**Detector**

**de**

**Partículas Nocivas**

**Alvaro Aguayo Orozco**

**Eduardo Jiménez Quintero**

**Antonio Puerto Borreguero**

**Introducción**

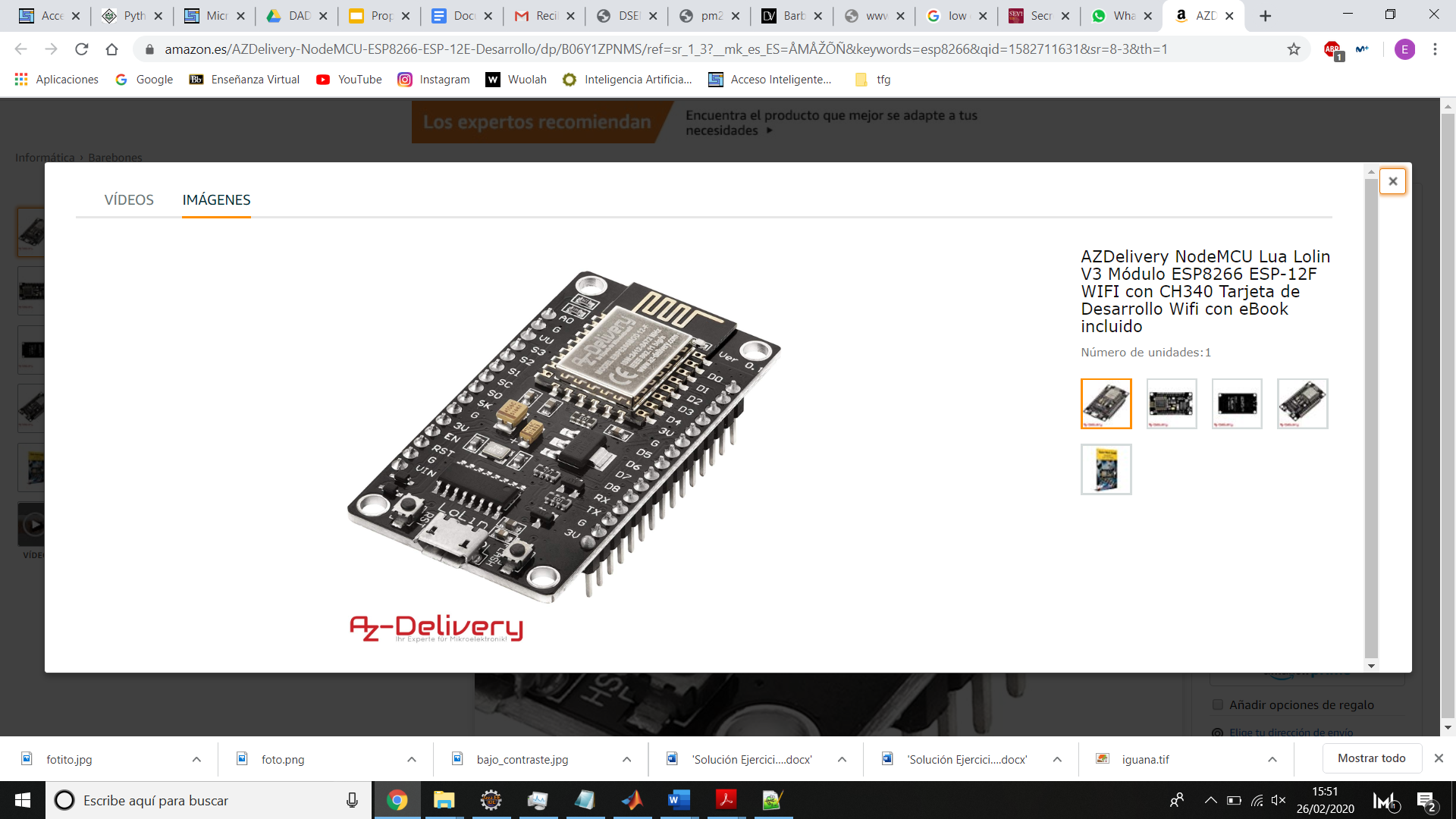
Este proyecto tiene como fin crear un dispositivo que sea capaz de detectar partículas nocivas para los seres humanos y alertar cuando los niveles no sean seguros. Estas partículas nocivas son las partículas PM-10 ( Tamaño <10 µm ) y PM-2,5 (Tamaño <2,5 µm ), estas últimas son las más perjudiciales ya que entran directamente en el torrente sanguíneo.

Para ello utilizaremos un sensor de partículas (PM2.5 Air Quality Sensor) que haga las mediciones y una serie de actuadores ( led o alarmas) que avisen de que hay unos niveles altos y displays que monitorizan los valores a tiempo real.

Para recopilar los datos del sensor utilizaremos el ESP8266. Estos datos pasarán a una base de datos donde un servidor podrá interpretarlos y procesarlos.

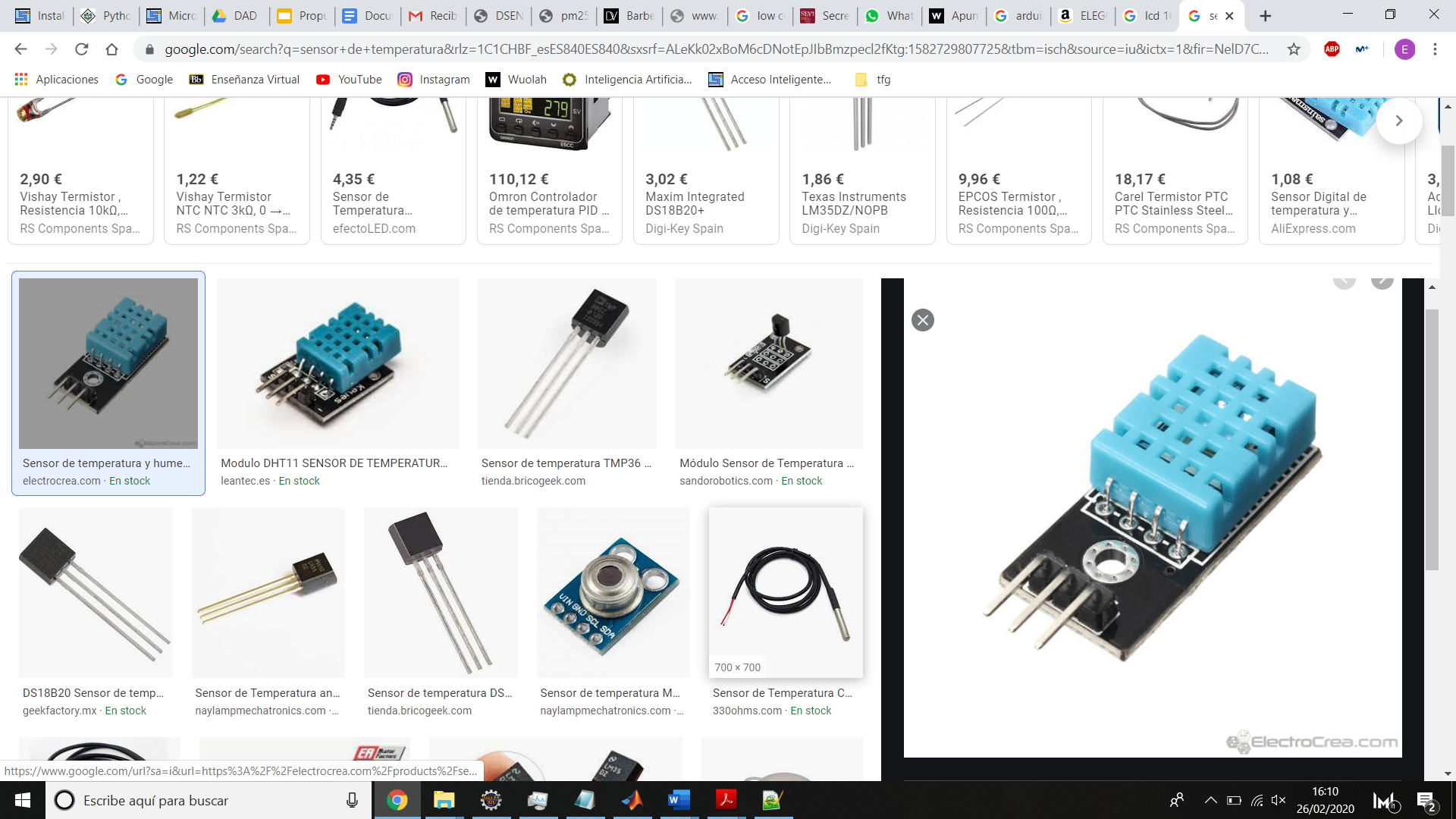
El usuario podrá acceder a estos datos mediante un bot de Telegram con el que podrá interactuar con distintos comandos para ver lo que necesite.

**Componentes Hardware**

**ESP8266 (NodeMCU Lua Lolin V3)**

NodeMCU LoLin V3 es una placa de desarrollo ESP. Equipado con un potente procesador ESP8266-12F de Expressif y WLAN integrado, este controlador ofrece una fácil entrada en el desarrollo IoT . Esta tarjeta es programable en LUA a través del ArduinoIDE con C, así como a través del Espressif-SDK. El chipset que se utilizó es el CH340.

Explicación y Datasheet: [NodeMCU, la popular placa de desarrollo con ESP8266](https://www.luisllamas.es/esp8266-nodemcu/)

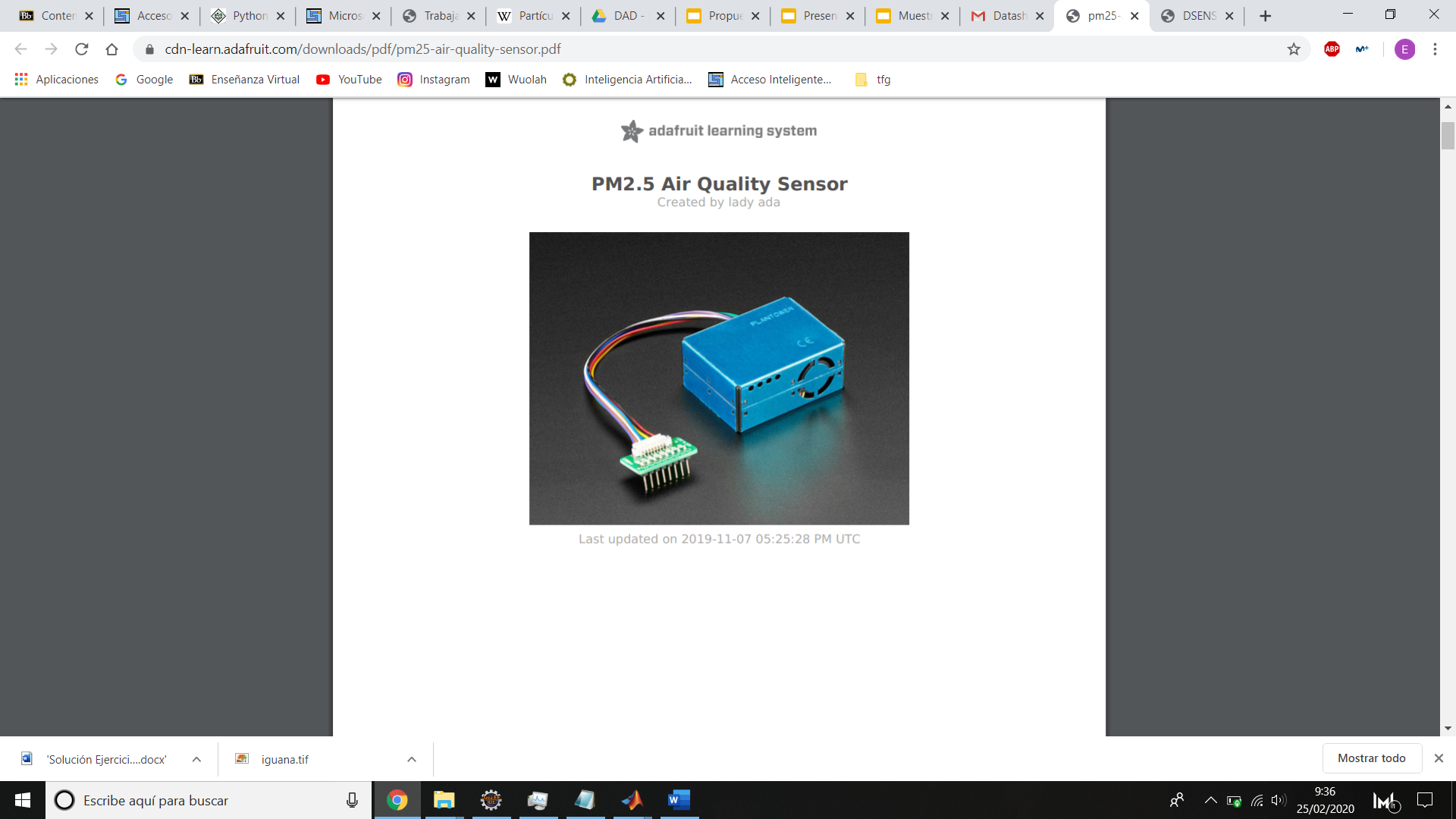
**DHT11** 

Es un sensor de temperatura y humedad el cual dispone de una salida digital calibrada. Compatible con sistemas electrónicos operando entre 3v-5v y de corriente máxima de 2.5 mA.

Valores recogidos:

* Humedad relativa: 0-80% (±5%)
* Temperatura: 0-50°C (±2°C)
* Tiempo de respuesta: ≈10 segundos

Datasheet: [DHT11 Humidity & Temperature Sensor](https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf)

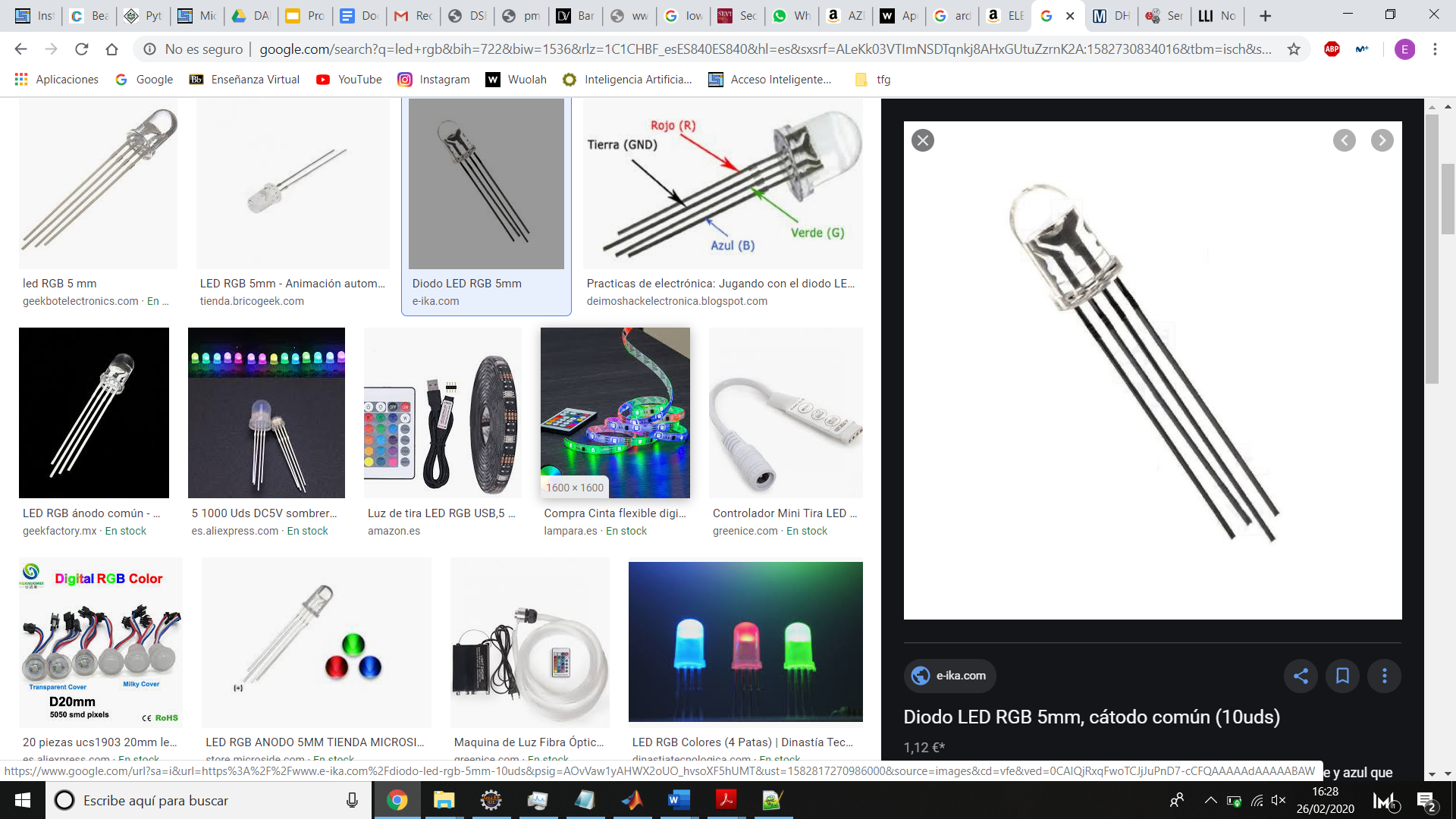
**PM2.5 Air Quality Sensor**

PMS5003 es un tipo de sensor de concentración de partículas universal y digital, que puede usarse para obtener la cantidad de partículas suspendidas en el aire, es decir, la concentración de partículas y su salida en forma de interfaz digital.

Funcionamiento:

El principio de dispersión láser se utiliza para dicho sensor, es decir, produce dispersión usando láser para irradiar partículas en suspensión en el aire, luego recoge la dispersión de la luz en cierto grado, y finalmente obtiene la curva de dispersión del cambio de luz con el tiempo. Al final, obtiene el diámetro de partícula equivalente y el número de partículas.

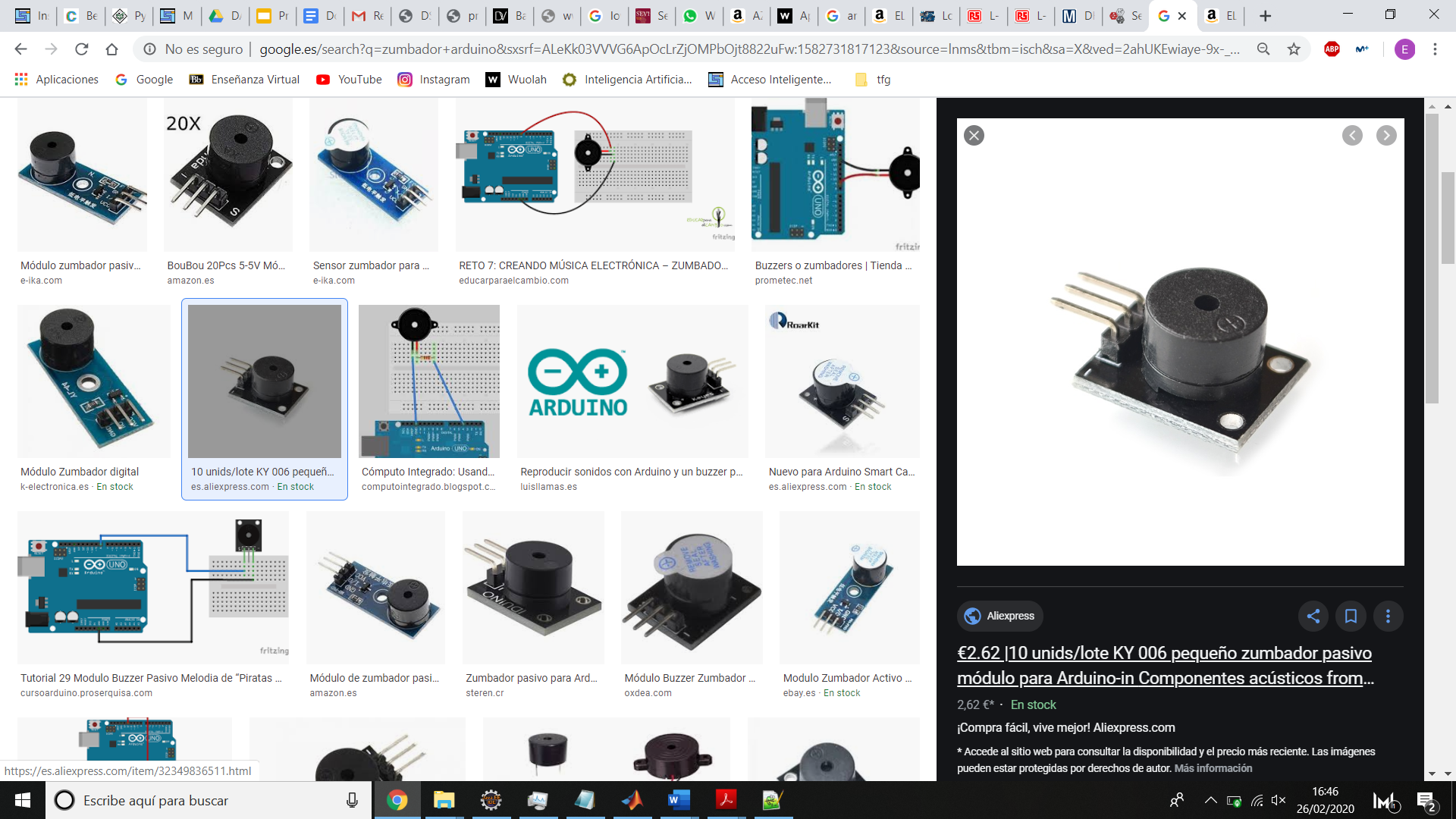
Datasheet: [Digital universal particle concentration sensor](https://cdn-shop.adafruit.com/product-files/3686/plantower-pms5003-manual_v2-3.pdf)

**Led RGB**

Es la unión de tres LEDs de los colores básicos, en un encapsulado común, compartiendo el cátodo

En función de la tensión que pongamos en cada pin podemos conseguir la mezcla de color que deseemos con relativa sencillez.

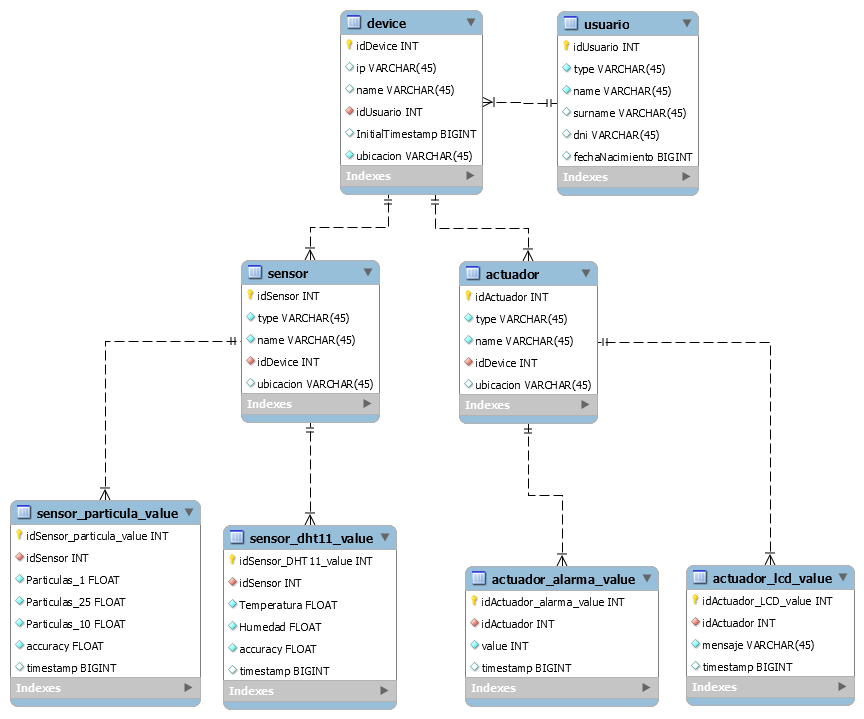
DataSheet:[Kingbright](https://docs.rs-online.com/ad97/0900766b8139d70b.pdf)

**Zumbador**

Un dispositivo que permite convertir una señal eléctrica en una onda de sonido. Estos dispositivos no disponen de electrónica interna, por lo que tenemos que proporcionar una señal eléctrica para conseguir el sonido deseado.

Descripción y Datasheet: [Reproducir sonidos con Arduino y un buzzer pasivo o altavoz](https://www.luisllamas.es/reproducir-sonidos-arduino-buzzer-pasivo-altavoz/)

**Base de datos**

****

Nuestra base de datos consta de las tablas arriba señaladas, cada dispositivo será un actuador o un sensor. Los valores de cada uno se guardan en las tablas acabadas en “value” y estas tablas mediante claves foráneas se identifican como sensor o actuador.

El usuario “dueño” de un dispositivo será el que pueda actuar sobre él, es una implementación futura puesto que ahora no se le da uso.

**API-REST**

La Api-Rest desplegada consta de los métodos para acceder a la base de datos que hemos estimado necesarios para nuestro proyecto.

En nuestra base de datos tenemos distintas tablas de las cuales hemos desplegado métodos delete, put y get según necesitaramos.

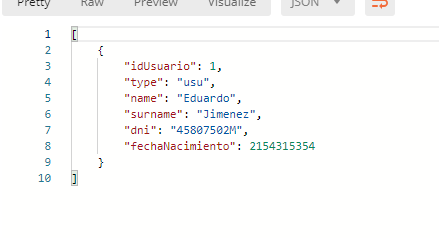
Vamos a explicar algunos métodos:

**Usuario**:

**Get**: Para usar el método get en el caso particular de Usuario necesitamos pegar en Postman o en nuestro navegador la siguiente ruta:

**/api/usuario/:idUsuario**

Intercambiando la ID que queremos por el parámetro **idUsuario** obtenemos lo siguiente:

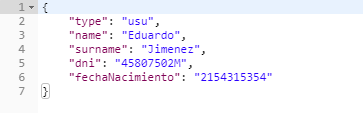


Como vemos en la imagen el servidor nos devuelve un Json con toda la información del usuario con la ID proporcionada que tengamos en la base de datos.

**Put**: Para usar el método put usaremos la siguiente ruta solo en Postman, puesto que le tenemos que pasar también un Json con la información del usuario a crear en la base de datos.

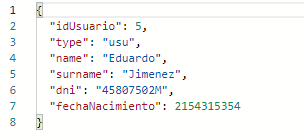
**/api/usuario**

La información a pasarle para crear el usuario tiene que tener el siguiente formato:



Para comprobar que ha funcionado el servidor ha de devolver el mismo Json indicando además la ID del nuevo usuario que se ha creado en la base de datos.

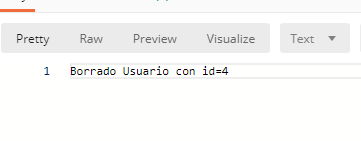
Tal y como se ve en la siguiente imagen:



**Delete:** Para usar el método delete usamos la misma ruta que para el get puesto que vamos a borrar toda la información almacenada del usuario dependiendo de su ID.

El resultado de mandar una petición delete al servidor con la url (**/api/usuario/:idUsuario**)

usando como idUsuario la ID 4 es el siguiente:



Las distintas url para los distintos métodos que tenemos son las siguientes:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **GET** | **PUT** | **DELETE** |
| **Sensor** | /api/sensor/:idSensor | /api/sensor/ | /api/sensor/:idSensor |
| **Actuador** | /api/actuador/:idActuador | /api/actuador/ | /api/actuador/:idActuador |
| **ActuadorAlarmaValue** | /api/actuador/actuadorAlarmaValue/ :idActuador\_alarma\_value | /api/actuador/  actuadorAlarmaValue | /api/actuador/actuadorAlarmaValue/ :idActuador\_alarma\_value |
| **ActuadorLcdValue** | /api/actuador/actuadorLcdValue/  :idActuador\_LCD\_value | /api/actuador/  actuadorLcdValue/ | /api/actuador/actuadorLcdValue/  :idActuador\_LCD\_value |
| **SensorParticulasValue** | /api/sensor/particulas\_values/  :idSensor\_particula\_value | /api/sensor/  particulas\_values/ | /api/sensor/particulas\_values/  :idSensor\_particula\_value |
| **SensorDht11Value** | /api/sensor/dht11\_values/  :idSensor\_dht11\_value | /api/sensor/  dht11\_values/ | /api/sensor/dht11\_values/  :idSensor\_dht11\_value |
| **Device** | /api/device/:idDevice | /api/device/ | /api/device/:idDevice |

En cada petición PUT hemos de recordar no mandarle al servidor la ID de la tabla que queramos crear puesto que es la propia base de datos la que se encarga de crear la ID.

Todo el código de nuestro proyecto se encuentra en el siguiente repositorio:

<https://github.com/alvaguoro/dadParticulas>

Está dividido en varias carpetas donde se encuentran los distintos proyectos, como el proyecto Vertx en Java donde tenemos el servidor creado además del bot de Telegram. En la carpeta MQTT está el software del microcontrolador y por último tenemos la carpeta de Base de datos donde tenemos un archivo para importar la base de datos.

Este proyecto podría ampliarse en un futuro creándose una red con suficientes dispositivos conectados a la misma y mediante Machine Learning procesar la información y desplegarla en una página web además del bot de Telegram, poniendo así a disposición de los usuarios muchísima más información que la actual.