# Practicas DAW

Las prácticas y el proyecto lo organizaré en Sprints de unos 15 días para poder colaborar con el trabajo del tutor de typsa , Javier Urdaneta.

## 

Ilustración 1 Componentes del Equipo inicial

## octubre de 2024

### 14/10/2024.

Comienzo de las prácticas, entrega de equipos y credenciales de acceso a Typsa.

Reunión para conocer a los integrantes del equipo Iría y Javier.

#### Instalación del Arduino IDE .

Instalo en el equipo el ide propio de Arduino para comenzar el desarrollo de rutinas

Al trabajar con ESP-32 como placa añado en archivo preferencias la ruta indicada:

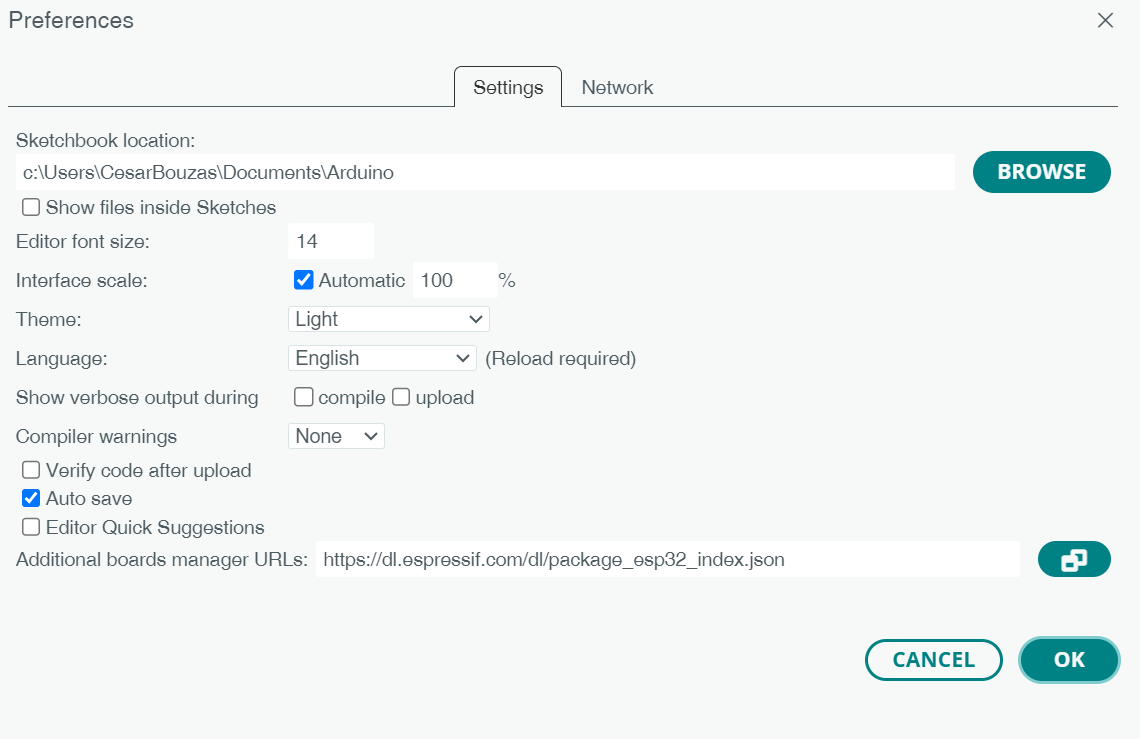


Ilustración 2 Ruta https://dl.espressif.com/dl/package\_esp32\_index.json

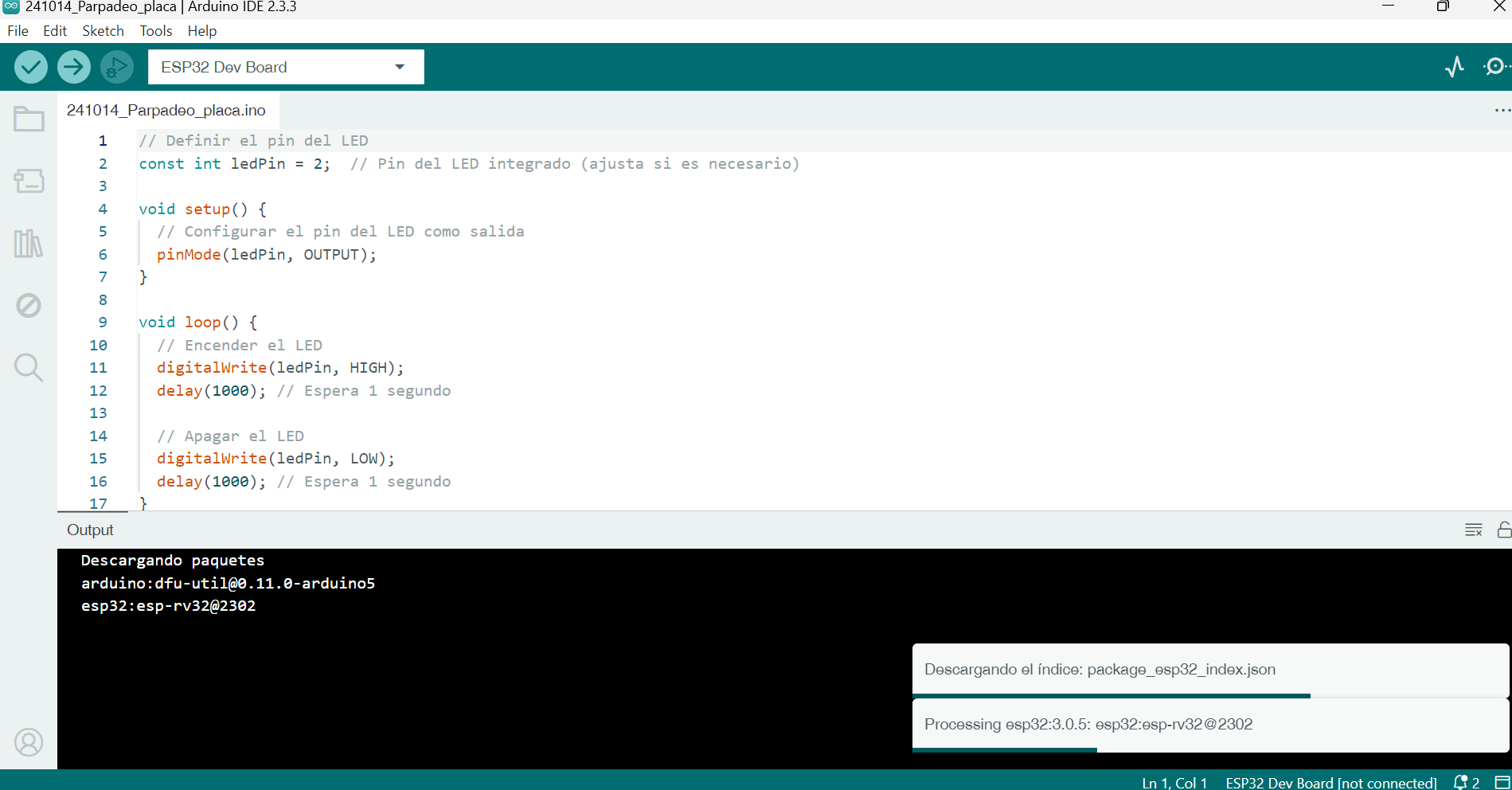


Ilustración 3 Descarga de paquetes

#### Descarga de Drivers para la utilización de puertos USB

Descargamos el controlador:

<https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers>

En dispositivos seleccionamos el puerto COM y LPT que aparece al conectar la placa y actualizamos el controlador con el descargado UART Bridge

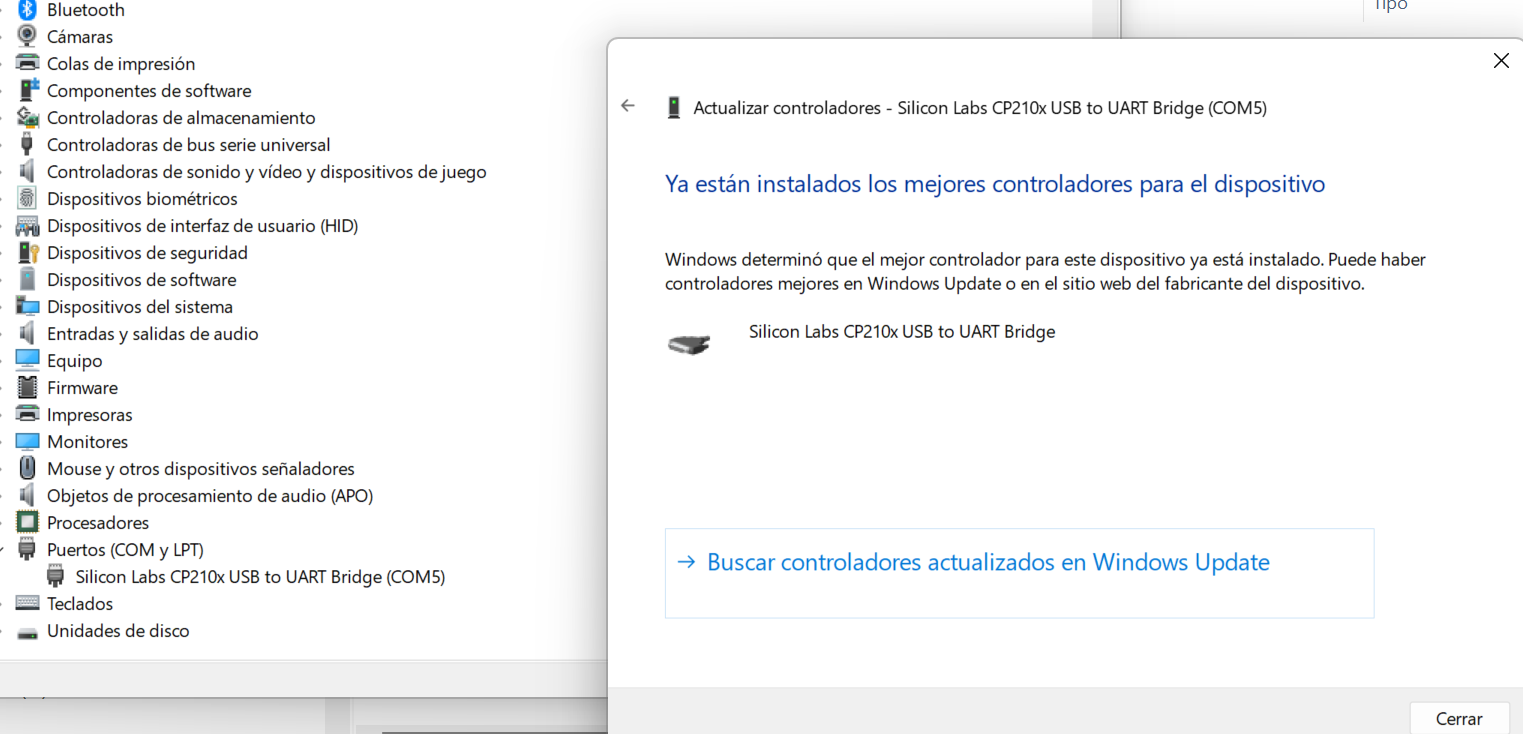
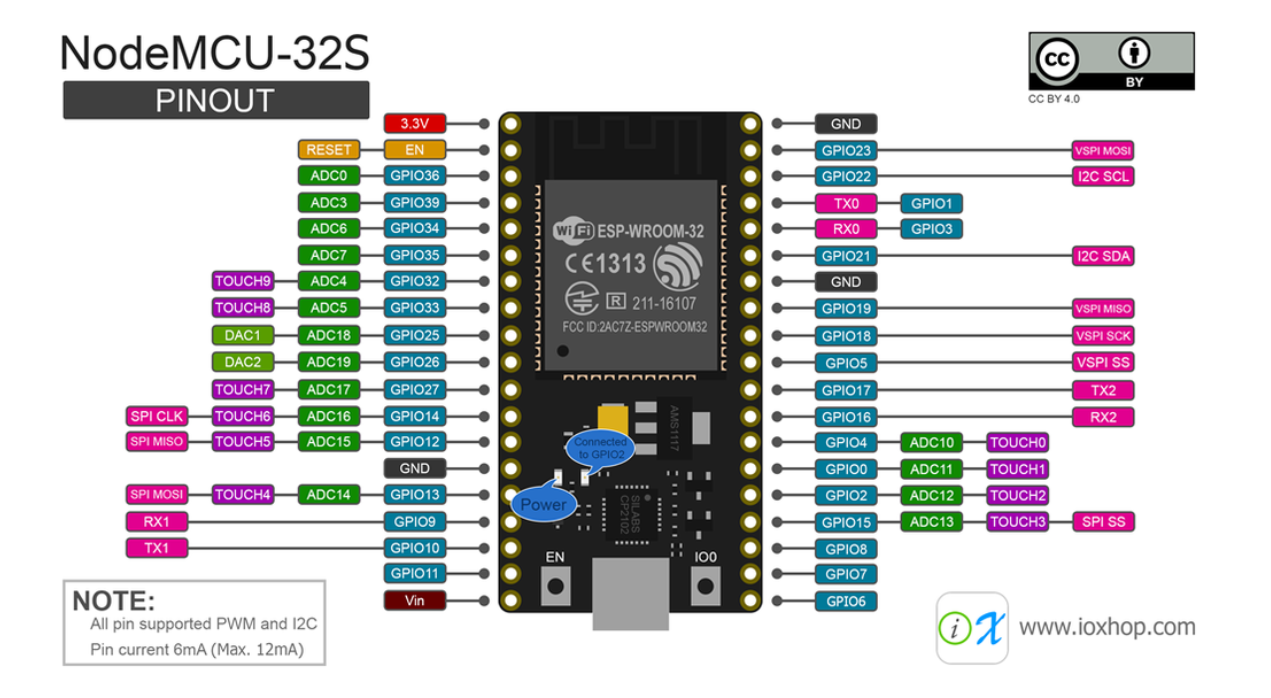


Ilustración 4 actualización de UART Bridge

Un UART Bridge es un mecanismo que permite la comunicación serial entre el microcontrolador ESP32 y otros dispositivos, como computadoras, módulos o sensores, mediante el protocolo UART (Universal Asynchronous Receiver-Transmitter). El ESP32 tiene múltiples interfaces UART que se pueden usar para transmitir y recibir datos.

**Conversión de Protocolo**: Si bien el ESP32 puede utilizar UART directamente, un UART Bridge (como un convertidor USB a UART) permite conectar el ESP32 a una computadora o a otro dispositivo que no tenga una interfaz UART directa. Esto facilita la programación del ESP32 y la visualización de datos en tiempo real.

La placa usada por Typsa es la node MCU ESP-3S



El primer problema al que me enfrento es la alimentación a 5 v del sensor pues la placa ESP32 funciona a 3,3 v.

## 15/10/2024

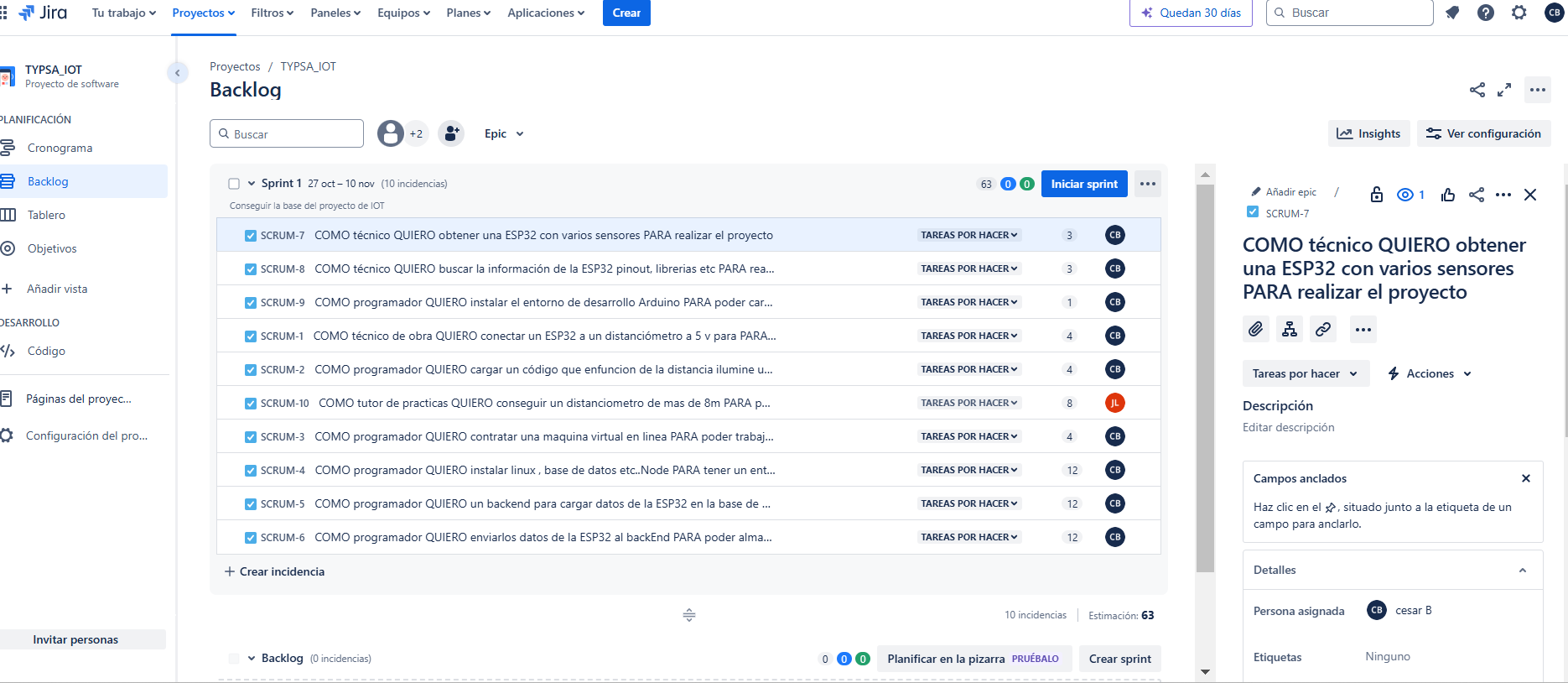
La placa realmente trabaja a 5v , el pin indicado como Vin es la salida.

Después de varias pruebas comienzo con el montaje de la placa y el sensor de distancia.

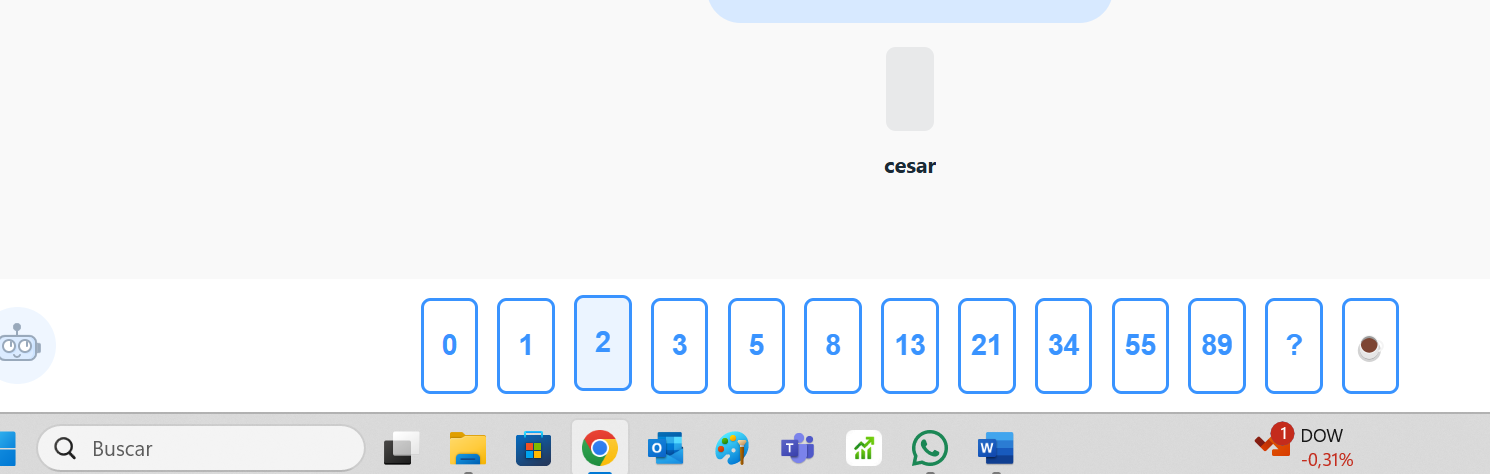
El trabajo lo voy a organizar en Sprints , con el programa Jira

## 16/10/2024

Registro en Jira y creación del Sprint 1 para organizar los trabajos.



Comienzo con una simulación de Poker planning para poder valorar el esfuerzo de cada tarea o incidencia.



## 17/10/2024

Reunión con Javier para obtener otro sensor resistente a la intemperie que permite hacer pruebas en exteriores , el funcionamiento es parecido .

El **A02YYUW** es un sensor ultrasónico impermeable de alta precisión que se utiliza para la detección de distancia en entornos hostiles o exteriores, como medición de nivel de agua, estacionamientos, y control de obstáculos. Tiene varias características útiles, como su resistencia al agua y capacidad para operar en condiciones difíciles.

**Características principales del A02YYUW:**

1. **Rango de detección**: Generalmente de **25 cm** hasta **450 cm**, con un ángulo de detección de alrededor de 30 grados.
2. **Impermeabilidad**: Diseñado para uso en exteriores o ambientes húmedos, es completamente resistente al agua.
3. **Salida de datos**: Este sensor puede tener diferentes métodos de salida, como:
   * **PWM (modulación por ancho de pulso)**.
   * **Salida analógica**.
   * **Comunicación UART (TTL)** para microcontroladores.
4. **Voltaje de operación**: Funciona con **5V DC**.
5. **Frecuencia**: Opera a una frecuencia ultrasónica de **40kHz**.

**Esquema de pines:**

1. **VCC**: Alimentación del sensor, **5V**.
2. **GND**: Tierra.
3. **TX/Trig**: Pin de transmisión para comunicación **UART** o para el pulso de disparo en modo **PWM**.
4. **RX/Echo**: Pin de recepción en modo **UART** o para recibir el eco en **PWM**.

**Conexiones en modo UART (TTL):**

1. **VCC (sensor)**: Conectar al pin **3.3V** o **5V** de la NodeMCU (según el modelo del sensor).
2. **GND (sensor)**: Conectar al **GND** común del sistema.
3. **TX (sensor)**: Conectar al pin **RX** de la NodeMCU.
4. **RX (sensor)**: Conectar al pin **TX** de la NodeMCU (si es necesario, para comandos bidireccionales).

**Conexiones en modo PWM:**

1. **VCC (sensor)**: Conectar a **5V**.
2. **GND (sensor)**: Conectar a **GND** común de la NodeMCU.
3. **Trig (sensor)**: Conectar a un pin digital de la NodeMCU para enviar el pulso de disparo (por ejemplo, **D1**).
4. **Echo (sensor)**: Conectar a un pin digital de la NodeMCU para recibir el pulso de eco (por ejemplo, **D2**).

Busco el pinOut de la placa y realizo las conexiones.

|  |  |
| --- | --- |
|  |  |

## 18/10/2024

Realizo cursos de Sys y calidad propios e internos de typsa del centro de trabajo.

## 21/10/2024 Sensor HC-SR04

Comienzo con las pruebas de la placa, el primer sensor a probar será el distanciómetro **HC-SR04** ,es un sensor ultrasónico ampliamente utilizado en proyectos de electrónica y robótica para medir distancias. Funciona emitiendo un pulso ultrasónico y midiendo el tiempo que tarda en reflejarse en un objeto y volver al sensor. Todo lo anterior por un precio de entorno a 2 euros



Ilustración 5 Sensor HC-SR04

**Componentes:**

1. **Módulo Emisor y Receptor de Ultrasonidos**:
   * **Trigger (TRIG)**: Este pin se usa para enviar un pulso de control, generalmente un pulso de 10 microsegundos que inicia la medición.
   * **Echo (ECHO)**: Este pin recibe el eco del pulso ultrasónico reflejado en el objeto y se usa para medir el tiempo que tardó en volver.
   * **Emisor y receptor ultrasónico**: El emisor genera ondas ultrasónicas, mientras que el receptor capta las ondas reflejadas.
2. **Controlador interno**: Gestiona el envío y la recepción de señales y calcula el tiempo que el pulso tarda en reflejarse para poder determinar la distancia.



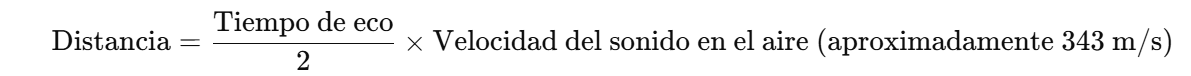
Ilustración 6 Esquema de montaje HC-SR04

**Especificaciones:**

* **Voltaje de operación**: 5V
* **Corriente de operación**: 15 mA
* **Frecuencia de funcionamiento**: 40 kHz
* **Rango de medición**: 2 cm a 400 cm
* **Precisión**: ±3 mm
* **Ángulo de detección**: Aproximadamente 15 grados

**Funcionamiento:**

1. El pin **TRIG** recibe un pulso de 10 µs.
2. El sensor emite una onda ultrasónica a una frecuencia de 40 kHz.
3. Cuando la onda se refleja en un objeto, el pin **ECHO** recibe el eco y el controlador mide el tiempo entre el envío y la recepción de la señal.
4. La distancia se calcula usando la fórmula:



## 22/10/2024 Servidor Cloud

Después de estudiar las alternativas con un pequeño comparativo me decanto por IONOS unida como marca a 1&1 combina dominios, correo electrónico y páginas web con hosting de primera clase, soluciones cloud escalables y asesor personal a un precio simbólico de 1 €/mes para un servidor Linux de muy baja capacidad, pero suficiente para la realización de pruebas a baja escala.



Ilustración 7 servidores cloud Ionos

En la pagina de Ionos después de leer dos pequeños manuales despliego un servidor con Ubuntu 24.04 e ip fija publica 217.160.229.64 para poder conectarme desde el exterior.

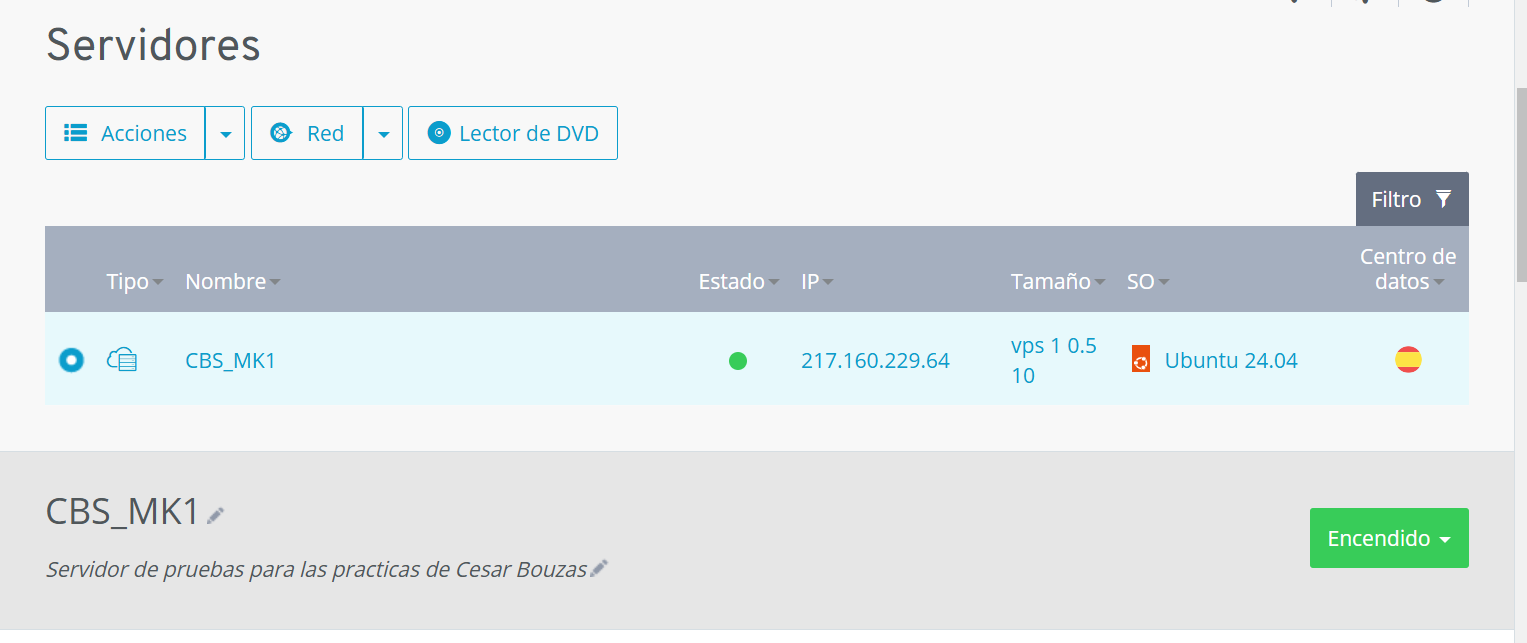


Ilustración 8 Servidor cloud de Ionos

## 23/10/2024 a 25/10/2024 Primeras pruebas con el distanciómetro.

Con el ide de Arduino ya instalado en el ordenador y las librerías cargadas con el gestor de placas o Boards Manager :

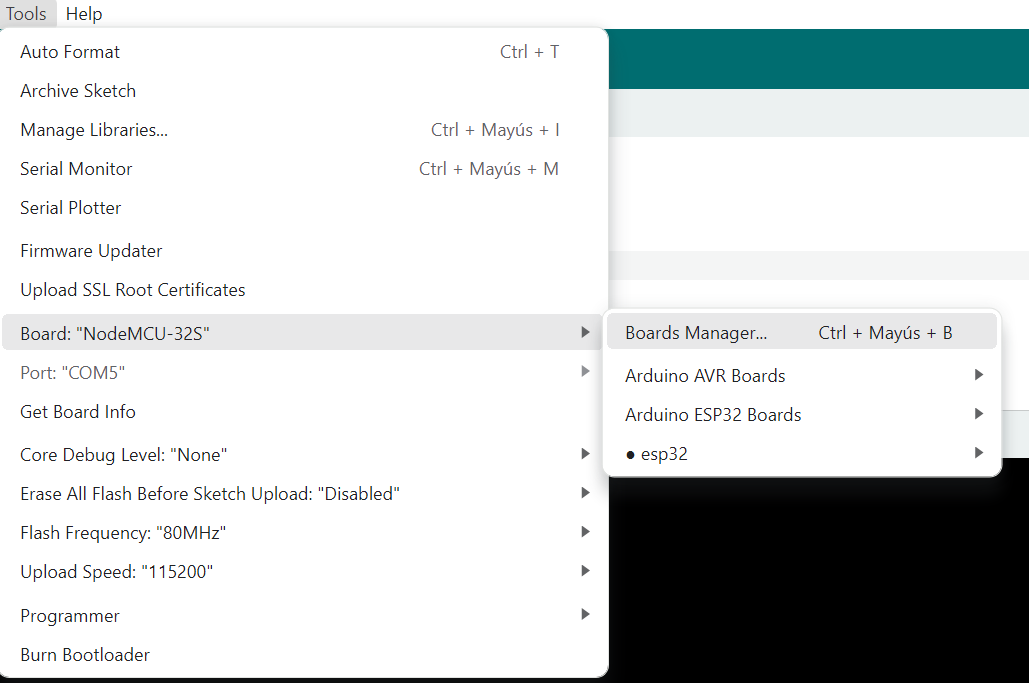


Ilustración 9 boards manager Arduino

Aquí te explico cómo hacerlo:

1. Abre el **Arduino IDE**.
2. Ve a **Archivo > Preferencias**.
3. En la sección de **Gestor de URLs de Tarjetas**, añade esta URL:

https://dl.espressif.com/dl/package\_esp32\_index.json

1. Ve a **Herramientas > Placa > Gestor de placas**.
2. Busca **esp32** y haz clic en **Instalar**.

**c. Seleccionar la Placa NodeMCU-32S**

1. Ve a **Herramientas > Placa > ESP32 Arduino** y selecciona **NodeMCU-32S** .
2. Configura el puerto adecuado en **Herramientas > Puerto** para seleccionar el puerto USB al que está conectada la placa.

Si se conecta la placa en mi caso por USB se localiza en el COM 5.

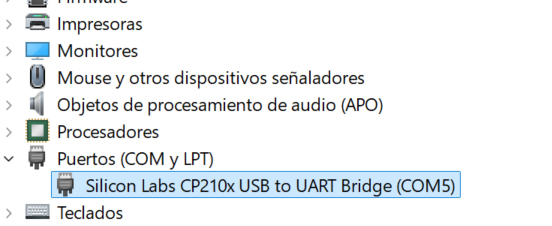
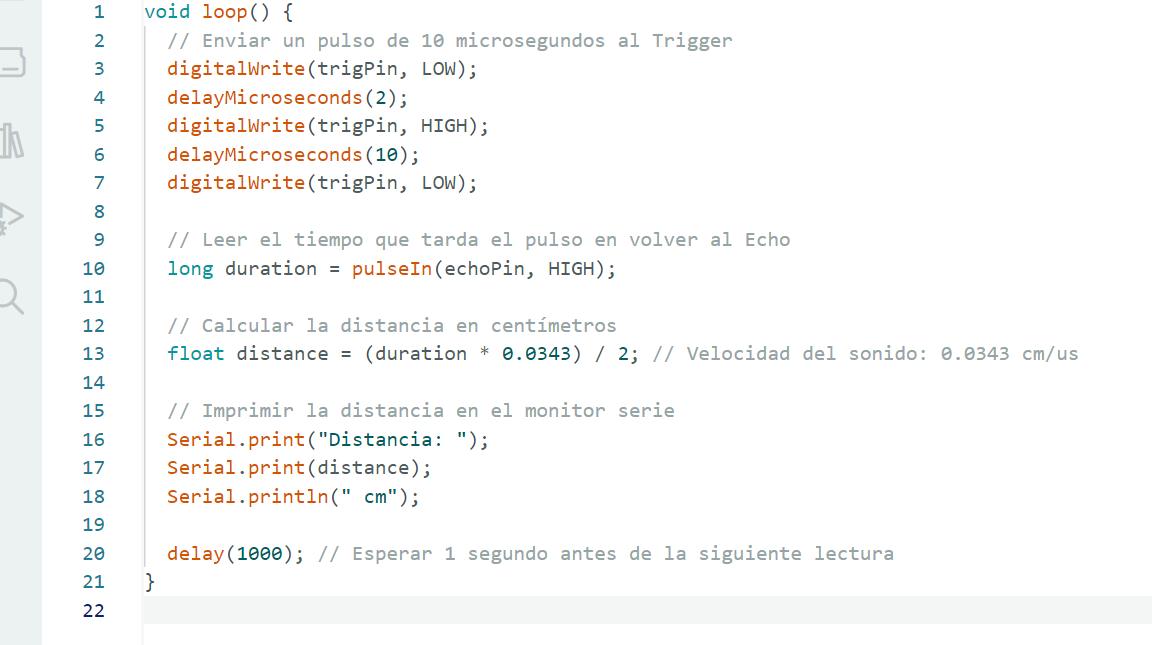


Ilustración 10 Driver y placa en el COM5

Como ya se explicó en un apartado anterior el funcionamiento de la placa coloco aquí directamente el código con una pequeña explicación:



Este código:

1. Define los pines.
2. Envía un pulso ultrasónico.
3. Mide el tiempo que tarda el eco en volver al sensor.
4. Calcula la distancia en base a ese tiempo.
5. Imprime la distancia medida en centímetros y repite el ciclo una vez por segundo.

## 28/10/2024 a 30/10/2024 Conexión al servidor y primeras pruebas.

Basándome en la tarea siguiente del Sprint que tienen una puntuación de 12

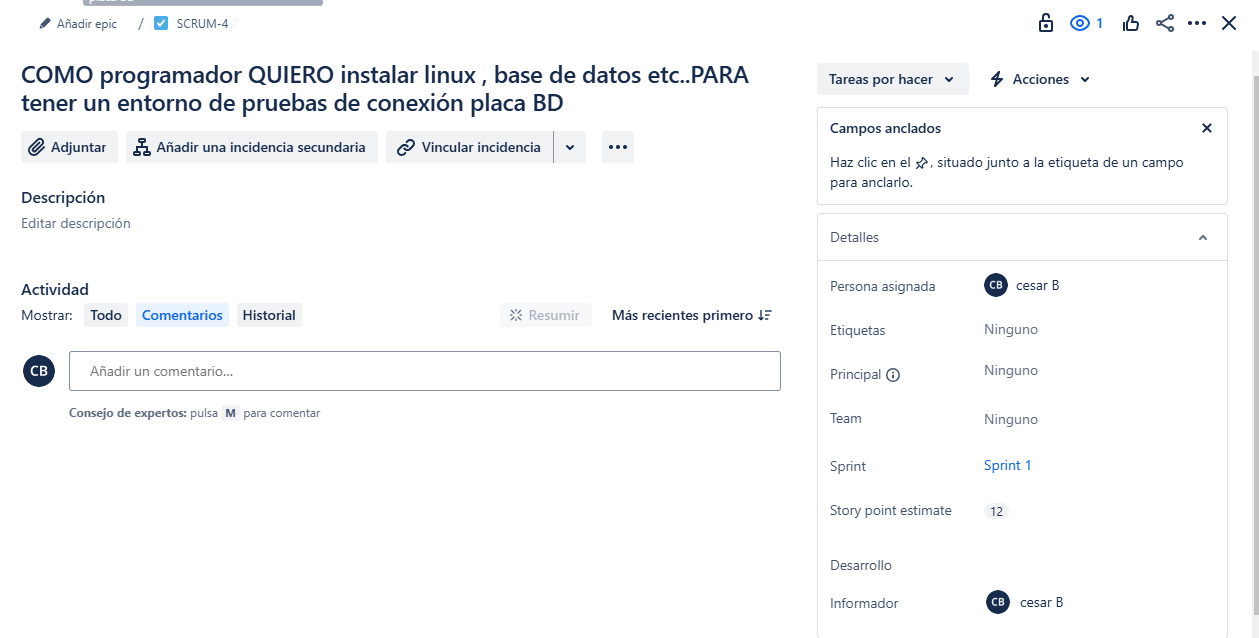


Ilustración 11 Jira Tarea Servidor Remoto

### Instalación de Putty .

Instalo el programa Putty para conectarme por consola al ordenador remoto.

|  |
| --- |
| Ilustración 12 interface Putty |

Al iniciar la conexión me pide el usuario y contraseña en nuestro caso root y s4u6iLUF

|  |
| --- |
| Ilustración 13 Terminal del server iCloud desde nuestro ordenador |

### Instalación de Linux , Python y Flask

Indicar que Linux ya viene instalado en la máquina virtual cloud , simplemente se debe elegir el SO que se desea (si es propietario puede tener un coste) en mi caso selecciono Ubuntu 24.

#### Python

Para comenzar con las pruebas montamos un fremework server flask

**Instalación de Python en un entorno virtual en Linux**

**1. Instalar Python**

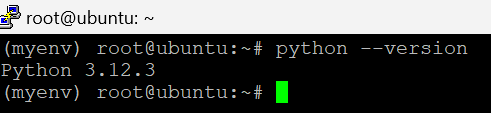
Para instalar Python, el siguiente comando:

sudo apt update

sudo apt install python3

Para verificar la instalación:

python3 –version



**2. Instalar venv**

El módulo venv viene preinstalado con Python 3.3 en adelante. Si no está, se instala con:

sudo apt install python3-venv

**3. Crear un entorno virtual**

En la carpeta donde se quiere crear el entorno virtual se ejecuta:

python3 -m venv mi\_entorno

Esto creará una carpeta llamada mi\_entorno que contendrá el entorno virtual.

**4. Activar el entorno virtual**

Para activar el entorno virtual,se ejecuta el siguiente comando:

source mi\_entorno/bin/activate

|  |
| --- |
|  |

### Instalación de Flask.

Ahora que el entorno virtual está activado, para instalar paquetes de Python específicos para este entorno usando pip sin afectar a la instalación global de Python.

pip install flask

**Flask** es un microframework de desarrollo web en Python, diseñado para ser simple y fácil de usar. Fue creado con la filosofía de proporcionar solo los componentes esenciales para construir una aplicación web, lo que permite a los desarrolladores tener mucha flexibilidad al agregar otras bibliotecas y herramientas según las necesidades del proyecto.

|  |
| --- |
| Ilustración 14 version de Flask |

**Este es el framework para pruebas más tarde y según la evolución de las practicas valoraré el uso de Django**. Siempre basado en Python por ser un lenguaje no tratado en el ciclo y con mucha popularidad y gran comunidad en el mundo de las ingenierías.

Ahora creo un archivo app.py con el objeto de recibir solicitud post con datos de un sensor .El objetivo del programa es recibir datos y mostrar la información recibida en una respuesta simple.

**Para las pruebas de funcionamiento utilizamos Postman** para las consultas sin necesidad de tener la placa funcionando, así independizamos las posibles causas de error.

|  |
| --- |
| re  Ilustración 15 primer codigo del server flask |

Explicando el código:

from flask import Flask , request

**Flask**: Es el módulo principal del framework Flask, que se usa para crear la aplicación web.

**request**: Permite acceder a los datos de la solicitud HTTP enviada al servidor, en este caso, accederemos a los datos enviados mediante el método POST.

app=Flask(\_\_name\_\_)

Se crea y almacena en una variable una instancia de framework Flask que es una aplicación web que maneja las solicitudes que llegan a ella.

@app.route(“/”,methods=[‘Post’])

Esta línea define una **ruta** o **endpoint**. La ruta '/' corresponde a la página principal del servidor.

El método POST indica que esta ruta solo acepta solicitudes **POST** (comúnmente usadas para enviar datos, como en formularios HTML).

El objeto request.form accede a los datos enviados en el cuerpo de la solicitud POST.

Def receive\_data():

sensor=request.form.get(‘sensor’)

value=request.form.get(‘value’)

**request.form.get('sensor')** obtiene el valor del campo sensor que fue enviado en la solicitud.

**request.form.get('value')** obtiene el valor del campo value que fue enviado en la solicitud.

return f"El valor de el sensor {sensor} es de {value}", 200

Esta línea devuelve una cadena de texto que será la respuesta enviada al cliente. La respuesta indica el nombre del sensor y el valor recibido.

Se utiliza una **f-string** para interpolar las variables sensor y value en la respuesta.

El 200 es el **código de estado HTTP**, lo que indica que la solicitud fue exitosa.

if \_\_name\_\_ == '\_\_main\_\_':

app.run(host='0.0.0.0', port=5000)

Este bloque asegura que la aplicación Flask se ejecute solo cuando el archivo se ejecute directamente (no cuando se importe como un módulo en otro archivo).

**app.run(host='0.0.0.0', port=5000)** inicia el servidor Flask en el puerto **5000**, y la opción host='0.0.0.0' permite que el servidor sea accesible desde cualquier dirección IP de la red (útil si se está ejecutando en una máquina remota) como es nuestro caso .

#### Probando el servidor .

El framework flask es rápido y sendillo . Una vez indiciado indica en la tyerminal los intentos de conexión con terminales remotos , hora y tipo de petición (GET,POST…)

|  |
| --- |
| Ilustración 16 Terminal de Ubuntu con Flask corriendo |

Utilizando Postman , podemos emular una petición post con el tipo de sensor y la lectura . En nuestro caso enviaremos un sensor tipo Temperature y una medición de 67.

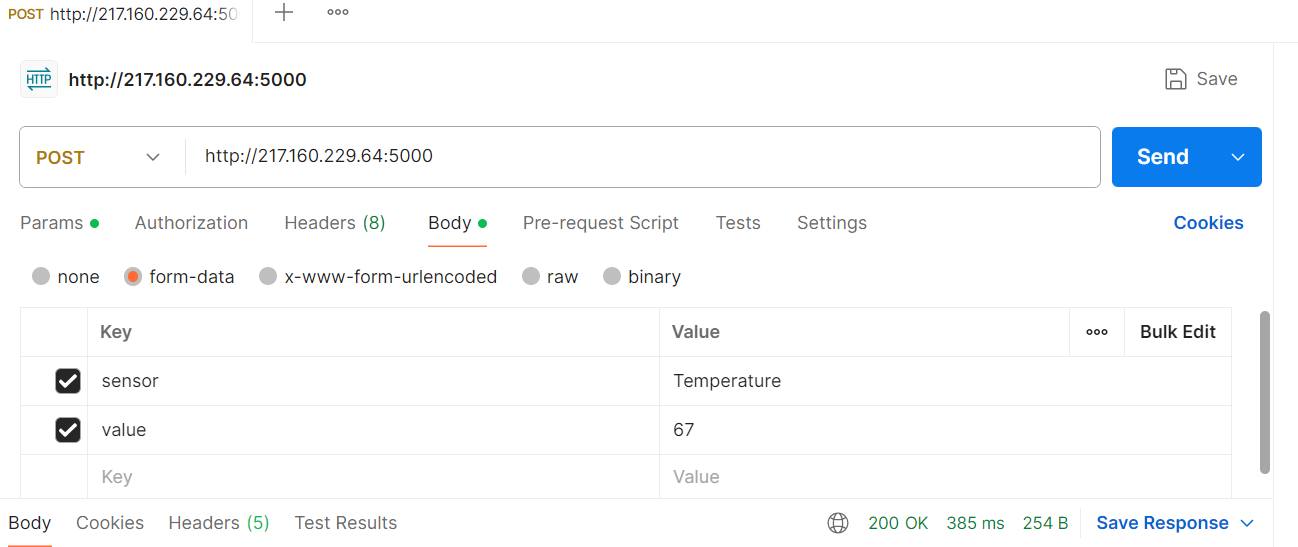


Ilustración 17 Petición tipo Post con postman

|  |
| --- |
| Ilustración 18 El resultado en Flask |

Se pude observar que se obtienen el código 200 , que significa que todo ha ido correctamente .

### Instalación de base de datos .

En este apartado me decanto por una base de datos relacional , en un principio solo tendré una tabla con la fecha, el tipo de sensor y el valor de la lectura . Pero en un futuro los tipos de sensor será una tabla propia .

El enunciado de lo que necesito podría ser el siguiente:

*Un sensor puede ser de varios tipos, un sensor puede realizar muchas mediciones y puede estar localizado solo o con otros sensores formando una estación de medición, las estaciones de mediciones estarán en una ubicación . Las medidas deberán guardar la fecha y hora con su valor y un identificador de medida. Las estaciones tendrán un identificador con los distintos identificadores de los sensores que las componen . Las ubicaciones tendrán lat y longitud asi como una ciudad y país , nombre personalizado y una estancia como puede ser oficina garaje etc..El sensor puede tener varios estados como son , activo ,averiado , apagado etc..*

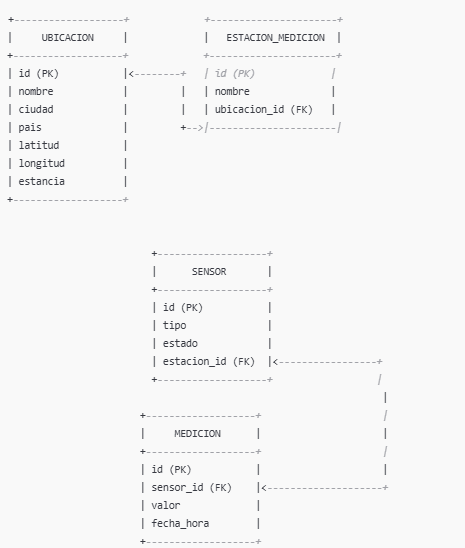
**

Ilustración 19 Base de datos relacional de pruebas

#### Postgresql en Linux

Mediante la terminal con Putty realizamos las siguientes operaciones:

1. **Actualizar la lista de paquetes:**

sudo apt update

1. **Instalar PostgreSQL:**

sudo apt install postgresql postgresql-contrib

1. **Iniciar el servicio de PostgreSQL:**

sudo systemctl start postgresql

1. **Habilitar PostgreSQL para que inicie al arranque:**

sudo systemctl enable postgresql

1. **Verificar que PostgreSQL esté corriendo:**

sudo systemctl status postgresql

|  |
| --- |
| Ilustración 20 Estado de Postgresql |

Debemos configurar el firewall de IONOS activamos el puerto 5432 .

En postgre debemos cambiar la configuración para poder conectar desde cualquier IP y con un determinado usuario para poder en fase de diseño poder ir viendo los resultados.

Para este ultimo punto instalamos la el gestor de base de datos Dbear que es universal para cualquier tipo de base de datos.



Ilustración 21 Inicio de Dbear gestor universal de BD

Mediante Dbear podemos conectarnos remotamente a la Bd y ver asi los distintos contenidos de las tablas , su creación etc…

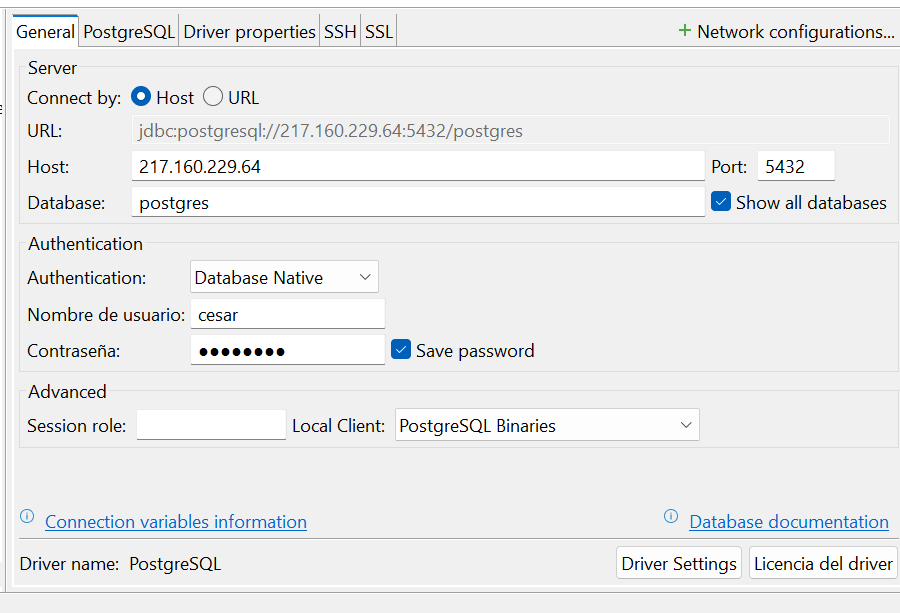


Ilustración 22 Datos de Conexión mediante Dbear

Para comenzar las pruebas creo una tabla de Distanciómetro :

|  |
| --- |
| Ilustración 23 Sql de creación de Tabla |
| Ilustración 24 Diagrama E-R |

## 01/11/2024 a 07/10/2024 Conexión a Wifi ESP32 , envío de post al server y almacenamiento de medidas en BD.

Este apartado se corresponde con la Story del Sprint 1 :



### Consiste en modificar el codigo de la placa para enviar el dato de la medición mediante post al server y que este lo almacene en la BD.

### Código ESP32 para conexión wifi y envio post.

El cambio es relativamente sencillo pues cargando una serie de dependencias en el código obtenemos el resultado pedido.

**Explicación de la conexión WiFi y envío de datos POST:**

1. **Conexión a la red WiFi**:
   * En la función setup, se configura la conexión WiFi usando WiFi.begin(ssid, password), que toma el SSID y la contraseña especificados. El programa espera hasta que se establece la conexión y muestra un mensaje en el monitor serie indicando que se ha conectado.
2. **Envío de datos mediante una solicitud POST**:
   * En el loop, si la conexión WiFi está activa, se utiliza HTTPClient para enviar datos al servidor en una solicitud POST.
   * Se configura la URL del servidor (http.begin("http://217.160.229.64:5000");) y se añade el encabezado Content-Type para especificar que se envía el contenido en formato application/x-www-form-urlencoded.
   * El contenido del mensaje se prepara con el formato clave=valor, donde clave es "sensor" y "value" contiene la distancia medida (String(distance)).
   * Luego, se envía la solicitud POST y se verifica la respuesta del servidor: si es exitosa, se muestra la respuesta; si falla, se imprime un mensaje de error.

**Otros componentes del funcionamiento:**

* **Medición de distancia**: En cada ciclo, se emite un pulso en el trigPin, y se mide el tiempo que tarda en volver en el echoPin usando pulseIn. Luego, se calcula la distancia en centímetros.
* **Indicación de ocupación**: Según el valor de la distancia, el sistema enciende el LED rojo (ocupado) o el verde (libre) y muestra un mensaje en el monitor serie.

|  |
| --- |
| * #include <WiFi.h> * #include <HTTPClient.h> * // Configura credenciales de WiFi * const char\* ssid = "ZAAPA"; * const char\* password = "manuelbouzas"; * // Definir pines * const int trigPin = 4;  // Pin de Trigger HC-SR04 * const int echoPin = 2;  // Pin de Echo HC-SR04 * const int redPin = 16;    //Pin rojo * const int greenPin = 17;  //Pin Verde * const int yellowPin = 5;  //pin Amarrilo * const float h = 23.5;     //Altura del aparcamiento * void setup() { * Serial.begin(115200);  // Iniciar comunicación serie * // Conectar a WiFi * WiFi.begin(ssid, password); * while (WiFi.status() != WL\_CONNECTED) {   + delay(1000);   + Serial.println("Conectando a WiFi..."); * } * Serial.println("Conectado a WiFi"); * pinMode(trigPin, OUTPUT);   // Configurar el pin Trigger como salida * pinMode(echoPin, INPUT);    // Configurar el pin Echo como entrada * pinMode(redPin, OUTPUT);    //Configurar el pin Rojo como salida * pinMode(greenPin, OUTPUT); //Configurar el pin Verde como salida * pinMode(yellowPin, OUTPUT);//Configurar el pin Amarillo como salida * } * void loop() { * // Enviar un pulso de 10 microsegundos al Trigger * digitalWrite(trigPin, LOW); * delayMicroseconds(2); * digitalWrite(trigPin, HIGH); * delayMicroseconds(10); * digitalWrite(trigPin, LOW); * // Leer el tiempo que tarda el pulso en volver al Echo * long duration = pulseIn(echoPin, HIGH); * // Calcular la distancia en centímetros * float distance = (duration \* 0.0343) / 2;  // Velocidad del sonido: 0.0343 cm/us * // Imprimir la distancia en el monitor serie * Serial.print("Distancia: "); * Serial.print(distance); * Serial.println(" cm"); * if (distance < h) {   + Serial.print("Plaza ocupada ");   + digitalWrite(greenPin, LOW);   + digitalWrite(redPin, HIGH); * } else {   + Serial.print("Plaza libre ");   + digitalWrite(greenPin, HIGH);   + digitalWrite(redPin, LOW); * } * // Si la conexión WiFi está activa, enviar los datos por POST * if (WiFi.status() == WL\_CONNECTED) {   + HTTPClient http;   + http.begin("http://217.160.229.64:5000");// Especifica la URL a la que se va a hacer la solicitud   + http.addHeader("Content-Type", "application/x-www-form-urlencoded"); // Configura el tipo de contenido   + String postData = "sensor=distanciometro&value=" + String(distance);// Crear el cuerpo del POST con el formato clave-valor   + int httpResponseCode = http.POST(postData);// Enviar la solicitud POST   + // Verificar la respuesta del servidor   + if (httpResponseCode > 0) {   + String response = http.getString();   + Serial.println("Respuesta del servidor: " + response);   + } else {   + Serial.println("Error en la solicitud POST");   + }   + http.end(); // Terminar la conexión * } * delay(1000);  // Esperar 1 segundo antes de la siguiente lectura * } |

### Código en Python para recepción del post y almacenamiento en la BD.

El objeto de esta parte es poder recibir el post desde la ESP32 y alamcenar los datos en la base de datos relacional . Como extra como prueba mostramso las mediciones en una tabla Html en la dirección /distanciómetro.

|  |
| --- |
| from flask import Flask, request, render\_template\_string  import psycopg2  from datetime import datetime  # Para manejar la fecha y hora en caso necesario  app = Flask(\_\_name\_\_)  # Datos de conexión a PostgreSQL  DB\_HOST = '217.160.229.64'  # Reemplaza con la IP de tu servidor  DB\_NAME = 'postgres'        # El nombre de tu base de datos  DB\_USER = 'cesar'           # El nombre de usuario  DB\_PASS = '21212121'        # La contraseña  DB\_PORT = '5432'            # Puerto de PostgreSQL  # Función para insertar datos en la base de datos  def insert\_data\_to\_db(sensor, value):      try:          # Conexión a PostgreSQL          conn = psycopg2.connect(              host=DB\_HOST,              database=DB\_NAME,              user=DB\_USER,              password=DB\_PASS,              port=DB\_PORT          )          cursor = conn.cursor()          # Consulta SQL para insertar el dato con el timestamp          query = """          INSERT INTO distanciometro (lectura, fecha)          VALUES (%s, CURRENT\_TIMESTAMP);          """          cursor.execute(query, (value,))          # Guardar los cambios          conn.commit()          # Cerrar la conexión          cursor.close()          conn.close()      except Exception as e:          print(f"Error al insertar datos en la base de datos: {e}")  # Ruta principal donde recibimos los datos y los mostramos en HTML  @app.route('/', methods=['POST'])  def receive\_data():      sensor = request.form.get('sensor')      value = request.form.get('value')      # Insertar el valor en la base de datos junto con la fecha y hora      insert\_data\_to\_db(sensor, value)      return f"El valor del sensor {sensor} es de {value} y ha sido insertado en la base de datos.", 200  # Ruta para mostrar el contenido de la tabla 'distanciometro' en HTML  @app.route('/distanciometro', methods=['GET'])  def show\_distanciometro():      # Obtener los datos de la tabla      data = get\_data\_from\_db()      # Plantilla HTML para mostrar la tabla      html\_template = '''      <html>      <head>          <title>Datos del Distanciómetro TYPSA MK1</title>          <style>              table {                  width: 70%;                  margin: 0 auto;                  border-collapse: collapse;              }              table, th, td {                  border: 1px solid black;              }              th, td {                  padding: 10px;                  text-align: center;              }              th {                  background-color: #f2f2f2;              }          </style>      </head>      <body>          <h2 style="text-align:center;">Datos del Distanciómetro Typsa V1</h2>          <table>              <thead>                  <tr>                      <th>ID</th>                      <th>Lectura</th>                      <th>Fecha y Hora</th>                  </tr>              </thead>              <tbody>                  {% for row in data %}                  <tr>                      <td>{{ row[0] }}</td>                      <td>{{ row[1] }}</td>                      <td>{{ row[2] }}</td>                  </tr>                  {% endfor %}              </tbody>          </table>      </body>      </html>      '''      # Renderizar los datos en la tabla HTML      return render\_template\_string(html\_template, data=data)  # Función para obtener los datos de la tabla 'distanciometro'  def get\_data\_from\_db():      try:          # Conexión a PostgreSQL          conn = psycopg2.connect(              host=DB\_HOST,              database=DB\_NAME,              user=DB\_USER,              password=DB\_PASS,              port=DB\_PORT          )          cursor = conn.cursor()          # Consulta SQL para obtener los datos incluyendo el timestamp          query = "SELECT id\_lectura, lectura, fecha FROM distanciometro;"          cursor.execute(query)          # Recuperar todos los registros          rows = cursor.fetchall()          # Cerrar la conexión          cursor.close()          conn.close()          return rows      except Exception as e:          print(f"Error al obtener datos de la base de datos: {e}")          return []  if \_\_name\_\_ == '\_\_main\_\_':      app.run(host='0.0.0.0', port=5000) |

El resultado de la ejecución de los dos apartados , es decir la ESP32 conectada a WIFI y emitiendo las distancias mediante Post hacia un server que las almacena en una base de dato , que se muestra en la dirección <http://217.160.229.64:5000/distanciometro>

|  |
| --- |
| Ilustración 25 Resultado del Get al Flask con los datos de la BD |

|  |
| --- |
| Ilustración 26 Envio del request del GET |

**Con este apartado podemos dar por terminado el Sprint 1 , antes de la fecha prevista . Debido a esta circunstancia se añade del backlog la story de la implementación de Django , pues es un entorno mas estable que flask y incrementa el valor de los trabajos al introducir una tecnología de gran popularidad en el sector** .

#### Curso de Django en OpenwebInars

Como nuevo paso en nuestro aprendizaje utilizamos en la story los criterios de aceptación. En este contexto, los **criterios de aceptación** son las condiciones que deben cumplirse para considerar que la tarea o el proyecto han sido completados satisfactoriamente. Funcionan como una lista de verificación que define cuándo el trabajo se ha realizado de acuerdo con las expectativas y requisitos definidos.

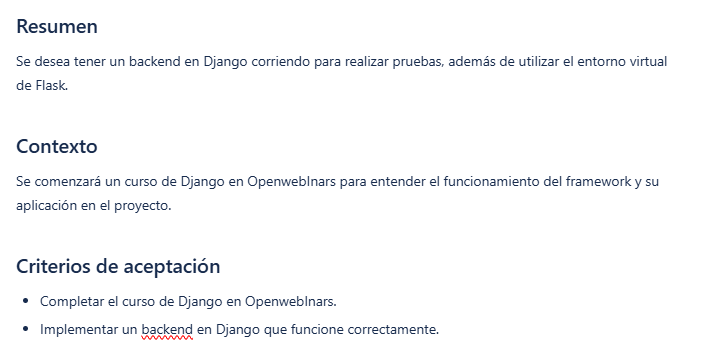


Ilustración 27 criterios de aceptación en Jira

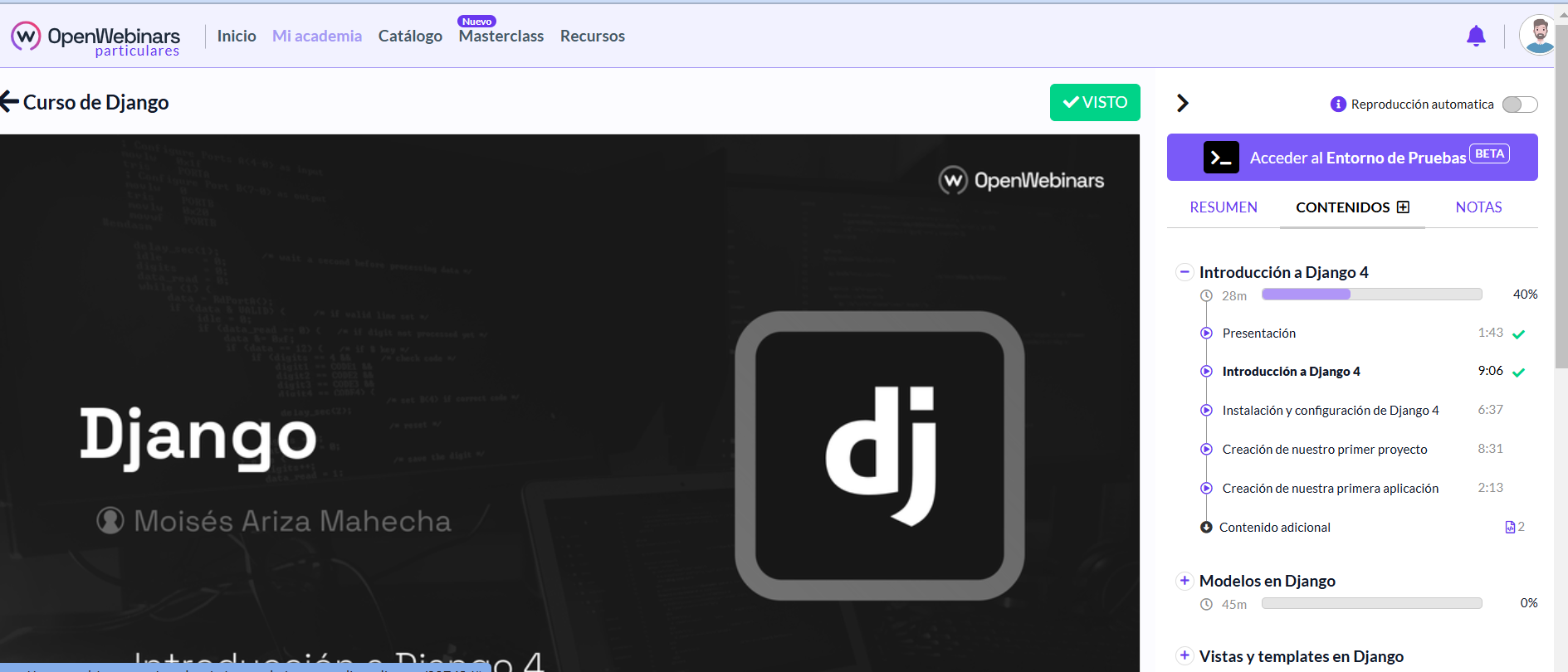


Ilustración 28 Comienzo del Curso de Django

En esencia el fujo de trabajo se parece mucho a modelo-vista-controlador pero realmente su nomenclatura es MVT es decir , modelo-vista-template cuyo simplificación es la siguiente:

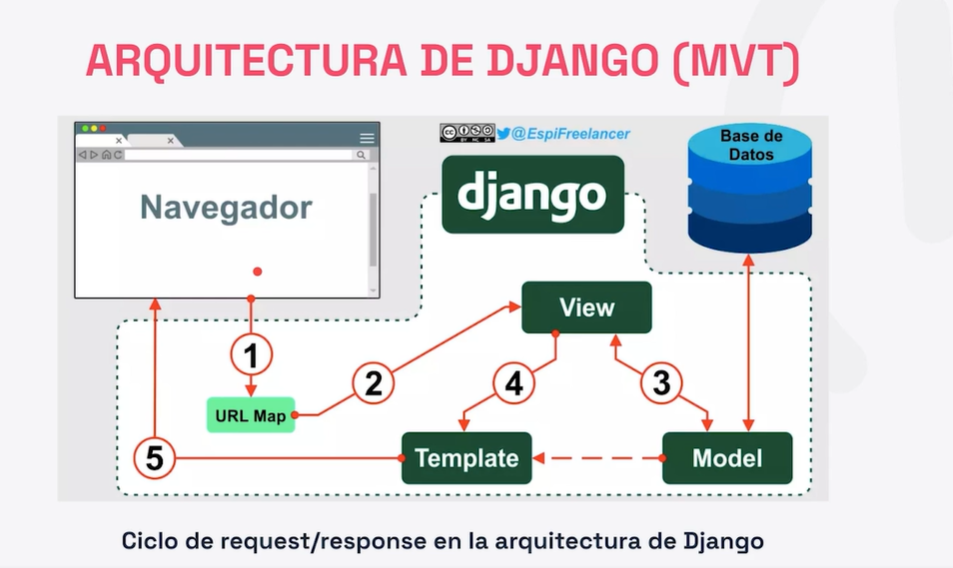


Ilustración 29 Modelo-vista-Template de Django

##### Instalación de Django .

Para instalar Django en Windows (Ordenador Local) lo primero que debo tener es Python instalado.

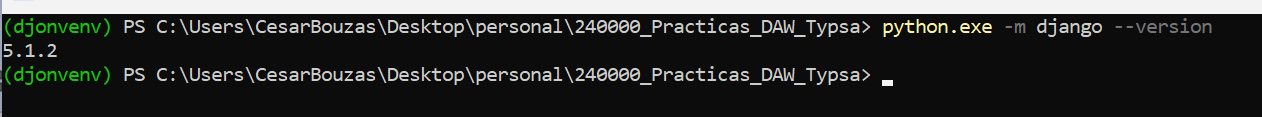
Creo un entorno virtual llamado djonvenv :

Python -m venv djonvenv

Lo activo mediante la ruta djonvenv\ Scripts\activate

Instalo Python pip install Django

El resultado es:



##### Creación del primer proyecto

Entramos en nuestro entorno virtual



##### Uso y migracion de la base de datos.

Para configurar la base de datos debemos editar el settings.py , en un pricnipio entraremos por ip publica para poder unsar el mismo codigo en local que en la terminal para hacer pruebas , esto lo cambiaremos llegado el momento de pasarlo a producción pues todas las conexiones serán desde el server como localhost.

La configuración en settings.py será la siguiente:

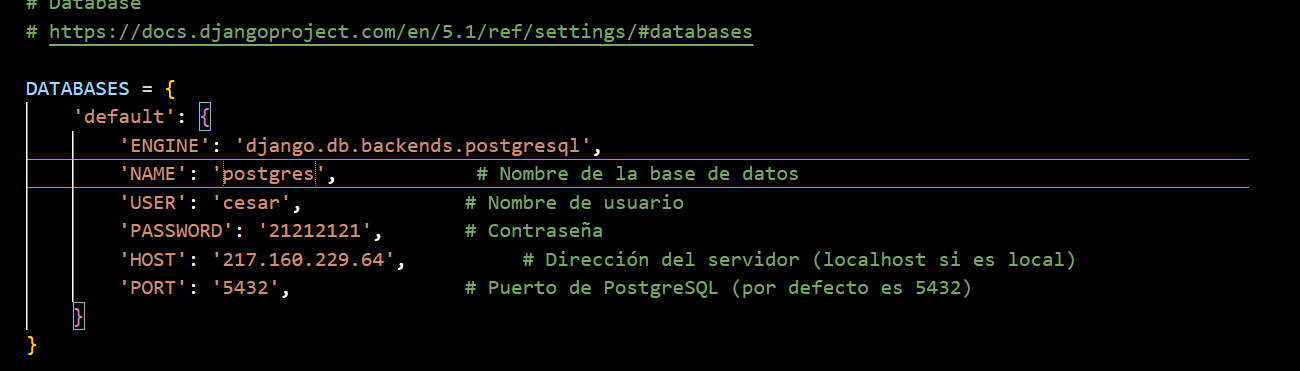


Ilustración 30 codigo en settings.py para nuestra bd

Al intentar conectar obtenemos un error debido a que no tenemos instalado la librería o dependencia psycopg2debemos usar el siguiente comando:

pip install psycopg2-binary

Después de introducir nuestros datos tenemos que realizar la migración . En el contexto de Django y otros frameworks web, una **migración** es un proceso para aplicar cambios en la estructura de una base de datos. Las migraciones permiten sincronizar la base de datos con el estado actual de los modelos definidos en el código, sin necesidad de realizar manualmente los cambios en la base de datos.

**¿Por qué son importantes las migraciones?**

Las migraciones ayudan a:

1. **Actualizar la estructura de la base de datos automáticamente** cuando se agregan, modifican o eliminan modelos o campos en el código.
2. **Evitar errores** al mantener la base de datos alineada con el código.
3. **Control de versiones**: Cada migración se guarda como un archivo que puede revertirse o actualizarse. Esto es útil para deshacer cambios si es necesario.

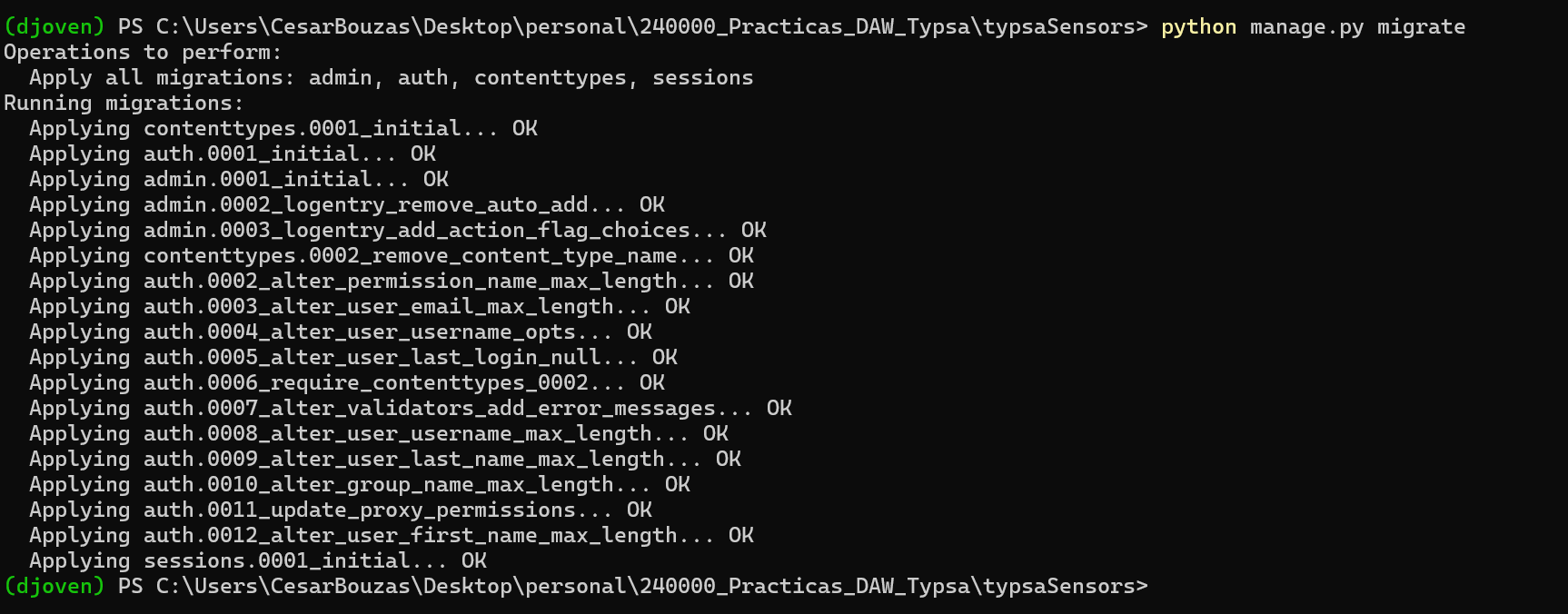
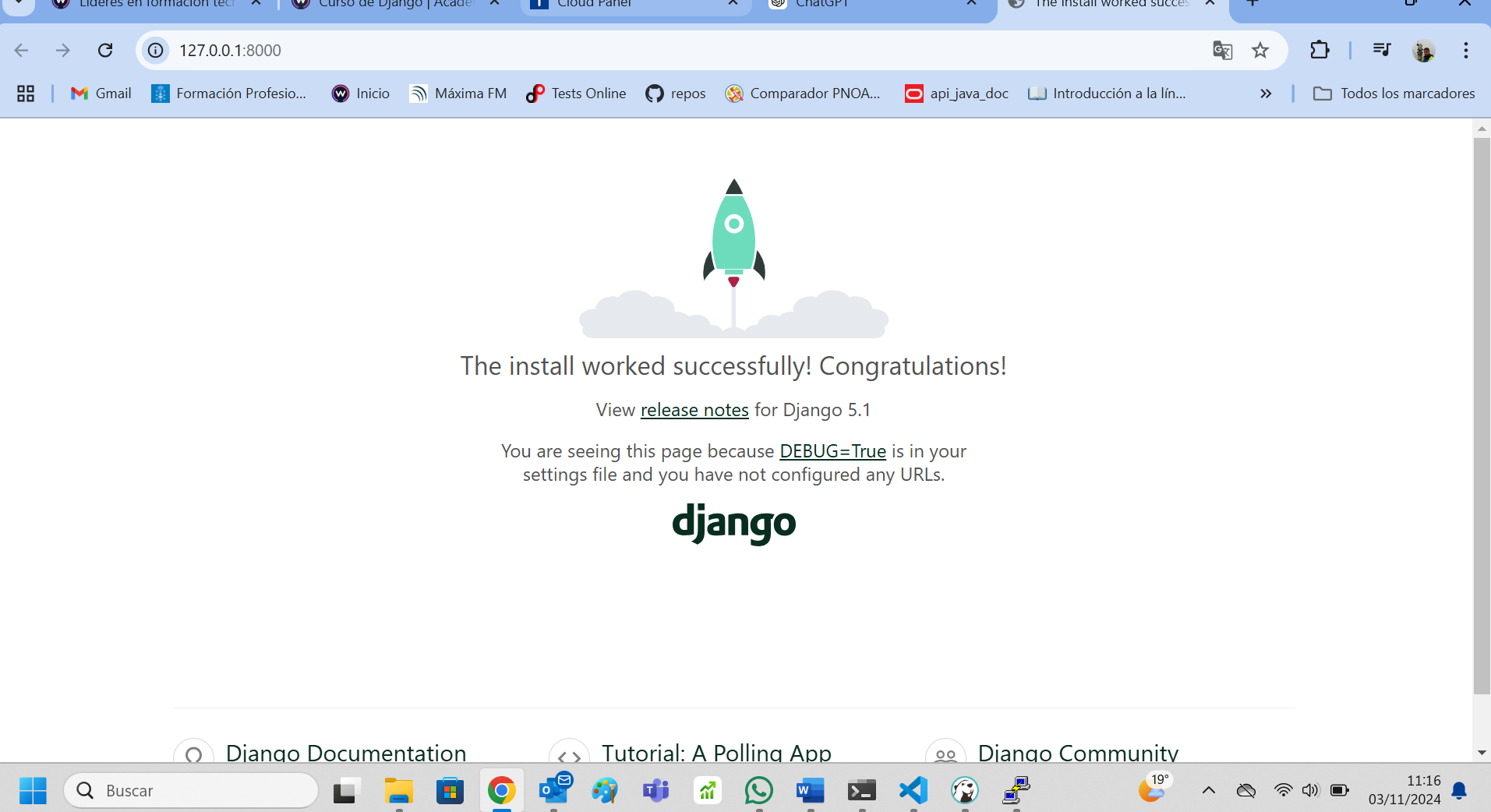


Ilustración 31 Resultado de la migracion

Llegado este momento podemos probar el server en local mediante el codigo :

Python manage.py runserver



Como vemos el server esta corriendo en nuestra terminal que hace como server . Llegado este momento queremos copiar este codigo en el la maquina virtual para lo cual intentaremos usar el programa GIT de control de versiones .

**Instalación y configuración de GIT (control de versiones)**

**Git** es un sistema de control de versiones distribuido, ampliamente utilizado en el desarrollo de software para rastrear y gestionar los cambios en el código fuente. Permite a varios desarrolladores trabajar en el mismo proyecto de manera simultánea sin sobrescribir el trabajo de otros, y es muy útil para mantener un historial detallado de todas las modificaciones realizadas en un proyecto.



**Características de Git**

1. **Distribuido**: Cada desarrollador tiene una copia completa del proyecto con todo su historial.
2. **Seguimiento de cambios**: Mantiene un historial completo de cambios para que puedas ver, revertir o fusionar versiones antiguas.
3. **Trabajo en equipo**: Facilita la colaboración entre varios desarrolladores al permitir trabajar en diferentes ramas y fusionar cambios.

Para que todo tenga sentido también creo una copia o imagen en Github para mantener los dos códigos sincronizados (terminal de trabajo y server de pruebas)

###### Github registro y creación de un repositorio.

El proceso de registro es sencillo como cualquier otra web de servicios :

|  |  |
| --- | --- |
|  |  |

###### Instalación en terminal remota Windows:

**Descargar el instalador**:

* Ve a la página oficial de Git: <https://git-scm.com>.
* Descarga la versión para Windows (se descarga automáticamente el instalador adecuado para tu sistema).

**Ejecutar el instalador**:

* Abre el archivo .exe descargado y sigue los pasos de instalación.
* Durante la instalación, puedes dejar las configuraciones por defecto, que son adecuadas para la mayoría de los usuarios.
* En el paso donde pregunta por el editor predeterminado, puedes elegir entre opciones.
* Selecciona la opción de instalar Git Bash si deseas usar una terminal similar a la de Linux para trabajar con Git en Windows.

###### Instalación en server ubuntu24: