MQTT y finalizando con la evaluación de las librerías que vamos a utilizar para construir los bots de las distintas redes sociales. De manera más general, podemos desglosar el proyecto en dos partes, hardware y software.

1.3 Herramientas

Para la realización de este proyecto necesitaremos los siguientes medios:

- Un ordenador con Windows o Ubuntu: todas las aplicaciones y plataformas que vamos a utilizar pueden instalarse en ambos entornos, aunque en nuestro caso y debido a la familiaridad hemos decidido utilizar Windows.
- Una placa protoboard y los diferentes sensores: para realizar el diseño del circuito que posteriormente programaremos.
- Una placa de Arduino: Podemos utilizar tanto el modelo *Arduino nano* o *Arduino uno*, aunque nosotros hemos optado por el primero, ya que es más compacto y consume menos energía que el segundo.

1. 4 Estructura del proyecto

La memoria se divide en cinco apartados. El primero de ellos, es en el que hemos explicado los objetivos y el planteamiento general del trabajo.

El apartado dos, ofrece un resumen de la parte teórica estudiada para su posterior implementación. Comenzando por el estudio de los elementos de la parte hardware, continuando con el análisis de las mejores herramientas para llevar a cabo la parte software.

En los apartados tres y cuatro, podemos ver una explicación más extensa sobre la implementación de las partes ya mencionadas. Cada una de ellas está dividida en subapartados, en los que explicamos paso a paso como utilizar las diferentes plataformas.

Por último, en el apartado número cinco encontramos las conclusiones obtenidas después de la realización del proyecto, así como problemas que nos han sido surgiendo.

Bases del proyecto

Este proyecto está basado en un dispositivo IoT que busca comunicarse a través de las redes sociales. Para realizarlo necesitamos un conocimiento previo acerca de todos los elementos.

En los siguientes subapartados, podemos encontrar una breve descripción teórica de las características de cada uno de los componentes que vamos a utilizar.

2.1 Hardware

La parte hardware es la que vamos a dedicar a la realización del dispositivo IoT. Para ello, debemos estudiar cada uno de los componentes que lo conforman y explicar el porqué de su elección.

2.1.1 Sensores

Los sensores, son elementos que nos permiten detectar cambios físicos y los convierte en una salida analógica o digital, legible para los humanos. Hemos elegido los siguientes:

- Sensor de temperatura y humedad: el modelo dht11 a diferencia de otros como el LM35 nos permite medir la temperatura y humedad de manera digital, lo que nos protege frente al ruido. Podemos ver sus especificaciones [1] en la siguiente tabla.

TEMPERATURA	
Rango	De 0ºC a 50ºC
Precisión	A 25ºC ± 2ºC
HUMEDAD	
Rango	De 20% a 90% RH
Precisión	Entre 0ºC y 50ºC ± 5% RH

- Sensor de movimiento: para el sensor de movimiento hemos elegido el modelo HC-SR50. Este nos indica si se ha detectado o no movimiento en un rango de 3 a 7 metros y con aperturas de 90 a 110º. En [2] nos detallan más extensamente las propiedades de este sensor.

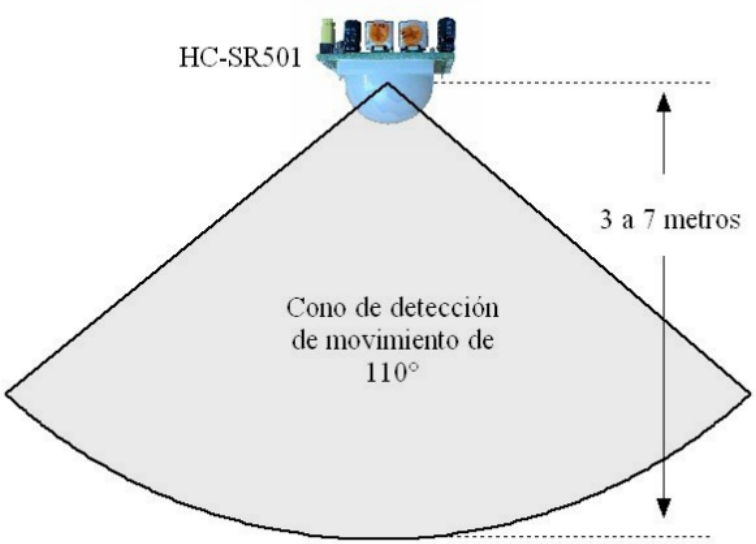

Ilustración 1. Sensor de movimiento

- Sensor de sonido: hemos escogido el modelo KY-038 [3]. Nos permite leer la salida tanto digital como analógica. Nosotros hemos pensado que para nuestro proyecto es más práctico optar por leer la salida digital, ya que no nos importa el nivel de ruido si no el hecho de que haya o no sonido. Para obtener bien la salida debemos ir ajustando la sensibilidad del sensor.

Ilustración 2. Sensor de sonido

- Sensor de luz: modelo KY-018 [4]. Este módulo viene con una fotorresistencia incluida para poder medir la intensidad de la luz. El valor que obtenemos será muy elevado si apenas hay luz y descenderá en caso de aumentar la intensidad lumínica.

2.1.2 Placa de Arduino

Como ya hemos comentado en el apartado *herramientas*, hemos optado por el modelo Arduino nano, ya que es una versión en miniatura del Arduino UNO lo que hace que sea más cómodo. Además, al ser más compacto consume menos energía. En la siguiente imagen podemos ver el modelo elegido y en [5] una mayor explicación de ambos compuestos.

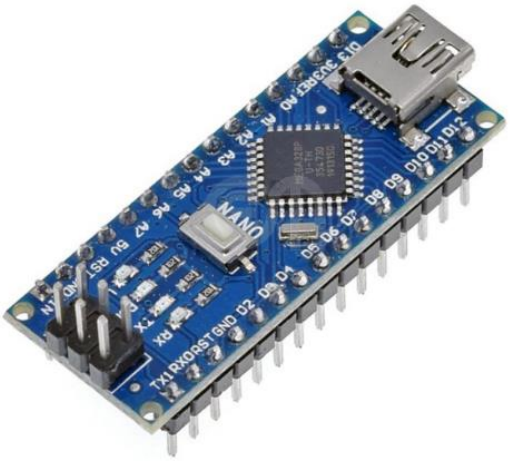

Ilustración 3. Arduino Nano

2.1.3 PCB

Una PCB es una tarjeta que lleva un circuito impreso [6]. Está construida por capas que alternan material conductor con material no conductor.

Para diseñarlas, hoy en día hay programas y herramientas especializados que nos ayudan a realizar las conexiones y simular nuestro circuito. Una vez impreso el circuito, podemos soldar nuestros componentes siguiendo las conexiones que hemos realizado previamente.

En nuestro caso, hemos elegido fabricar una PCB debido a que es más compacto que tener el circuito en una protoboard, por lo que es más sencillo transportarla.

2.2 Software

En la parte de software, es en la que vamos a programar el dispositivo IoT y lo vamos a conectar con las distintas redes sociales. Para ello, primero debemos ver la manera de comunicación estudiando las distintas opciones que tenemos, y posteriormente, veremos el funcionamiento de las redes sociales.

2.2.1 Comunicación entre sistemas

Lo más importante para realizar este proyecto es ser capaces de unir la parte hardware con la parte software, para ello necesitamos un protocolo de comunicación. Para elegir el más adecuado, primero hemos barajado las siguientes opciones:

- MQTT: es un protocolo de comunicación que funciona con el patrón publicador/subscriptor. Los mensajes son almacenados en un servidor central llamado broker, ahí estos se organizan en topics. Cada cliente puede publicar en un topic o subscribirse de forma que el broker le devolverá el mensaje correspondiente [7].
  Los clientes deben iniciar una conexión TCP/IP con el broker a través del puerto 1883, la cual no se cerrará hasta que el mismo cliente la finalice.
- CoAP: es un protocolo diseñado especialmente para su uso entre dispositivos restringidos. Su funcionamiento se basa en nodos. Uno envía un paquete a otro y este será extraído por el servidor CoAP, el cual lo interpretará y decidirá qué hacer con el mismo [9].
  Además, no tenemos por qué recibir una confirmación de que el paquete ha cumplido la petición.
- AMQP: este protocolo es bastante parecido a MQTT, ya que se basa en topics, publicadores y suscriptores. Sin embargo, es más complejo ya que intenta suplir a MQTT en los ámbitos que este no controla [8].
- HTTP: es un protocolo que sigue el patrón cliente/servidor. Su funcionamiento es el siguiente, un cliente pregunta por un dato al servidor y este le responde. Es rápido y sencillo y permite mandar distintos tipos de datos, no únicamente texto, también pueden ser archivos multimedia.

Después de estudiar estos cuatro protocolos, hemos decidido estudiar más a fondo MQTT y HTTP ya que se ajustan más a lo que necesitamos para nuestro proyecto. En la siguiente tabla y en [10] podemos ver las características más señaladas de ambos.

	MQTT	HTTP
Modelo	Publicador/Suscriptor	Cliente/Servidor
Cabecera	2 Bytes	26 Bytes
Calidad del servicio	3 niveles	No implementa
Retención de mensajes	Sí	No

Por todas estas especificaciones nuestra elección es MQTT, debido a su sencillez y a lo extendido que esta su uso. Además, es un protocolo muy útil para dispositivos IoT, ya que normalmente estos poseen escasa potencia y este protocolo es muy ligero y fácil de utilizar.

Además, el patrón suscriptor/publicador hace que ningún cliente esté conectado directamente con otro, por lo que si hay algún fallo en el sistema no habrá un bloqueo total en este. Para entender MQTT debemos saber algunos de los términos más importantes.

- Cliente MQTT: es una aplicación cliente que se conecta a internet implementando MQTT para poder recibir o enviar mensajes.
- Topic: es un identificador que se utiliza para clasificar los mensajes de manera jerárquica.
- Publicador: es un cliente MQTT que envía mensajes.
- Suscriptor: es un cliente MQTT que recibe mensajes de un cierto topic al que está suscrito.
- Broker MQTT: es un servidor que controla tanto a los clientes como el almacenamiento de los mensajes.

Después de entender los conceptos básicos, podemos pasar a ver un ejemplo de su funcionamiento de manera esquemática en la siguiente ilustración.

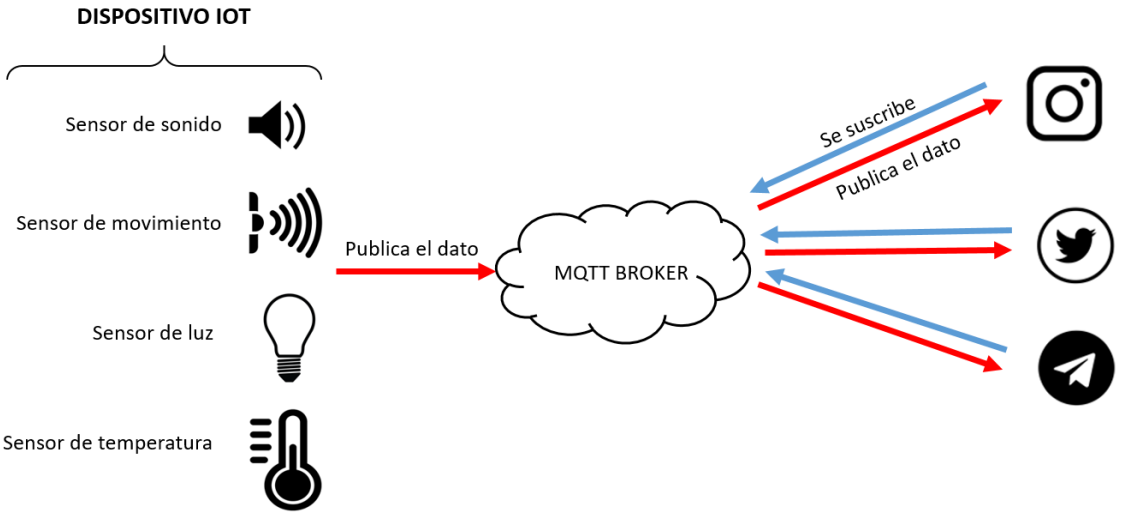

Ilustración 4. Esquema MQTT

2.2.1.1 Librería Paho

Además de por las características que hemos visto, hemos optado por MQTT debido a que podemos hacer uso de la librería *paho* [11]. Esta, implementa una clase cliente, la cual nos va a permitir crear un cliente MQTT capaz de publicar y suscribirse a los canales. Para ello, debemos importar la librería paho y crear el cliente tal y como vemos a continuación.

```
import paho.mqtt.client as mqtt #import the client
client = mqtt.Client()
```

El siguiente método que vamos a utilizar es el *client.connect()* sus parámetros son los siguientes, aunque son opcionales.

- Host: indica el broker al que vamos a conectarnos.
- Port: especifica el puerto por el que nos conectaremos al broker.
- Keepalive: tiempo que se va a mantener abierta la conexión.

```
client.connect("test.mosquitto.org", 1883, 60)
```

El otro método será el que publique los mensajes en los canales, *client.publish()*. Para este programa no nos hará falta el suscriptor ya que no vamos a leer ningún dato. Los parámetros son los siguientes:

- Topic: indica el topic en el que vamos a publicar los mensajes.
- Payload: mensaje que vamos a publicar en el canal.
- Retain: es un booleano que indica si publicamos los mensajes con o sin retención. Esto quiere decir, que almacena el último mensaje hasta que se recibe uno nuevo.
- QoS: especifica el nivel de calidad de servicio deseado. Por defecto es 0. En nuestro caso elegimos el 1 ya que garantiza la entrega del mensaje.

```
client.publish(topic_movimiento,total,retain=True,qos=1)
```

2.3 Redes sociales

Las redes sociales son plataformas digitales que permiten a los usuarios compartir mensajes e información. Actualmente, su uso está muy extendido sobre todo entre gente joven, aunque muchas empresas las utilizan también para promocionarse y hacer negocio. En nuestro caso hemos optado por elegir las más populares entre los jóvenes:

- Instagram: permite a los usuarios subir fotos o videos a su perfil e interaccionar con estos y comunicarse a través de sus mensajes directos.
- Twitter: al contrario que Instagram, esta red social está más orientada a compartir publicaciones con mensajes de texto cortos, aunque también se pueden compartir imágenes. Lo que sí tienen en común es que permite a los usuarios establecer conversaciones a través de sus mensajes directos.
- Telegram: es una plataforma de mensajería, aunque se caracteriza por sus grupos, que pueden estar formados por más de 20.000 usuarios. El administrador de los mismos, establece si los usuarios pueden compartir información por el canal o únicamente leer lo que ellos publican.

Después de la explicación teórica, vamos a continuar con el desarrollo del proyecto, indicando los pasos a seguir en las diferentes plataformas que vamos a utilizar.

Desarrollo Hardware

En nuestro caso, vamos a basar esta parte en dos plataformas:

- *Arduino*: esta plataforma la utilizaremos tanto en la parte hardware como en la parte software. La elección de utilizar esta herramienta es debido a lo extendido que esta su uso, ya que esto nos facilita a la hora de encontrar documentación y comprar componentes. Además, es muy común verla entre gente principiante, por la facilidad y el bajo coste para realizar aplicaciones o instrumentos.
- *EasyEDA:* es un conjunto de herramientas que nos va a ayudar a realizar una simulación de nuestro circuito para posteriormente, obtener la PCB del mismo.

En los siguientes subapartados podemos ver una breve explicación de los pasos que debemos seguir para usar estas herramientas.

3.1 Arduino

En este apartado veremos cómo diseñar físicamente el circuito que deseemos. Para ello, necesitamos una *protoboard*, los sensores que hemos especificado anteriormente, un led RGB, tres resistencias de 220 Ω y cables para realizar las conexiones.

Para ir conectando todos los sensores, necesitamos fijarnos en los *datasheet* de los componentes. Esto es una hoja técnica en la que nos especifican donde debemos conectar cada uno de los pines. El módulo principal de nuestro circuito es la placa de arduino nano, por lo que el resto de componentes deberán ir conectados a ella.

El resultado es el que vemos a continuación.

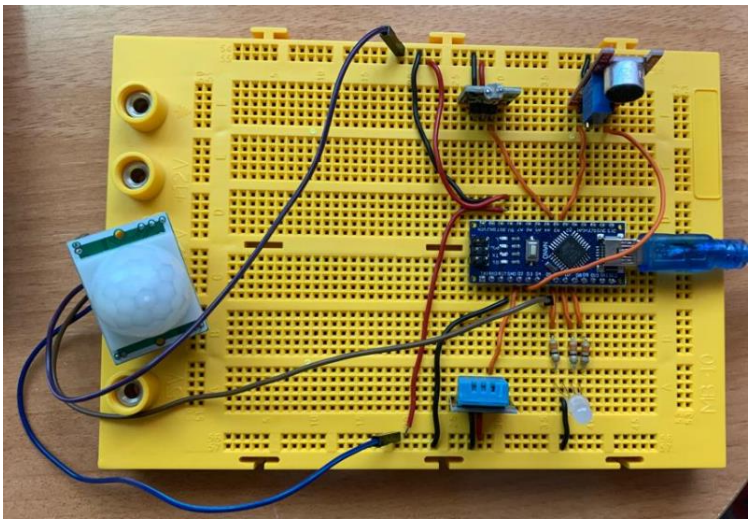

Ilustración 5. Circuito en la protoboard

3.2 EasyEDA

3.2.1 Instalación

El siguiente paso a seguir será instalarnos EasyEDA para construir la PCB, una placa con el circuito impreso a la que posteriormente deberemos soldar los componentes. Para su instalación debemos dirigirnos a su página oficial [12] y descargar el ejecutable. Después de su descarga únicamente debemos elegir la ubicación y darle a continuar.

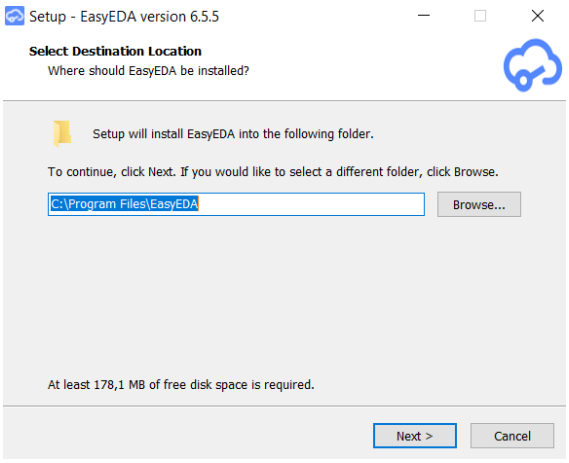

Ilustración 6. Instalación EasyEDA

Una vez instalado, se nos abre automáticamente la aplicación y nos pide que creemos una cuenta en la que guardar nuestros proyectos, o acceder a través de Google. En nuestro caso decidimos lo segundo debido a la comodidad.

3.2.2 Implementación del circuito

Para implementar nuestro circuito en el simulador, primero tenemos que crear un nuevo proyecto tal y como vemos en la siguiente imagen.

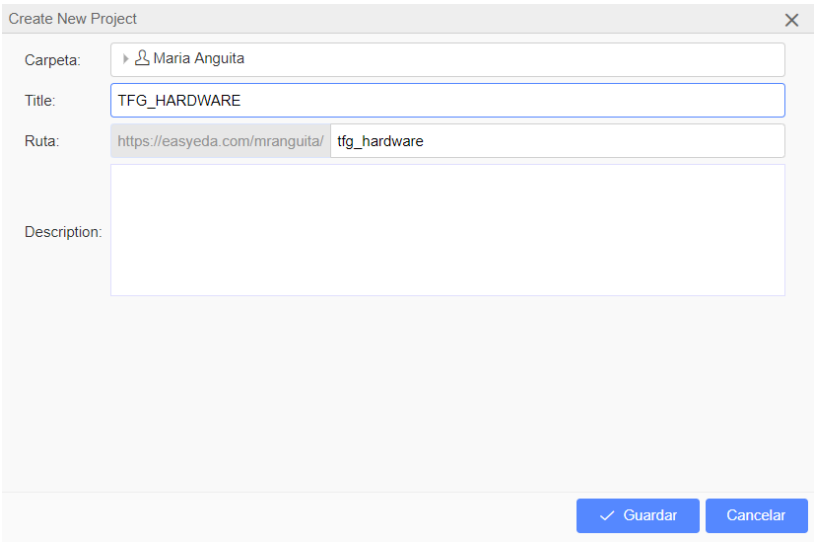

Ilustración 7. Crear proyecto EasyEDA

Ahora ya podemos comenzar a diseñar nuestro circuito. Para ello, esta plataforma nos proporciona una librería donde podemos encontrar una simulación de cada uno de los componentes que necesitamos. Incluso nos proporcionan el precio y un enlace para comprarlos.

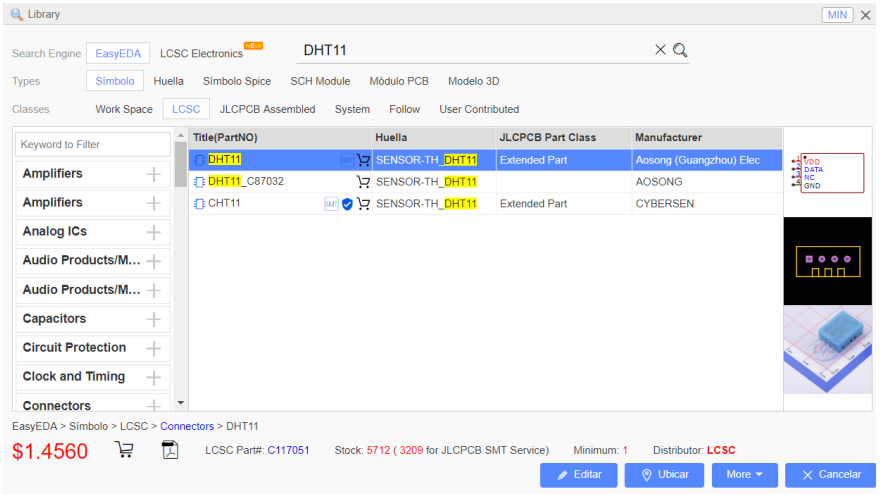

Ilustración 8. Librería de componentes EasyEDA

3.2.3 Resultado del circuito

Después de encontrar todos los componentes de nuestro circuito, podemos comenzar a realizar las conexiones, añadiendo también la toma a tierra y a corriente. El resultado es el que vemos en la siguiente imagen.

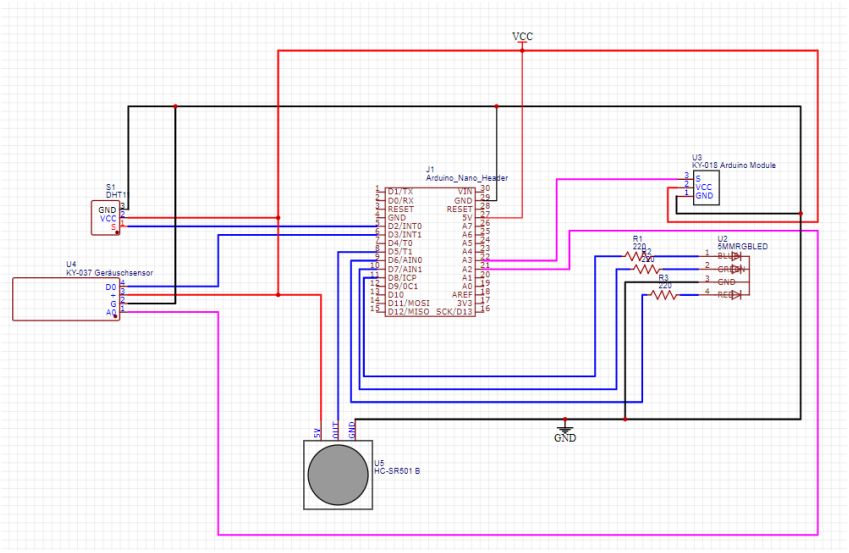

Ilustración 9. Circuito EasyEDA

3.2.4 Implementación de la PCB

Después de tener el circuito implementado, podemos pasar a la realización de la PCB. Para ello debemos tener en cuenta que todos los componentes de nuestro circuito llevan incluido un *footprint*, esto es un patrón de los componentes eléctricos que se utiliza como guía para soldar correctamente los componentes. En la siguiente imagen podemos ver como obtendríamos el sensor DHT11 al buscarlo en la librería de EasyEDA.

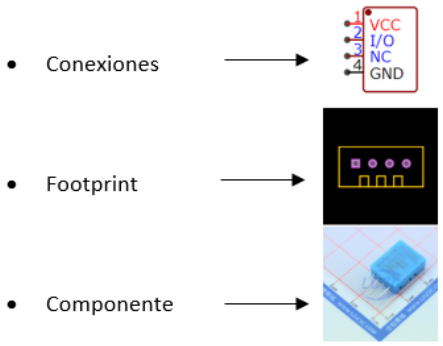

Ilustración 10. Footprint EasyEDA

Una vez nos hemos asegurado de esto, podemos convertir nuestro esquema a PCB con un simple botón.

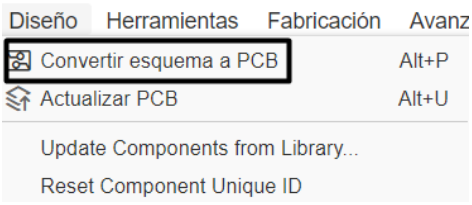

Ilustración 11. Convertir a PCB EasyEDA

Al seleccionar esta opción, nos devolverá el contorno de una PCB y el *footprint* de todos los elementos del circuito para que los dispongamos a nuestro gusto dentro de la misma.

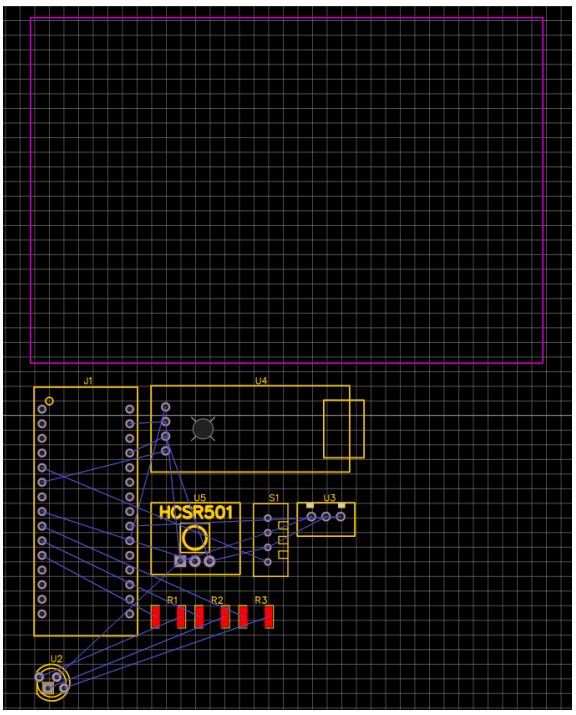

Ilustración 12. PCB EasyEDA

Una vez los hemos colocado, nos dan la opción de realizar las conexiones manualmente o de forma automática. La segunda podemos hacerlo de la siguiente manera.

Ilustración 13. Hacer conexiones EasyEDA

De forma manual, tendremos que ir cambiando de capa, ya que ningún cable puede pasar por encima a otro ya que provocaría un cortocircuito. Esto lo evitaríamos

utilizando la herramienta *vía*, la cual nos permite mover los cables entre las capas. Podemos encontrarlas en la siguiente imagen marcadas con un cuadrado negro en el menú *herramientas PCB.*

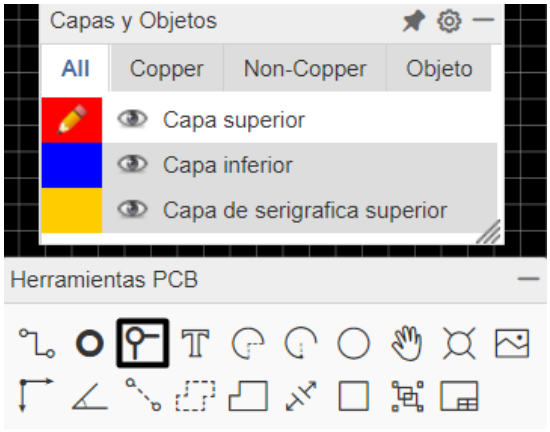

Ilustración 14. Herramientas PCB EasyEDA

3.2.4 Resultado de la PCB

Después de realizar las conexiones, el resultado obtenido es el siguiente. Podemos ver en rojo las conexiones de la capa superior y en azul las de la capa inferior.

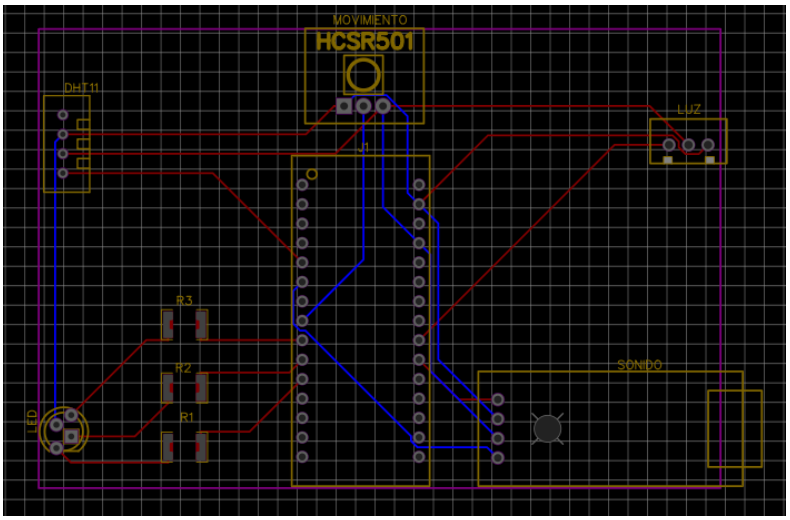

Ilustración 15. Resultado PCB

Esta placa podemos obtenerla de manera física desde la misma aplicación. En la parte superior derecha encontramos el siguiente símbolo.

Ilustración 16. Archivo gerber EasyEDA

Este nos genera un archivo *gerber*. Este es el que debemos enviar a la hora de solicitar nuestra PCB física ya que contiene toda la información necesaria de nuestra placa.

En el menú de la parte izquierda localizamos la tienda para hacer nuestro pedido. Elegimos el diseño de la misma, en nuestro caso dejaremos las características por defecto, y solicitamos nuestra placa. Antes de realizar el pedido nos dan varias opciones económicas en función de los días que tarde el envío.

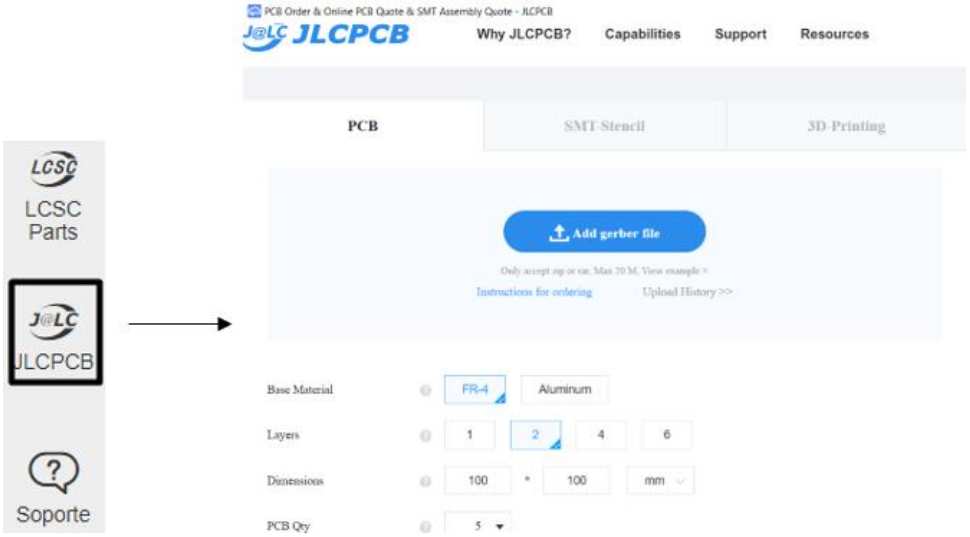

Ilustración 17. Pedido de la PCB

Nos llegarán a casa 5 PCB idénticas, ya que es el mínimo que se puede pedir, como las de la siguiente imagen.

Ilustración 18. PCB física

Desarrollo Software

En este apartado, explicaremos todos los pasos a seguir para realizar la parte de software. Las herramientas que utilizaremos serán las siguientes:

- *Arduino:* la parte software de esta plataforma, la dedicaremos a programar el circuito que hemos diseño anteriormente.
- *Mosquitto:* Eclipse mosquitto es un intermediario de mensajes que implementa el protocolo MQTT. Tiene librerías que nos permiten crear clientes y suscribirnos y publicar en los topics creados. Es fácil de utilizar para el usuario, aunque esto hace que cualquiera llegue a ser capaz de publicar datos en un canal y lo llene de información no deseada para el resto de usuarios.
  Además, nos ofrecen un servidor de prueba para que nos familiaricemos con la herramienta.
- *Python*: es un lenguaje de programación muy sencillo ya que se acerca mucho al lenguaje humano. En nuestro caso, vamos a utilizar la herramienta *visual studio code*, ya que es un editor de código fuente que nos permite trabajar con diversos lenguajes de programación y la depuración de código es muy sencilla e intuitiva.

4.1 Arduino

4.1.1 Instalación

El primer paso para utilizar arduino es descargarnos la aplicación software. El archivo de instalación podemos encontrarlo de manera gratuita en su página oficial [13]. Una vez descargado, ejecutamos el archivo y aceptamos la licencia.

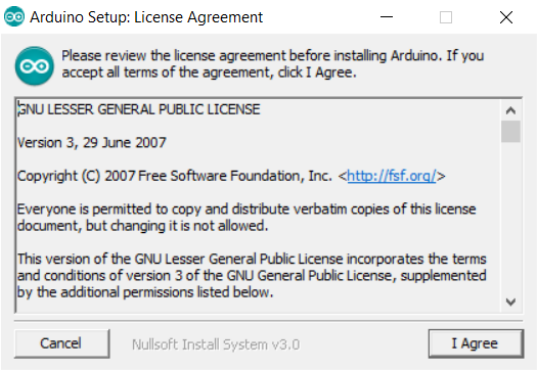

Ilustración 19. Instalación Arduino 1

En el siguiente paso, elegimos lo que queremos instalar, en nuestro caso lo dejaremos como esta en la imagen que vemos a continuación y continuamos.

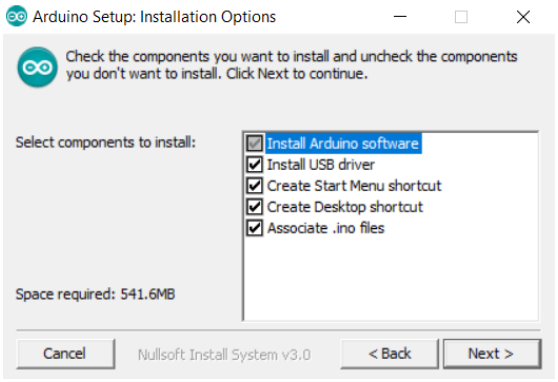

Ilustración 20. Instalación arduino 2

Por último, elegiremos la ruta en la que vamos a instalar el programa. Al igual que en el paso anterior nosotros dejaremos la que se indica en la siguiente imagen, que es la que se establece por defecto

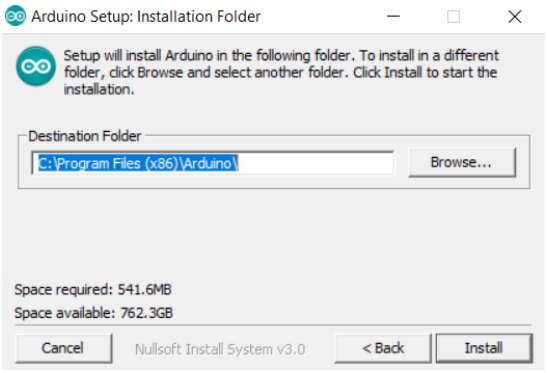

Ilustración 21. Instalación arduino 3

4.1.2 Configuración

Antes de ejecutar nuestro código debemos comprobar que tenemos la aplicación bien configurada, seleccionando el modelo de nuestra placa, su procesador y el puerto al que la vamos a conectar.

Cada placa de arduino utiliza un procesador diferente, el cual podemos encontrar también en su página web oficial. En nuestro caso necesitamos la de Arduino nano podemos encontrar su configuración en [14].

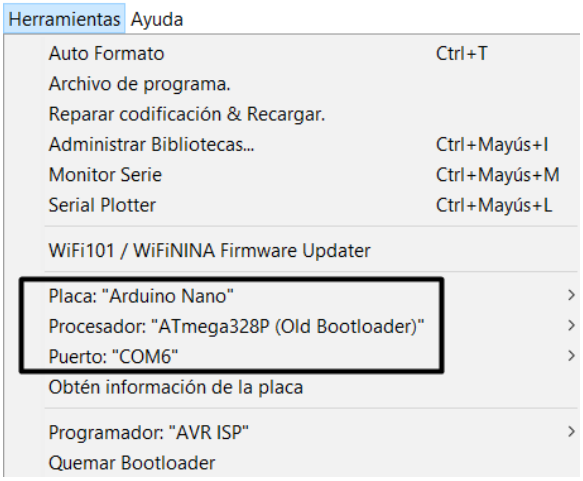

Ilustración 22. Configuración Arduino

4.1.3 Librerías

En cuanto a las librerías que vamos a utilizar, únicamente necesitamos importar la del sensor de temperatura, DHT11. Podemos hacerlo de dos maneras diferentes.

- Buscándola en su página oficial [15] y descargando la última versión. El archivo que obtendremos es un zip que debemos incluir en la aplicación de la siguiente forma.

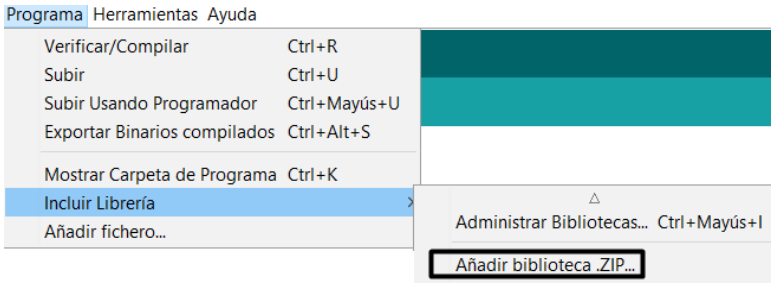

Ilustración 23. Añadir librerías Arduino (opción 1)

- Instalándola directamente desde la aplicación software. Lo primero que haremos será lo siguiente.

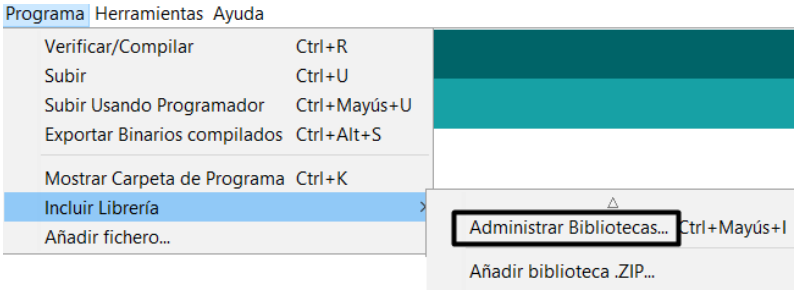

Ilustración 24. Añadir librerías Arduino (opción 2) 1

Con esto abriremos el gestor de librerías, donde buscaremos el nombre, DHT11 pulsaremos en el botón de instalar. De esta forma ya estaría añadida.

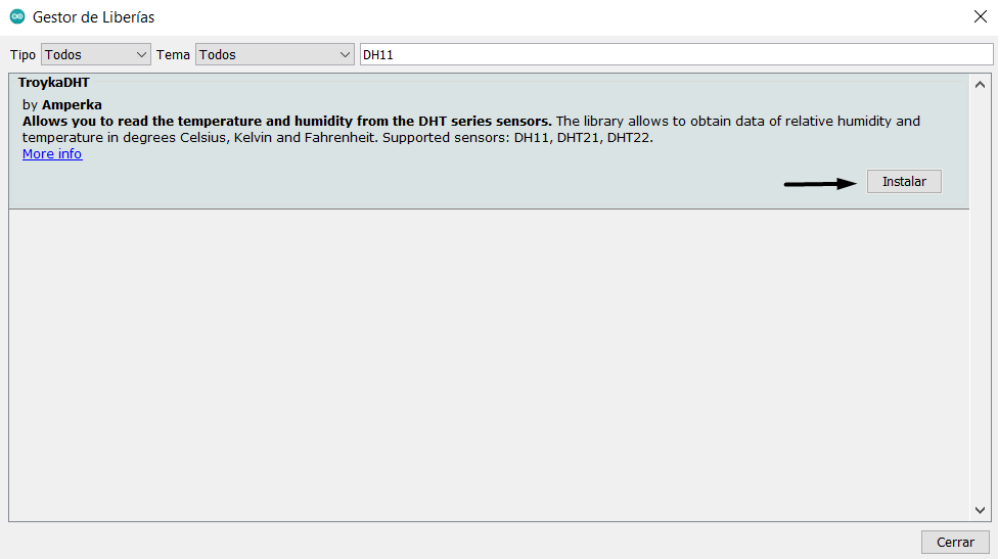

Ilustración 25. . Añadir librerías Arduino (opción 2) 2

Para la implementación de nuestro proyecto, comenzaremos por realizar el circuito en la protoboard. Una vez hecho esto, lo programaremos para obtener los valores deseados.

4.1.4 Implementación

Para el diseño del programa, debemos tener en cuenta a que pines del arduino nano hemos conectado cada uno de los componentes. Ya que, lo primero que haremos, será importar las librerías e instanciar variables con el pin de cada uno de ellos. En la siguiente ilustración podemos ver un ejemplo.

```
#include <DHT.h>
#define DHTPIN 2 //Esta en el pin2
#define DHTTYPE DHT11 //Definimos el tipo
nt SensorM = 5; //Sensor de movimiento
int SensorL = A3; //Sensor de luz
```

Después de esto, iniciaremos la comunicación con el ordenador a través de la instrucción *Serial.begin* y comenzaremos a leer los datos de los sensores. Lo que hacemos en la siguiente imagen lo repetiremos con todos los sensores.

```
Serial.begin(9600); //Indica el inicio de la comunicacion con la frecuencia }

void loop() {

  int DatoM = digitalRead(SensorM); //Leemos el valor del sensor de
movimiento
  int DatoLuz = analogRead(SensorL); //Leemos el dato del sensor de luz y lo
guardamos
```

Seguido de esto, debemos programar cada uno de los sensores. En el caso del sensor de movimiento, pondremos a 0 o a 1 la variable en función de lo que detecte el sensor.

```
if (DatoM == HIGH)    // Si detecta movimiento estará a 1
  {
     Serial.println ("Se ha detectado movimiento");
     digitalWrite(LEDA, HIGH);  //Encendemos el led
     delay(1000);
  }
  else // Si no se detecta movimiento
  {
     Serial.println ("No se ha detectado movimiento");
     digitalWrite(LEDA, LOW); //Apagamos el led
     delay(1000);
  }
```

Esto mismo, lo haremos con el sensor de sonido.

En cambio, para el resto de los sensores, temperatura, humedad y luz, estableceremos un baremo para comprobar si los valores son muy altos o no, en función de esto encenderemos el led o lo apagaremos. Además, obtendremos por pantalla el valor numérico del sensor en lugar de una frase indicándonos su estado.

```
//Suponemos que 24 grados es un valor alto,
// si superamos ese valor encenderemos el LED verde
if (temperatura > 24)
{
digitalWrite (LEDV, HIGH);
  delay(2000);
}
else
{
  digitalWrite (LEDV, LOW);
  delay(1000);
}
```

4.1.5 Resultado

Una vez hemos terminado el programa, procedemos a compilarlo para ver si hay algún error que debamos corregir. Este paso, tenemos que hacerlo con la protoboard conectada a través del USB de la placa de arduino a nuestro ordenador.

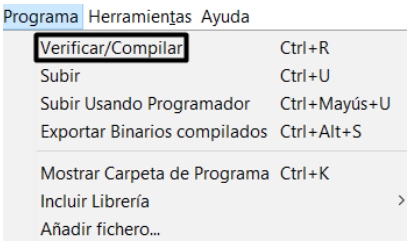

Ilustración 26. Compilar programa arduino

Si no obtenemos ningún error, podemos subir el archivo de la siguiente forma.

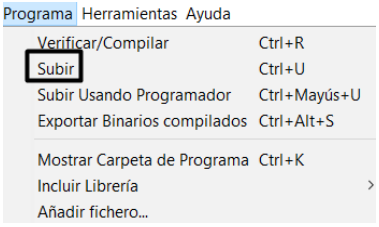

Ilustración 27. Subir programa arduino

Por último, abrimos el monitor seria para ver si el programa nos devuelve lo que deseamos.

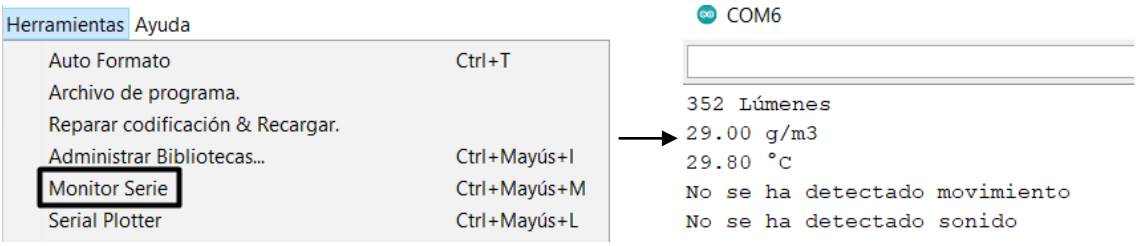

Ilustración 28. Resultado de nuestro programa

4.2 Mosquitto

4.2.1 Instalación

Para instalar *mosquitto*, vamos a repetir el mismo proceso que en el resto de las plataformas. Nos vamos a su página web [16] y descargamos el ejecutable. Una vez descargado, ejecutamos el archivo y le damos a continuar hasta que aparezca el botón de instalar.

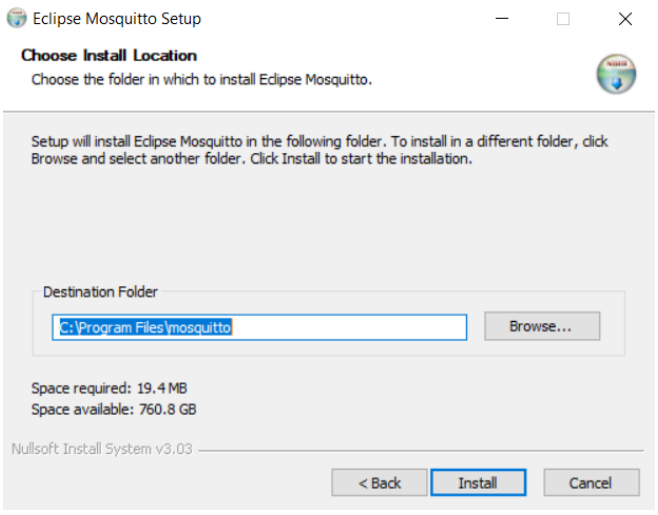

Ilustración 29. instalación de mosquitto

4.2.2 Configuración

Lo primero que debemos hacer a la hora de configurar mosquitto es saber por qué puerto está escuchando. Para ello utilizamos el comando *netstat –an* sin iniciar mosquitto, este nos indica los puertos que están activos.

Después iniciamos mosquitto con el comando *mosquitto –v* desde la ubicación en la que hemos instalado el archivo. Como vemos en la siguiente imagen ya nos dice que está escuchando en el puerto 1883.

```
C:\Program Files\mosquitto>mosquitto -v
1654156415: mosquitto version 2.0.14 starting
1654156415: Using default config.
1654156415: Starting in local only mode. Connections will only be possible from clients running
 on this machine.
1654156415: Create a configuration file which defines a listener to allow remote access.
1654156415: For more details see https://mosquitto.org/documentation/authentication-methods/
1654156415: Opening ipv4 listen socket on port 1883.
1654156415: Opening ipv6 listen socket on port 1883.
1654156415: mosquitto version 2.0.14 running
```

Ilustración 30. Ejecutar mosquitto

Una vez iniciado ejecutamos de nuevo el comando *netstat –an* para ver que puerto nuevo hay escuchando, este será el que está utilizando mosquitto. A la izquierda de la imagen vemos los puertos activos antes de ejecutar mosquitto y a la derecha una vez ejecutado, como vemos aparece uno nuevo, el 1883. Tal y como nos indicaban al ejecutar mosquitto.

Conexiones activas		Conexiones activas	
Proto	Dirección local	Proto	Dirección local
TCP	0.0.0.0:135	TCP	0.0.0.0:135
TCP	0.0.0.0:445	TCP	0.0.0.0:445
TCP	0.0.0.0:808	TCP	0.0.0.0:808
TCP	0.0.0.0:1462	TCP	0.0.0.0:1462
TCP	0.0.0.0:2869	TCP	0.0.0.0:2869
TCP	0.0.0.0:5040	TCP	0.0.0.0:5040
TCP	0.0.0.0:6878	TCP	0.0.0.0:6878
TCP	0.0.0.0:6879	TCP	0.0.0.0:6879
TCP	0.0.0.0:6880	TCP	0.0.0.0:6880
TCP	0.0.0.0:6881	TCP	0.0.0.0:6881
TCP	0.0.0.0:8621	TCP	0.0.0.0:8621
TCP	0.0.0.0:49664	TCP	0.0.0.0:49664
TCP	0.0.0.0:49665	TCP	0.0.0.0:49665
TCP	0.0.0.0:49666	TCP	0.0.0.0:49666
TCP	0.0.0.0:49667	TCP	0.0.0.0:49667
TCP	0.0.0.0:49668	TCP	0.0.0.0:49668
TCP	0.0.0.0:49670	TCP	0.0.0.0:49670
TCP	0.0.0.0:53421	TCP	0.0.0.0:53421
TCP	0.0.0.0:53425	TCP	0.0.0.0:53425
TCP	0.0.0.0:54173	TCP	0.0.0.0:54173
TCP	0.0.0.0:54192	TCP	0.0.0.0:54192
TCP	0.0.0.0:54221	TCP	0.0.0.0:54221
TCP	127.0.0.1:5354	TCP	127.0.0.1:1883
TCP	127.0.0.1:5354	TCP	127.0.0.1:5354
TCP	127.0.0.1:5354	TCP	127.0.0.1:5354
TCP	127.0.0.1:5354	TCP	127.0.0.1:5354
TCP	127.0.0.1:6463	TCP	127.0.0.1:5354
TCP	127.0.0.1:14622	TCP	127.0.0.1:6463

Ilustración 31. Puertos abiertos

Sin embargo, no estamos permitiendo todas las conexiones, únicamente las de *local host* o de la ip 127.0.0.1. Para cambiar esto, ejecutamos la aplicación de bloc de notas como administrador y abrimos el archivo *mosquitto.conf.*

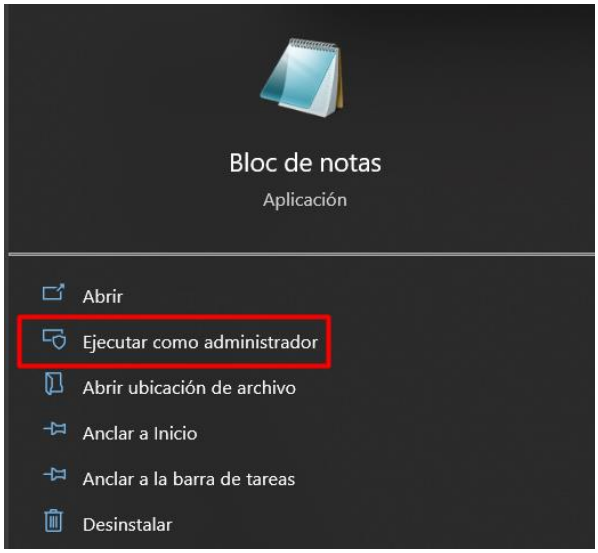

Ilustración 32. Abrir bloc de notas como administrador

Una vez abierto, nos dirigimos a la parte de *listeners* del documento y añadimos el puerto por el que vamos a estar escuchando, el 1883, permitimos cualquier dirección ip y cualquier conexión, aunque sea anónima.

```
# ================================================================
# Listeners
# ================================================================

listener 1883 0.0.0.0
allow_anonymous true
```

Ilustración 33. Modificar archivo de configuración de mosquitto

Por último, utilizamos el comando *mosquitto –c mosquitto.conf* para ejecutar mosquitto con el archivo de configuración que hemos modificado, y comprobamos que las modificaciones se han añadido de manera correcta.

```
Conexiones activas

  Proto  Dirección local        Dirección remota       Estado
  TCP    0.0.0.0:135            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:445            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:808            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:1462           0.0.0.0:0              LISTENING
  TCP    0.0.0.0:1883           0.0.0.0:0              LISTENING
  TCP    0.0.0.0:2869           0.0.0.0:0              LISTENING
```

Ilustración 34. Puertos abiertos

4.2.3 Resultado

Como ya hemos dicho anteriormente, esta plataforma tiene formato suscriptor/publicador por lo que podemos comprobar su funcionamiento con dos simples comandos que encontramos en [17].

El primero de ellos es el de suscripción que tiene el siguiente formato.

```
C:\Program Files\mosquitto>mosquitto_sub -h dirección ip -t nombredelcanal
```

Ilustración 35. Suscripción a un topic

Y el de publicación de mensajes.

```
C:\Program Files\mosquitto>mosquitto_pub -h dirección ip -t nombredelcanal -m "mensaje"
```

Ilustración 36. Publicación en un topic

En la siguiente imagen vemos el resultado de publicar un mensaje en un canal al que estamos suscritos.

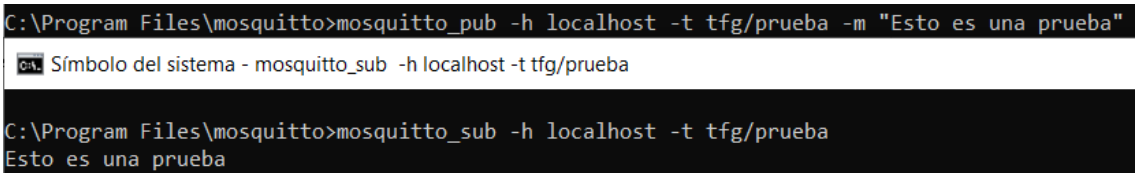

Ilustración 37. Resultado de la publicación de un mensaje

4.3 Python

En cuanto a Python, vamos a utilizar la herramienta *visual studio code*, ya que es un editor de código fuente que nos permite trabajar con diversos lenguajes de programación y la depuración de código es muy sencilla e intuitiva.

4.3.1 Instalación

La instalación de visual studio code es idéntica a la del resto de plataformas. Accedemos a su página web [18] y nos descargamos el ejecutable. Lo abrimos y seguimos los pasos hasta llegar a la pantalla en la que nos permiten instalar la aplicación.

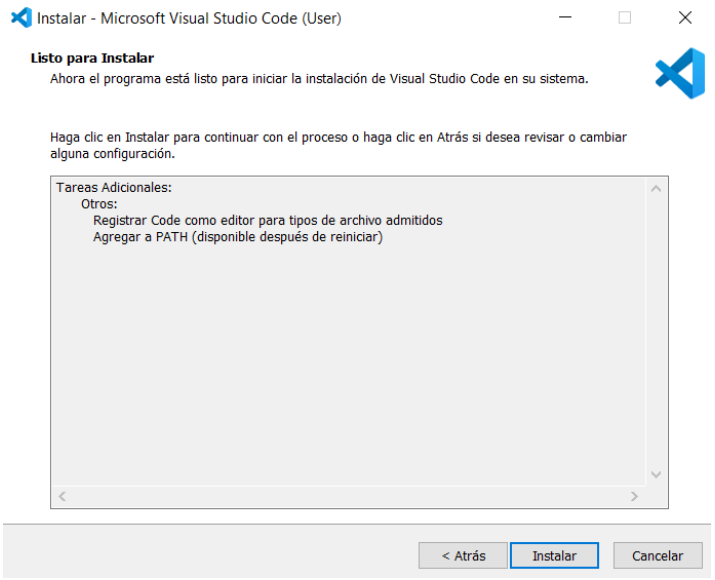

Ilustración 38. Instalación visual studio

Para comenzar a utilizarlo, únicamente debemos crear un archivo y ponerle la extensión del lenguaje de programación que vayamos a utilizar. Como en nuestro caso vamos a usar Python, la extensión será *.py*.

4.4 Archivo de configuración

Para comenzar nuestro proyecto con Python, primero vamos a crear un archivo de configuración en el que añadiremos todas las contraseñas y variables importantes. Esto lo hacemos ya que son datos que vamos a utilizar varias veces en todos los programas que vamos a implementar a continuación, por lo que nos vendrá bien para evitar cometer errores y arrastrarlos. Aquí podemos ver el código con los distintos valores. A este archivo le hemos puesto el nombre de *password.py*.

```
##GENERAL
topic_movimiento = "XXXXXXX/XXXXX"
topic_humedad = "XXXXXXX/XXXXX"
topic_sonido = "XXXXXXX/XXXXX"
topic_temperatura = "XXXXXXX/XXXXX"
topic_luz = "XXXXXXX/XXXXX"
puerto = 'COMX'

##INSTAGRAM
usuario_insta = 'XXXXX'
pssw = 'XXXXXXX'
id = 51896543990

##TWITTER
```

```
consumer_key = "jDrasI2GXXXXXXXX7H9vt5K"
consumer_secret = "YNRZuOmtryZPJCPXXXXXXXXXyiOKVXrA4YnfGde0q"
access_token = "15192740793XXX7587-XXXXXXMX5Ei4ek2bQoX5kS"
access_token_secret = "cXpr7QhXXXXXXXXQPKQaJJ4SZ7FfsmCGONgjdpiX"

##TELEGRAM
api_id = 19XXX12
api_hash = 'b5a5f1e733XXXXXX2245172cc3748821'
```

Por último, también tendremos que importar el programa en los demás scripts. El asterisco, nos indica que añadimos todas las variables.

```
from password import *
```

4.5 Programa para MQTT

Este programa será el enlace entre arduino y las redes sociales. Clasificará los datos recogidos en arduino y los publicará en los distintos canales.

4.5.1 Librería serial

Para conectarnos con el programa de arduino desde Python y leer los datos que este recoge de los sensores, necesitamos la librería serial.py. Para instalarla nos dirigimos en la aplicación símbolo del sistema a la carpeta en la que está instalado Python y ejecutamos el comando *pip install pyserial*.

Importamos la librería como se indica en [19] y nos conectamos indicando el mismo puerto que configuramos en la aplicación de arduino y la frecuencia de la comunicación.

```
import serial
ser = serial.Serial(puerto, 9600) #Nos conectamos con arduino con el puerto y
la frecuencia
```

4.5.2 Implementación

Lo primero que haremos será leer los datos de arduino. Para ello la librería serial implementa el método *readline()*, este lee línea a línea el monitor serie.

Por último, decodificamos el dato y lo guardaremos en la variable resultado.

```
if True:
      dato = ser.readline()
resultado = dato.decode("utf-8")
```

Para publicar los datos en los canales primero los clasificaremos viendo las unidades de cada uno. En el caso del sonido y el movimiento distinguiremos también si su valor es 1 para poder indicar la hora a la que se ha detectado por última vez con la librería *time*.

```
    if "movimiento" in resultado:
        if "1" in resultado:
            resultado = "La ultima vez que se ha detectado movimiento ha sido
el "
            print("Publishing message to topic",topic_movimiento)
            tiempo_cadena = time.strftime("%d-%m-%Y %H:%M", time.gmtime())
            total = resultado+tiempo_cadena
            client.publish(topic_movimiento,total,retain=True,qos=1)
    elif "Lúmenes" in resultado:
        print("Publishing message to topic",topic_luz)
        client.publish(topic_luz,resultado,retain=True,qos=1)
```

4.5.3 Resultado

A través del cmd de Windows y suscribiéndonos a alguno de los canales, podemos comprobar que los datos se están publicando correctamente.

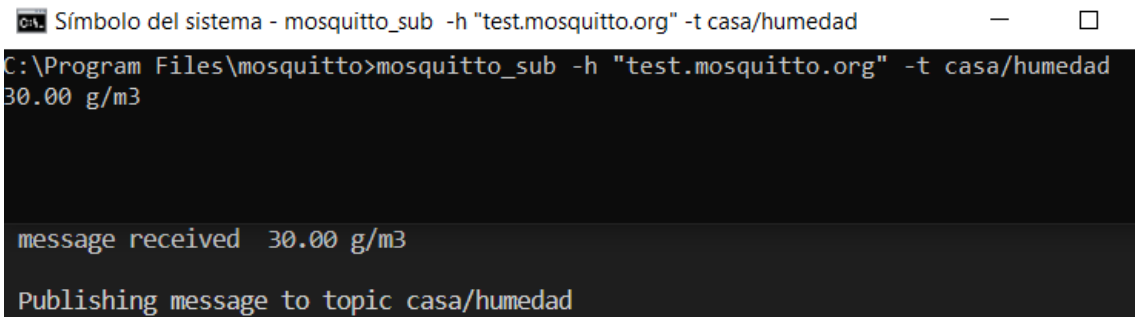

Ilustración 39. Comprobación del funcionamiento del programa

4.6 Programa para Instagram

En el caso de Instagram, el objetivo es que nuestro bot sea capaz de contestar a los mensajes directos de los usuarios, en caso de que nos pregunten por alguno de los datos que reciben nuestros sensores, y si no es así los ignore.

4.6.1 Crear cuenta

Antes de comenzar a escribir el código, debemos crear una cuenta para nuestro bot en esta red social. Accediendo a la misma página [20] nos da la opción de registrarnos.

Ilustración 40. Registrarse en Instagram

Únicamente nos piden un correo eléctrico, un nombre usuario y una contraseña. El nombre de usuario de nuestro bot será *insta_sensores.* Después de rellenar los datos podremos acceder a nuestra cuenta.

Ilustración 41. Cuenta de Instagram

4.6.2 Librería instabot

La librería principal de nuestro programa será Instabot [21], la cual nos permite interactuar con los usuarios a través de esta red social, aunque también haremos uso de la librería paho, en concreto de los métodos que nos permiten crear el cliente y suscribirnos a los topics. Por último, utilizaremos la librería time para detener la ejecución del programa 10 minutos después del envío de cada mensaje, ya que es el

tiempo mínimo que debe pasar entre cada respuesta del bot. Además, añadiremos el archivo de configuración con todas las variables que son necesarias como el usuario y contraseña de nuestra cuenta de Instagram.

```
import time
import paho.mqtt.client as mqtt
import paho.mqtt.subscribe as subscribe
from instabot import Bot #importamos el módulo
from password import *
```

4.6.3 Implementación

Al igual que en el programa anterior, crearemos el cliente MQTT. Seguidamente instanciaremos nuestro bot e iniciaremos sesión con nuestras credenciales.

```
instasensores = Bot () #Nombramos al bot  como instasensores
instasensores.login(username=usuario_insta, password=pssw) #Nos logeamos en
nuestra cuenta de Instagram con nuestras credenciales
```

Como hemos comentado antes, el objetivo es responder a los mensajes directos. Para ello, debemos comprobar que nuestra bandeja de entrada no está vacía y que además los mensajes que hemos recibido son de tipo texto, es decir, que no son emoticonos o contenido multimedia. También comprobaremos que el mensaje que estamos leyendo no lo hayamos enviado nosotros, esto lo haremos teniendo en cuenta el id de nuestro usuario.

```
while  True:
    if instasensores.api.get_inbox_v2():
        data = instasensores.last_json["inbox"]["threads"]
        for item in data:
            last_item = item["last_permanent_item"] ##Guardamos el ultimo mensaje
            thread_item = item ["thread_id"] ##Guardamos el id de la conversacion
            item_type = last_item["item_type"] ##item_type es un atributo de item que
indica si el mensaje es o no de texto

            if item_type == "text": ##Si el mensaje es de tipo texto
                print(last_item["text"])
                msj = last_item["text"]
                usuarioid = last_item ["user_id"] ##Guardamos el id del usuario que
ha mandado el mensaje
                if usuarioid != id: ##Si el mensaje no lo ha mandado el bot
```

Cuando ya hemos comprobado que el mensaje no lo ha mandado el bot, contestaremos al usuario que ha hecho la petición analizándola para ver que dato nos pide. En cada uno de los casos nos suscribiremos al canal correspondiente, decodificaremos el mensaje y lo mandaremos con el método *send_mensaje* que nos proporciona la librería instabot, indicando el usuario, el mensaje y el id de la conversación. Una vez enviado detendremos la ejecución 10 minutos antes de continuar con el programa.

```
if "decibelios" or "sonido" or "Decibelios" or "Sonido" or "ruido" or "Ruido"
in msj: ##Opciones
      x = subscribe.simple(topic_sonido, hostname= "test.mosquitto.org") ##Nos
      suscribimo al canal
      print("%s : %s" % (x.topic, x.payload.decode("utf-8")))
      msj = x.payload.decode("utf-8")
      instasensores.send_message(msj, usu,thread_item)
      time.sleep(600) #Esperamos 10 min
```

4.6.4 Resultado

Ilustración 42. Resultado Instagram

4.7 Programa para Twitter

En Twitter, el objetivo es que nuestro bot sea capaz de publicar *tweets* informando de los valores que detectan los sensores cada día.

4.7.1 Crear cuenta

Para comenzar, al igual que en Instagram, nos tenemos que crear una cuenta. Accediendo a la página de Twitter [22] nos permiten registrarnos de diferentes formas. En nuestro caso lo haremos con Google, ya que lo único que necesitamos es un correo electrónico.

Ilustración 43. Crear cuenta Twitter

El nombre de usuario será el que vemos a continuación.

Ilustración 44. Usuario Twitter

4.7.2 Obtención de claves

La librería que usaremos en este caso será Tweepy. Esta, a diferencia de la de Instagram, no nos permite iniciar sesión con nuestras credenciales, si no que necesitamos unas claves que obtendremos de la página oficial de desarrolladores de Twitter [23].

Para ello, iniciaremos sesión en dicha página con la cuenta que hemos creado en el paso anterior. Lo primero que nos encontramos es la siguiente pantalla, en la que nos piden que creemos una aplicación y le asociemos un nombre.

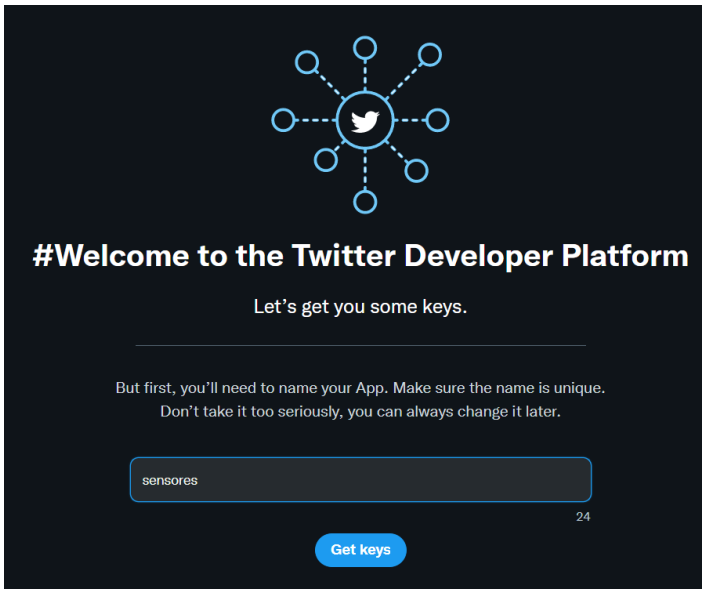

Ilustración 45. Crear API ed Twitter

Después de crear la aplicación, nos aparecerá una pantalla emergente en la que nos proporcionan las claves que necesitamos para nuestro programa.

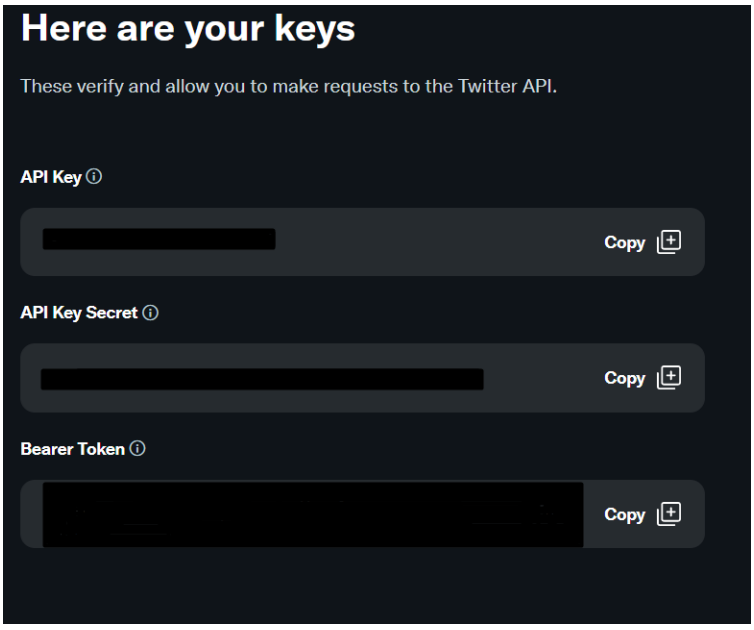

Ilustración 46. Claves de Twitter

Sin embargo, únicamente con estas claves no podemos hacer publicaciones en esta red social. Esto se debe a que solo tenemos acceso esencial en nuestro proyecto. Podemos verlo si accedemos al apartado *Twitter API v2* del menú de la izquierda.

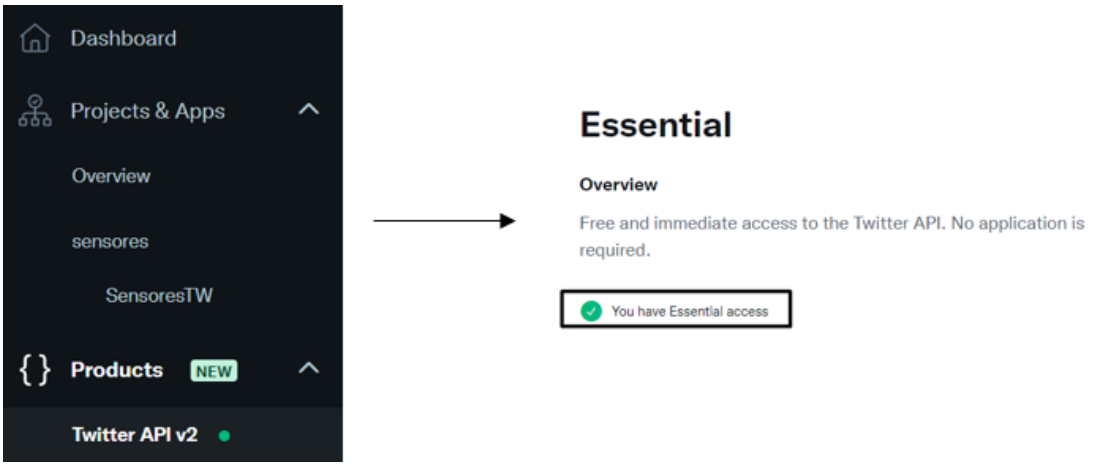

Ilustración 47. Propiedades de la API

Para poder publicar *tweets* necesitamos tener acceso elevado. Accediendo al proyecto nos permiten solicitarlo rellenando un formulario en el que debemos indicar, entre otras cosas, para que vamos a utilizar esta api, como vamos a analizar los datos de Twitter y con qué finalidad.

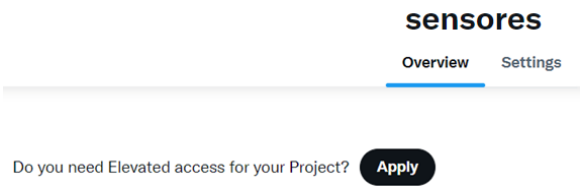

Ilustración 48. Pedir acceso elevado API

4.7.3 Librería Tweepy

Ya hemos mencionado en el paso anterior la librería principal de nuestro programa, la librería Tweepy [24]. Esta nos permite realizar publicaciones en Twitter, aunque al igual que en el resto de aplicaciones, también haremos uso de la librería paho, para crear el cliente MQTT y suscribirnos a los canales. Por último, utilizaremos la librería time para detener la ejecución del programa durante un día una vez se hayan enviado todos los tweets y el archivo de configuración con todas las variables necesarias.

```
import time
import tweepy
import paho.mqtt.client as mqtt
import paho.mqtt.subscribe as subscribe
from password import *
```

4.7.4 Implementación

Lo primero de todo, será crear el cliente y conectarlo. Seguido de esto, inicializaremos cuatro variables con las claves que hemos obtenido de la página de desarrolladores de Twitter e instanciaremos el objeto *handler*.

```
###Ponemos las claves de nuestro usuario para poder acceder a nuestra cuenta
consumer_key = consumer_key
consumer_secret = consumer_secret
access_token = access_token
access_token_secret = access_token_secret

###Instanciamos el objeto handler
auth= tweepy.OAuthHandler(consumer_key,consumer_secret)
auth.set_access_token(access_token, access_token_secret)
```

Posteriormente, cargaremos estas claves en una variable utilizando el método *tweepy.API()* con los siguientes parámetros.

- Auth: claves de la página web de desarrolladores de Twitter
- Wait_on_rate_limit: esto hace que, si llegamos al límite de tweets, el programa no se caiga, simplemente espere a que este se actualice y pueda seguir enviando datos.

```
api = tweepy.API(auth, wait_on_rate_limit=True)

data = api.verify_credentials()
```

Después de esto, nos suscribiremos a los canales y recogeremos el dato.

Para publicarlo, utilizaremos el método *update_status()*, al que únicamente tendremos que pasarle el mensaje que debe twittear.

```
x = subscribe.simple(topic_movimiento, hostname= "test.mosquitto.org")
print("%s : %s" % (x.topic, x.payload.decode("utf-8")))
msj = x.payload.decode("utf-8")
api.update_status(msj)
```

4.7.5 Resultado

Para ver el funcionamiento, accedemos al perfil de nuestro bot y vemos que los tweets se han publicado correctamente.

Ilustración 49. Resultado Twitter

4.8 Programa Telegram

4.8.1 Crear cuenta

Para registrarnos en esta red social únicamente necesitaremos un número de teléfono, el cual debemos introducir al acceder a su página web [25].

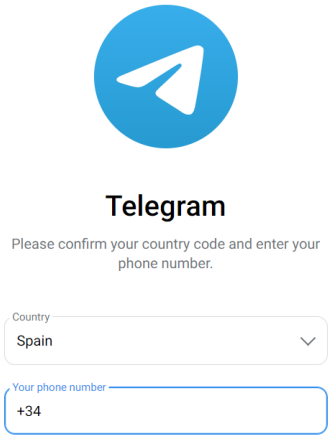

Ilustración 50. Crear cuenta Telegram

El nombre de usuario lo cambiaremos una vez dentro de la aplicación, aunque en este caso como estoy utilizando mi número personal no lo modificaremos. En la siguiente imagen vemos donde se cambiaría en caso de ser necesario.

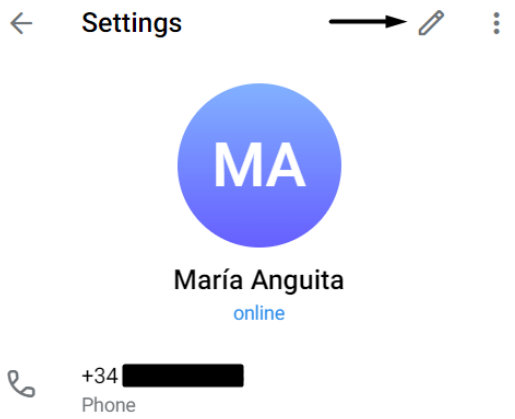

Ilustración 51. Usuario Telegram

4.8.2 Obtención de claves

De la misma forma que en Twitter, necesitamos una serie de claves para poder crear nuestro bot en Telegram. Para conseguirlas debemos acceder a la página web de desarrolladores [26] de esta red social utilizando el mismo número de teléfono que proporcionamos a la hora de crearnos la cuenta.

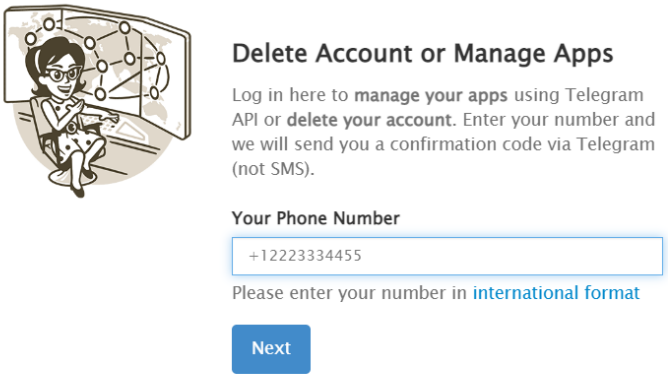

Ilustración 52. Ver mis APIs Telegram

Al darle a siguiente, nos pedirán un código que nos han enviado a nuestra cuenta.

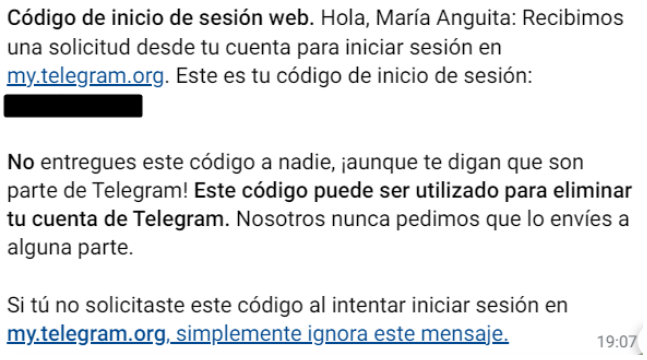

Ilustración 53. Código inicio de sesión

Introducimos el código y nos dirigimos a un formulario en el debemos rellenar varios datos para poder obtener las claves.

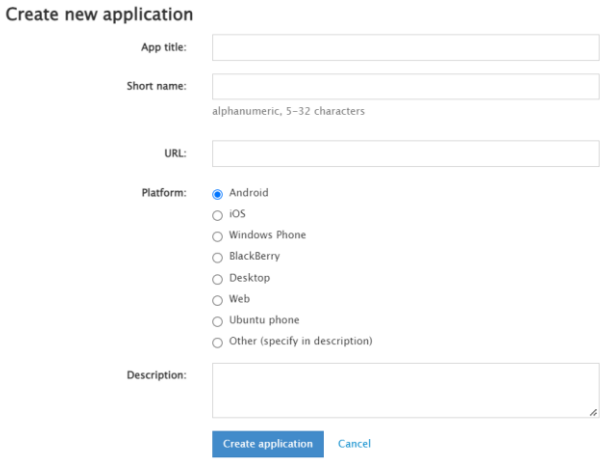

Ilustración 54. Crear API de Telegram

Una vez creada la aplicación, ya nos dan acceso a dichas claves.

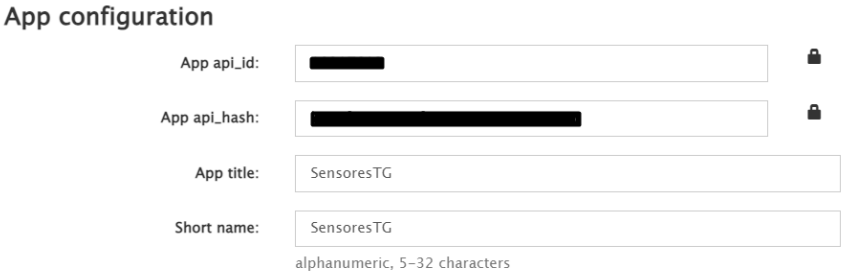

Ilustración 55. Claves de Telegram

4.8.3 Librería Telethon

Esta librería [27] nos permite interactuar a través de Telegram de una manera muy sencilla. Además de esta, utilizaremos la misma que en el resto de apartados, la

librería paho e importaremos el archivo de configuración que contiene todas las librerías necesarias.

```
import paho.mqtt.client as mqtt
import paho.mqtt.subscribe as subscribe
from telethon import TelegramClient, Button, events
from password import *
```

4.8.4 Implementación

Además de crear el cliente MQTT como hemos hecho para todas las redes sociales, también tenemos que instanciar el cliente de telegram con las claves y el nombre de la aplicación que hemos obtenido previamente.

```
api_id = api_id
api_hash = api_hash

TelegramClient = TelegramClient('SensoresTG', api_id, api_hash)
```

Al ejecutar el programa la primera vez para ver que nos conectamos correctamente a la aplicación, nos encontramos con un mensaje en el que nos piden que ingresemos el número de teléfono o un *bot token*. Esto solo ocurre la primera vez, ya que es para autorizar al ordenador a entrar en nuestra cuenta de Telegram.

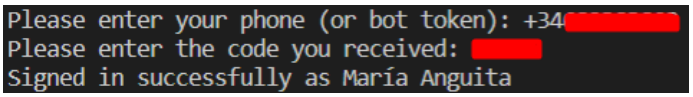

Ilustración 56. Autorización Telegram

El bot_token, podemos obtenerlo si buscamos en Google al usuario *BotFather* [28], es un bot que controla el resto de bots y que es capaz de crearlos por cuenta propia. Para crear un bot a través de él, simplemente tenemos que utilizar el comando */newbot* y seguir las instrucciones.

En la siguiente imagen podemos ver el proceso entero y como al final, nos devuelve una clave, el *bot token*.

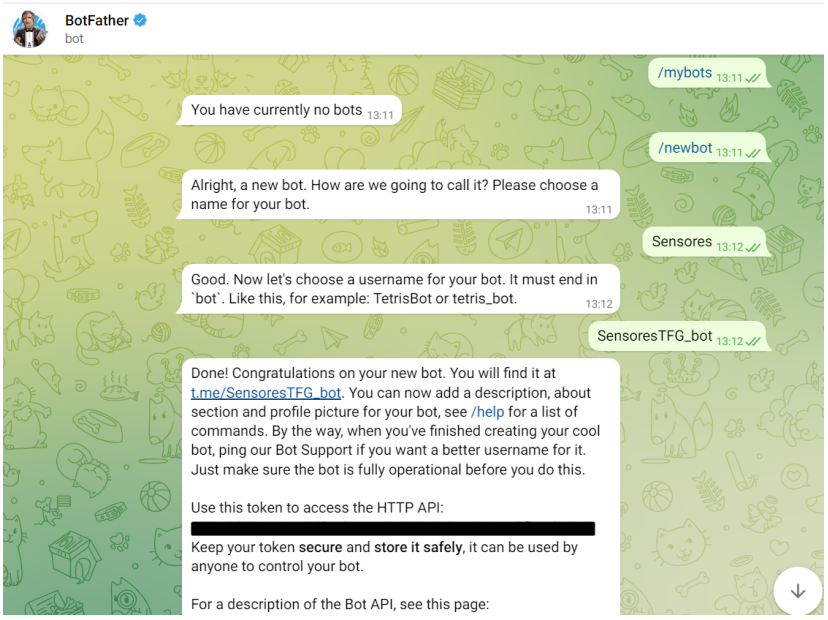

Ilustración 57. Bot de Telegram

Para programar el bot vamos a utilizar la opción de crear botones, de esta manera no deberemos tener en cuenta el mensaje que recibimos por parte del usuario. El método button.inline, nos permite crearlos y establecer el texto que llevara cada uno de ellos, así como el orden.

```
@TelegramClient.on(events.NewMessage(pattern=""))##Mande el mensaje que mande el usuario la respuesta será
async def handler(event):

    keyboard = [
        [
            Button.inline("Temperatura", b"1"),
            Button.inline("Movimiento", b"2")
        ],
        [
            Button.inline("Sonido", b"3"),
            Button.inline("Luz", b"4")
        ]
    ]

    await TelegramClient.send_message(event.chat_id, "Elige que dato quieres saber:", buttons=keyboard)
```

Después, debemos comprobar cuál ha sido el botón que se ha activado, suscribirnos al canal correspondiente y con la función *await.edit* mandar el dato. Esta suspende la ejecución hasta que se ha devuelto el valor esperado y luego continua con el bucle. Además, sustituye el mensaje de los botones por el dato, de esta manera evitamos mantener los botones a lo largo de todo el chat.

```
@TelegramClient.on(events.CallbackQuery)
async def callback(event):
    if event.data == b'1' :
        x = subscribe.simple(topic_temperatura, hostname=
"test.mosquitto.org")
        print("%s : %s" % (x.topic, x.payload.decode("utf-8")))
        msj = x.payload.decode("utf-8")
        await event.edit(msj)
```

4.8.5 Resultado

Como podemos ver en la imagen, en el momento en el que recibamos un mensaje de algún usuario, sea el mensaje que sea, el bot responderá con los botones.

Ilustración 58. Resultado Telegram 1

En la siguiente imagen vemos como es sustituido el mensaje anterior por el dato que ha seleccionado el usuario.

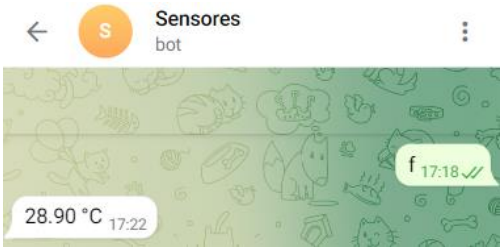

Ilustración 59. Resultado Telegram 2

4.9 Programa para Discord

En cuanto a Discord, vamos a crear nuestro bot y lo añadiremos a un servidor. El objetivo es que este, sea capaz de contestar a las peticiones de los usuarios que se encuentran en dicho servidor.

4.9.1 Crear cuenta

Antes de comenzar a programar el bot, debemos crearle una cuenta en esta red social. Podemos hacerlo entrando a la página de Discord a través de un navegador [29] o descargándonos la aplicación.

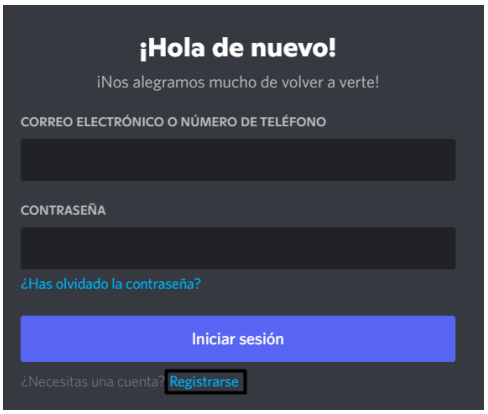

Ilustración 60. Registrarse en Discord

En este caso, nos piden un correo electrónico, la fecha de nacimiento y un nombre de usuario, *sensores*, con su contraseña. Después de rellenar los datos podemos iniciar sesión en nuestra cuenta.

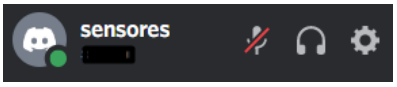

Ilustración 61. Cuenta de Discord

4.9.2 Crear servidor

Una vez dentro de nuestra cuenta, podemos pasar a crear el servidor en el que vamos a añadir nuestro bot. Únicamente debemos dirigirnos al menú de la izquierda y darle al más.

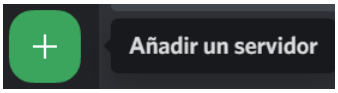

Ilustración 62. Crear servidor Discord

Para crearlo nos piden que seleccionemos la plantilla que vamos a utilizar o que creemos la nuestra propia, que es la opción que vamos a elegir en nuestro caso.

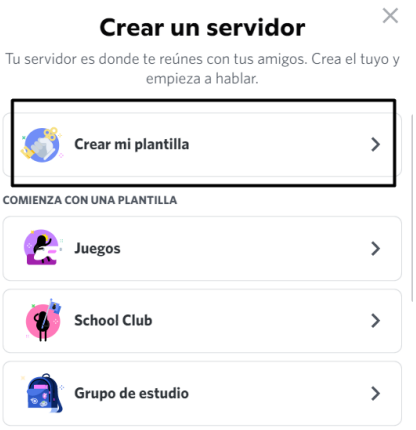

Ilustración 63. Configurar servidor Discord 1

Posteriormente elegiremos hacia quien va dedicado nuestro servidor. Nosotros queremos que sea un servidor abierto no únicamente para nuestros amigos por lo que elegiremos la primera opción de las que vemos en la siguiente ilustración.

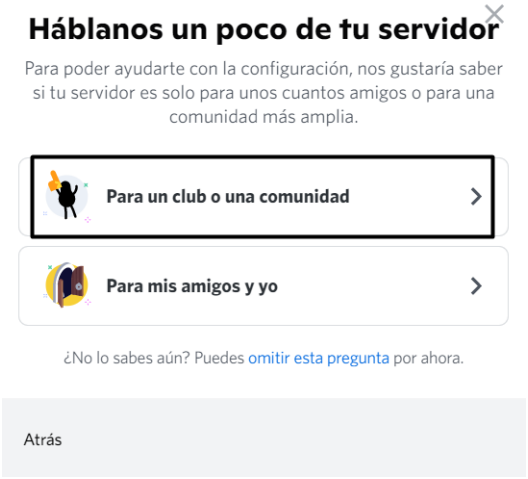

Ilustración 64. Configurar servidor Discord 2

Establecemos el nombre y ya tenemos nuestro servidor. Ahora debemos crear el bot para añadirlo. Para ello, accedemos a la página de desarrolladores de Discord [30] y creamos una nueva aplicación con el nombre que deseemos, tal y como vemos en la siguiente imagen.

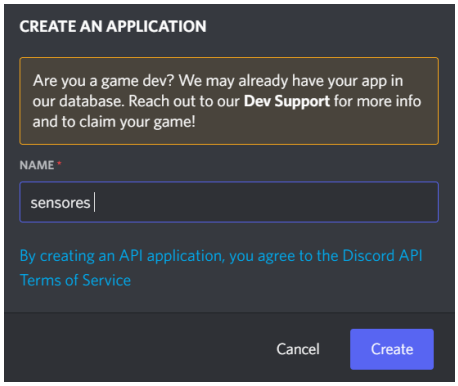

Ilustración 65. Crear API Discord

Dentro de la misma, en el menú de la izquierda accedemos al apartado *bot* y lo añadimos a nuestra aplicación.

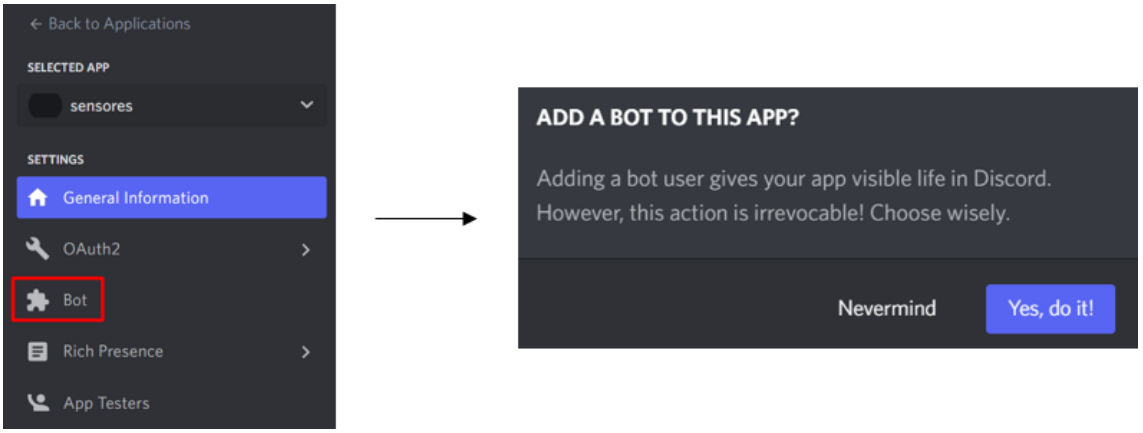

Ilustración 66. Añadir bot a nuestra API

Una vez creado, debemos añadirlo a nuestro servidor. Para esto necesitamos una *URL*, la cual generaremos como vemos en las siguientes imágenes. Accediendo al menú *URL generator*, seleccionando *bot* y dándole los permisos que deseemos, en nuestro caso los de administrador.

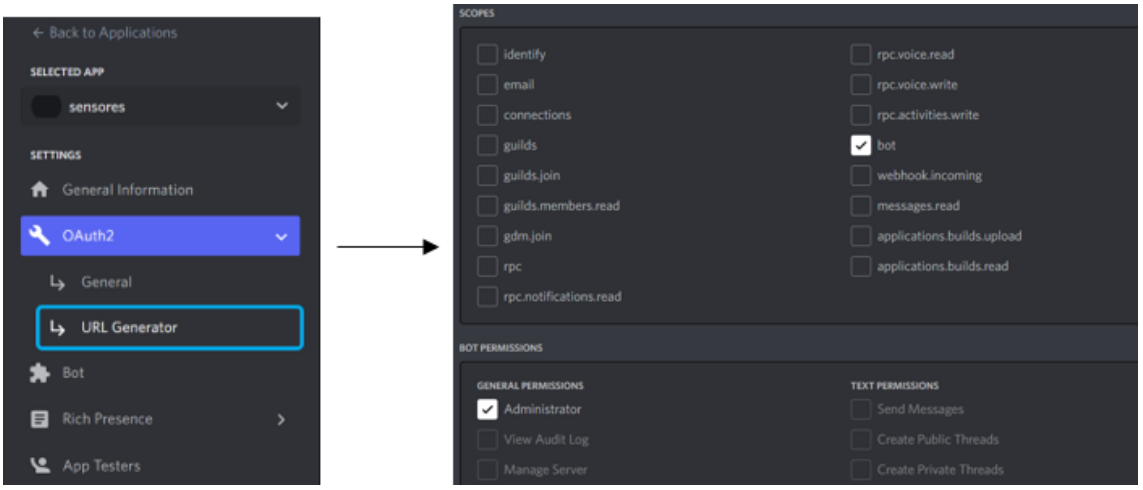

Ilustración 67. Generar URL

Después de esto, podemos acceder a la *URL* y añadir nuestro bot al servidor que habíamos creado en el paso anterior.

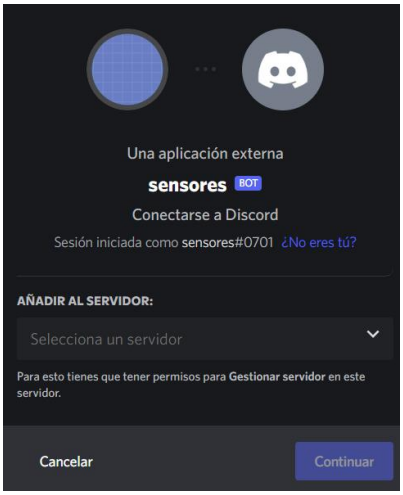

Ilustración 68. Añadir bot al servidor

Finalizados todos los pasos, podemos comprobar en nuestro servidor que se ha añadido correctamente. Esto lo podemos ver en la siguiente imagen.

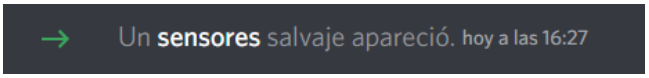

Ilustración 69. Resultado al añadir el bot

Sin embargo, aún nos queda el paso más importante, hacer que nuestro servidor sea público para que todo el mundo tenga acceso. Para conseguir esto, nos dirigimos haciendo *click* con el botón derecho en nuestro servidor a los ajustes de comunidades y solicitamos iniciar nuestra comunidad.

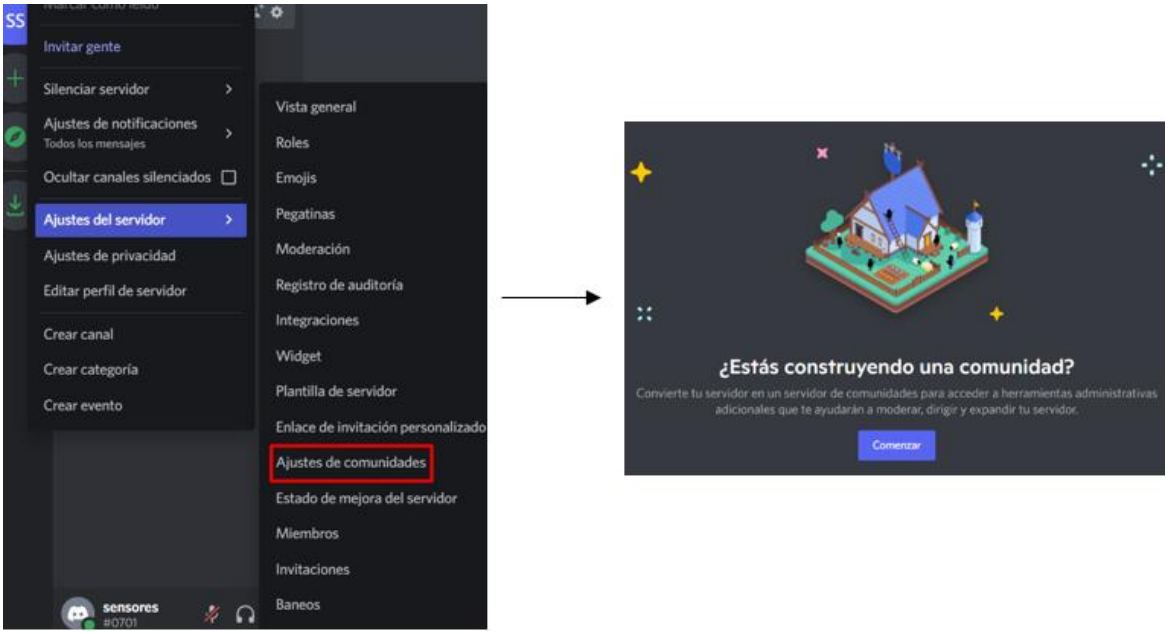

Ilustración 70. Crear comunidad en Discord

A continuación, seguimos los pasos que nos indican estableciendo la configuración de nuestra comunidad. El primer paso es la solicitar la verificación de los usuarios que quieran unirse a ella a través de un correo electrónico y permitir que Discord filtre los mensajes multimedia que manden los miembros.

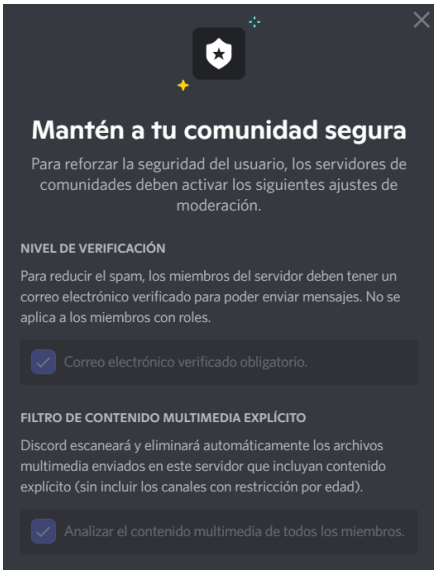

Ilustración 71. Configuración de la comunidad 1

En el siguiente paso, nos piden que creemos un canal donde explicaremos las normas de nuestro servidor, si no queremos crear uno nuevo podemos hacemos en el general.

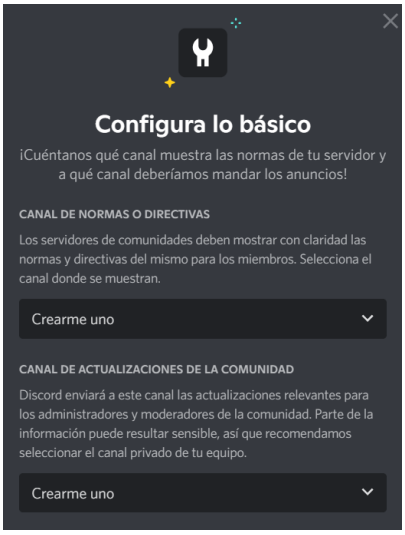

Ilustración 72. Configuración de la comunidad 2

Por último, aceptamos los términos y condiciones.

Esto que hemos hecho simplemente sirve para crear una comunidad, ahora debemos hacerla pública. Una vez finalizados todos estos pasos nos aparece la siguiente pantalla en la que nos indican si queremos entrar en *descubrimiento*, que es la herramienta en la que podemos encontrar todos los servidores públicos.

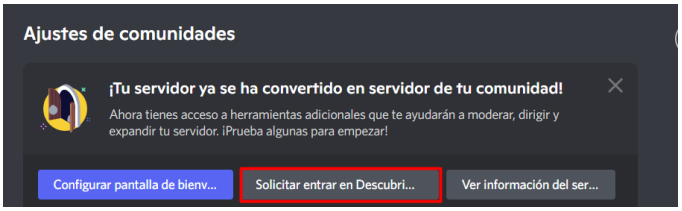

Ilustración 73. Hacer servidor público

Sin embargo, al solicitarlo nos indican que nuestro servidor no cumple todos los requisitos para hacerlo público. Entre otros, no tener más de mil miembros en nuestro servidor.

Ilustración 74. Requisitos para hacer público el servidor

Debido a esto, no podemos continuar con la herramienta de Discord, ya que en nuestro proyecto es indispensable que todo el mundo pueda solicitar información a nuestro bot.

Presupuesto

En este apartado vamos a analizar el presupuesto del proyecto dividiéndolo en tres apartados. Iniciando con un análisis de las horas dedicadas a cada parte del trabajo y continuando con el coste en horas de trabajo y el coste en hardware.

5.1 Planificación del proyecto

En cuanto a las horas de trabajo, debemos tener en cuenta tanto las horas empleadas en recopilar documentación, como las dedicadas a la implementación del proyecto. En total, podemos asumir una duración aproximada de seis meses de trabajo. En la siguiente tabla podemos ver la planificación del proyecto.

Actividad	**Duración (horas)**
Documentación	90
Implementación	280
Análisis de resultados	120
Redacción	120
TOTAL	610

5.2 Coste de mano de obra

Suponemos que el proyecto ha sido realizado por un ingeniero. Con estos datos, se estima que el coste en mano de obra sea un total de 15.250€.

Mano de obra	**Duración (horas)**	**Coste (€/hora)**	**Coste total (€)**
Ingeniero	610	25	15.250

5.3 Coste en hardware

Para realizar este proyecto necesitamos distintos elementos hardware, desde un ordenador portátil hasta los sensores de Arduino. Podemos ver el coste de cada uno de estos elementos en la siguiente tabla.

Dispositivo	**Coste (€)**
Ordenador portátil MSI	859
Sensor de movimiento	1,24
Sensor de sonido	0,43

Sensor de temperatura y humedad	0,87
Sensor de luz	0,32
Led RGB	0,61
Cables	2,01
Placa protoboard	7,99
Placa de arduino nano	4,75
PCB	4,85
Soldador	14,49
TOTAL	896,56

5.4 Coste total

Una vez desglosado el coste de cada una de las partes del proyecto, podemos concluir que el coste total de este trabajo ha sido de 16.146,56€.

Proyecto	**Coste (€)**
Coste mano de obra	15.250
Coste de materiales	896,56
Coste total	16.146,56

Conclusiones

Este trabajo se ha centrado en la implementación de un sistema IoT capaz de interactuar con los usuarios a través de las redes sociales, utilizando herramientas simples y gratuitas al alcance de todo el mundo.

Una vez finalizado, podemos sacar varias conclusiones. La primera de ellas, es la importancia que tienen en nuestra vida diaria los sistemas inteligentes. Cualquier dispositivo que utilicemos en nuestro día a día es capaz de conectarse a internet para intercambiar datos con otros sistemas.

También hemos podido ver como los bots, algo que estaba muy mal visto hace no muchos años, son capaces de automatizar tareas necesarias ya que muchas veces, para los humanos es imposible abarcar tal número de solicitudes. Actualmente, los bots empiezan a verse como una herramienta de ayuda. Por esto, hemos podido encontrar numerosos artículos con información y muchas librerías que nos han sido útiles a la hora de realizar el proyecto.

Aun así, plataformas como Discord siguen poniéndonos trabas a la hora de crear un servidor público con nuestro propio bot, aunque si nos permiten utilizar bots que ellos mismos han creado.

Además, hemos podido comprobar la relevancia que tienen hoy en día las redes sociales. Esto se debe a lo intuitivas y fáciles que son de utilizar para el usuario, además de lo sencillo que es crearse una cuenta en cualquiera de ellas.

Posiblemente, la parte con mayor dificultad ha sido encontrar la solución a muchos de los errores que nos daban los programas al usar las librerías de los bots, ya que podíamos ver muchos usuarios con los mismos problemas, pero sin ninguna solución.

Otro de los problemas que nos ha surgido, ha sido para encontrar la manera de acceder a Twitter, ya que la versión de la api ha cambiado hace relativamente poco y la información acerca de esto era escasa y ambigua.

6. Bibliografía

[1] Pleva GmbH, "DHT11 Humidity & Temperature Sensor," *Melliand Textilberichte*, vol. 76, no. 12, p. 1112, 1995.

[2] "(PDF) HC-SR501 Datasheet - PIR MOTION DETECTOR." http://www.datasheet.es/PDF/775434/HC-SR501-pdf.html (accessed Jun. 15, 2022).

[3] "Sensor de sonido | Tienda y Tutoriales Arduino." https://www.prometec.net/sensor-sonido-led-s4a/# (accessed Jun. 13, 2022).

[4] "UNIT Electronics Módulo KY-018 Sensor Foto Resistor." https://uelectronics.com/producto/modulo-ky-018-sensor-foto-resistor/ (accessed Jun. 13, 2022).

[5] "Comparación Arduino Uno – Arduino Nano | Robots Didácticos." https://robots-argentina.com.ar/didactica/comparacion-arduino-uno-arduino-nano/ (accessed Jun. 13, 2022).

[6] "PCB significado ¿Qué es un PCB y para qué sirve? – Altium." https://resources.altium.com/es/p/what-is-a-pcb (accessed Jun. 13, 2022).

[7] "¿Qué es MQTT? Su importancia como protocolo IoT." https://www.luisllamas.es/que-es-mqtt-su-importancia-como-protocolo-iot/ (accessed May 29, 2022).

[9] "MQTT — Part I: Understanding MQTT | by Onur Dündar | Medium." https://medium.com/@onur.dundar1/mqtt-part-i-understanding-mqtt-aade455baec9 (accessed Jun. 13, 2022).

[10] "MQTT vs CoAP, the battle to become the best IoT protocol." https://www.pickdata.net/es/noticias/mqtt-vs-coap-mejor-protocolo-iot (accessed Jun. 15, 2022).

[11] "MQTT vs HTTP: ¿qué protocolo es mejor para IoT? – BorrowBits." https://borrowbits.com/2020/04/mqtt-vs-http-que-protocolo-es-mejor-para-iot/ (accessed Jun. 13, 2022).

[12] "paho-mqtt · PyPI." https://pypi.org/project/paho-mqtt/#usage-and-api (accessed Dec. 11, 2021).

[13] "Download Center - EasyEDA." https://easyeda.com/page/download (accessed Jun. 13, 2022).

[14] "Software | Arduino." https://www.arduino.cc/en/software (accessed Jun. 13, 2022).

[15] "Arduino Nano | Arduino.cl - Compra tu Arduino en Línea." https://arduino.cl/arduino-nano/ (accessed Jun. 13, 2022).

[16] “DHT sensor library - Arduino Reference.” https://www.arduino.cc/reference/en/libraries/dht-sensor-library/ (accessed Jun. 13, 2022).

[17] “Download | Eclipse Mosquitto.” https://mosquitto.org/download/ (accessed Jun. 13, 2022).

[18] “Guía de introducción a MQTT con ESP8266 y Raspberry Pi.” https://programarfacil.com/esp8266/mqtt-esp8266-raspberry-pi/ (accessed Jun. 13, 2022).

[19] “Download Visual Studio Code - Mac, Linux, Windows.” https://code.visualstudio.com/download (accessed Jun. 15, 2022).

[20] “Comunicación entre Python 3 y Arduino - Parte 1 (Descarga e instalación de PySerial) | by Cristian Leiton Valencia | Medium.” https://medium.com/@crileiton/comunicación-entre-python-3-7-y-arduino-parte-1-descarga-e-instalación-de-pyserial-daebcf05ebb7 (accessed Jun. 15, 2022).

[21] “Instagram.” https://www.instagram.com/ (accessed Jun. 13, 2022).

[22] “instabot-py · PyPI.” https://pypi.org/project/instabot-py/ (accessed Dec. 11, 2021).

[23] “Twitter. Es lo que está pasando. / Twitter.” https://twitter.com/?lang=es (accessed Jun. 13, 2022).

[24] “Use Cases, Tutorials, & Documentation | Twitter Developer Platform.” https://developer.twitter.com/en (accessed Jun. 13, 2022).

[25] “Tweepy Documentation — tweepy 4.4.0 documentation.” https://docs.tweepy.org/en/stable/ (accessed Dec. 11, 2021).

[26] “Telegram Web.” https://web.telegram.org/z/ (accessed Jun. 13, 2022).

[27] “Authorization.” https://my.telegram.org/auth (accessed Jun. 13, 2022).

[28] “Telethon · PyPI.” https://pypi.org/project/Telethon/ (accessed Dec. 11, 2021).

[29] “Telegram: Contact @botfather.” https://t.me/botfather (accessed Jun. 13, 2022).

[30] “Discord.” https://discord.com/register (accessed Jun. 13, 2022).

[31] “Discord.” https://discord.com/login?redirect_to=%2Fdevelopers%2Fapplications (accessed Jun. 13, 2022).

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá