



Departamento de Informática

Programación de bases de datos para Timeseries

Juan Gualberto

Noviembre 2023

Índice

| | |
|---|-----------|
| 1 Gestión de bases de datos para el IoT | 3 |
| 1.1 Redis | 4 |
| 1.1.1 Características Principales de Redis: | 5 |
| 1.1.2 Usos Comunes de Redis: | 5 |
| 1.2 InfluxDB | 6 |
| 1.2.1 Características Principales: | 6 |
| 1.2.2 Alternativas: | 7 |
| 2 Preparación del entorno | 9 |
| 2.1 La pila MING | 9 |
| 3 Introducción al IoT con ESP32 | 14 |
| 3.1 Objetivos: | 14 |
| 3.1.1 Metodología: | 15 |
| 3.2 Introducción a IoT y ESP32 | 15 |
| 3.2.1 Internet de las Cosas (IoT) | 15 |
| 3.2.2 ESP32 | 16 |
| 3.2.3 Casos de Uso Comunes | 16 |
| 3.2.4 Desafíos y Consideraciones en IoT | 16 |
| 3.2.5 Futuro del IoT y Desarrollo con ESP32 | 17 |
| 3.3 Primer proyecto de programación con ESP32 | 18 |
| 4 Configurando un hardware de ejemplo | 20 |
| 4.1 ESP32 Wroom | 20 |
| 4.2 Placa expansión | 22 |
| 5 Ejercicios de Arduino | 24 |
| 5.1 Parpadeo con un LED | 24 |
| 5.2 Sensor de luz | 26 |
| 5.3 Sensor de temperatura y humedad | 27 |
| 5.4 Buzzer (zumbador) | 32 |
| 5.5 PIR (Sensor Infrarrojo Pasivo) | 34 |
| 5.6 Conexión a WiFi | 37 |
| 5.6.1 1. Configurar ESP32 como Estación (Conexión a un Punto de Acceso): | 37 |
| 5.6.2 2. Configurar ESP32 como Punto de Acceso: | 38 |
| 5.6.3 3. Configurar ESP32 en Modo Ad-Hoc: | 38 |

| | |
|--|-----------|
| 6 Introducción a NodeJS | 40 |
| 6.1 Instalación de NodeJS | 40 |
| 6.2 Modo interactivo | 40 |
| 6.3 Backend de ejemplo | 40 |
| 6.3.1 Paso de parámetros: | 42 |
| 6.4 Frontend | 43 |
| 7 Protegiendo el servicio | 46 |
| 7.1 1. Configuración del Proyecto | 46 |
| 7.2 2. Configuración de MySQL | 46 |
| 7.3 3. Configuración del Servidor Express | 46 |
| 7.4 4. Crear las Vistas Pug | 48 |
| 7.4.1 Contenido de views/index.pug | 48 |
| 7.4.2 Contenido de views/login.pug | 49 |
| 7.5 5. Ejecución del Proyecto | 49 |
| 8 Ejemplo: Almacenamiento de datos de temperatura y humedad en InfluxDB | 50 |
| 8.0.1 1. Instalar Paquetes Necesarios: | 52 |
| 8.0.2 2. Configurar el Nodo MQTT: | 52 |
| 8.0.3 3. Configurar el Nodo InfluxDB: | 52 |
| 8.0.4 4. Conectar los Nodos: | 53 |
| 8.0.5 5. Desplegar el Flujo: | 53 |
| 9 Apéndice | 54 |
| 9.1 Un poco de redes | 54 |
| 9.2 Bibliografía | 55 |

1 Gestión de bases de datos para el IoT

Cada vez más dispositivos llamados inteligentes están presentes en nuestras vidas, asistentes, enchufes, bombillas, sensores de puertas y ventanas, cerraduras, y un largo etcétera. Pero el internet de las cosas, el éxito del IoT no viene sólo del abaratamiento de estas tecnologías, sino por la democratización que supone tener acceso a ellas, poder programarlas, poder crearlas a un bajo coste.

Imagina que una gran empresa nos encarga un proyecto del Internet de las Cosas donde la información se genera desde sensores Arduino, un broker MQTT los va leyendo (como Mosquitto), una base de datos de series temporales (como InfluxDB) almacena la información, y una interfaz web con Node.js y Grafana nos presenta la información. ¿Me sigues?

¿Cómo lo abordarías? Por supuesto empezando por el hardware y terminando por la interfaz Web. Pero para acelerar el desarrollo usaremos software libre y concretamente una pila MING (MQTT -Mosquitto-, InfluxDB, Node-RED, Grafana). Todo el software a coste cero. Las fases del proyecto serían algo parecido a esto:

1. Configurar el Hardware:

- Conecta los sensores Arduino y asegúrate de que estén enviando datos a través del protocolo MQTT al broker Mosquitto. Puedes utilizar bibliotecas como [PubSubClient](#) en el lado del Arduino para facilitar la comunicación MQTT.

2. Configurar Mosquitto:

- Instala y configura el broker MQTT Mosquitto. Asegúrate de que esté escuchando en el puerto adecuado y que los temas (topics) estén bien definidos para tus sensores.

3. Configurar InfluxDB:

- Instala InfluxDB y crea una base de datos para almacenar tus datos de series temporales. Define las retenciones de datos según tus necesidades.

4. Desarrollar el Servidor Node.js:

- Crea un servidor Node.js para gestionar la comunicación entre InfluxDB y Grafana. Puedes utilizar bibliotecas como [express](#) para facilitar la creación de tu servidor.

5. Integrar MQTT con Node.js:

- Utiliza bibliotecas como [mqtt](#) para Node.js para suscribirte a los temas relevantes en Mosquitto y recibir los datos de los sensores.

6. Almacenar Datos en InfluxDB:

- Procesa los datos recibidos del broker MQTT y almacénalos en InfluxDB. Asegúrate de manejar la inserción de datos de manera eficiente, ya que puede haber un gran volumen de datos en un entorno IoT.

7. Desarrollar la Interfaz Web con Node.js y Grafana:

- Crea las páginas web utilizando Node.js y elige un marco de trabajo como Express para la gestión de rutas y vistas. Grafana se integrará más adelante para la visualización de datos.

8. Integrar Grafana:

- Instala y configura Grafana para conectarse a tu base de datos InfluxDB. Crea paneles y gráficos para visualizar los datos de los sensores de manera efectiva.

9. Asegurar la Comunicación:

- Considera agregar medidas de seguridad a tu aplicación, como autenticación y autorización, especialmente si la interfaz web es accesible públicamente.

10. Pruebas y Monitoreo:

- Realiza pruebas exhaustivas para asegurarte de que todos los componentes funcionen correctamente. Implementa medidas de monitoreo para supervisar el rendimiento del sistema.

11. Documentación:

- Documenta el proyecto, incluyendo cómo configurar y ejecutar cada componente. Esto facilitará el mantenimiento y la colaboración futura.

12. Despliegue:

- Despliega la aplicación en un entorno de producción, asegurándote de que todos los componentes estén configurados correctamente.

Como puedes adivinar, el elemento clave de este desarrollo es Influx, son los datos. La información (junto con el conocimiento) es la riqueza de las empresas de nuestro sector.

Pero no sólo vamos a necesitar una base de datos timeseries, también necesitaremos otra para la gestión de identidades de nuestros usuarios. Un ejemplo de base de datos para gestión de logins puede ser Redis. Otra opción muy interesante que ya conocemos es MySQL.

1.1 Redis

Redis es una base de datos en memoria (in-memory database) de código abierto que se utiliza como almacén de estructuras de datos clave-valor. Es extremadamente rápido y eficiente, ya que mantiene

todos sus datos en la memoria principal en lugar de en el disco, lo que permite un acceso rápido y tiempos de respuesta casi instantáneos. Aquí te explico algunas características y usos comunes de Redis:

1.1.1 Características Principales de Redis:

1. Almacenamiento de Datos en Memoria:

- Todos los datos se mantienen en la RAM, lo que permite operaciones de lectura y escritura muy rápidas.

2. Estructuras de Datos Soportadas:

- Redis no solo almacena simples pares clave-valor, sino que también admite diversas estructuras de datos como strings, hashes, listas, conjuntos y conjuntos ordenados.

3. Persistencia Opcional:

- Aunque Redis se destaca por ser una base de datos en memoria, ofrece opciones de persistencia para guardar datos en disco, lo que facilita la recuperación después de reinicios.

4. Atomicidad de Operaciones:

- Las operaciones en Redis son atómicas, lo que significa que son ejecutadas en su totalidad o no se ejecutan en absoluto.

5. Soporte para Transacciones:

- Redis admite transacciones, lo que permite agrupar varias operaciones y ejecutarlas como una unidad.

1.1.2 Usos Comunes de Redis:

1. Caché en Memoria:

- Una de las aplicaciones más comunes de Redis es como un sistema de almacenamiento en caché en memoria. Almacena datos que son costosos de calcular o recuperar, lo que mejora significativamente los tiempos de respuesta.

2. Colas de Mensajes:

- Redis es utilizado para implementar colas de mensajes, donde las aplicaciones pueden enviar y recibir mensajes entre sí de manera eficiente.

3. Sesiones de Usuario:

- Almacenar información de sesión de usuario en Redis es una práctica común para aplicaciones web, ya que proporciona tiempos de acceso rápidos y puede manejar grandes volúmenes de sesiones de usuario.

4. **Conteo de Visitas y Estadísticas en Tiempo Real:**

- Debido a su velocidad, Redis es ideal para contadores de visitas y estadísticas en tiempo real, como el seguimiento de usuarios en un sitio web.

5. **Listas y Colas:**

- Las estructuras de datos tipo lista en Redis permiten la implementación de colas y sistemas de mensajería pub/sub.

6. **Gestión de Sesiones en Juegos en Tiempo Real:**

- En entornos de juegos en tiempo real, Redis se utiliza para gestionar información del estado del juego y sesiones de usuario.

7. **Bloqueo de Recursos Compartidos:**

- Redis proporciona primitivas de bloqueo que se pueden utilizar para implementar patrones de bloqueo en entornos distribuidos.

Redis es versátil y se utiliza en una variedad de casos de uso. Su velocidad y flexibilidad lo hacen adecuado para aplicaciones donde se requiere acceso rápido y eficiente a los datos, y donde la pérdida de datos en caso de reinicio no es crítica.

1.2 InfluxDB

InfluxDB es una base de datos de series temporales diseñada específicamente para el manejo eficiente de datos temporales, como los generados por dispositivos de Internet de las cosas (IoT), monitoreo de servidores, aplicaciones de análisis de datos en tiempo real, entre otros.

1.2.1 Características Principales:

1. **Modelo de Datos de Series Temporales:**

- Almacena datos temporales con un enfoque en la eficiencia y rendimiento.
- Utiliza un lenguaje de consulta llamado InfluxQL para interactuar con los datos.

2. **Arquitectura:**

- Distribuida y altamente escalable.

- Diseñada para admitir grandes volúmenes de datos con escrituras y consultas rápidas.

3. **Retention Policies:**

- Permite definir políticas de retención para gestionar automáticamente la expiración de datos.

4. **Soporte para lenguajes y herramientas:**

- Admite diversos lenguajes de programación y tiene integraciones con herramientas populares como Grafana para visualización.

1.2.2 Alternativas:

1. **OpenTSDB:**

- Base de datos de series temporales que se ejecuta sobre HBase.
- Escalabilidad y rendimiento, adecuada para grandes volúmenes de datos temporales.

2. **Prometheus:**

- Sistema de monitoreo y alerta diseñado para ambientes dinámicos.
- Almacena datos de series temporales y utiliza el lenguaje de consulta PromQL.

3. **Graphite:**

- Enfocado en la recopilación y representación de datos de rendimiento.
- Usa Whisper como backend para almacenamiento a largo plazo.

4. **Cassandra:**

- Base de datos NoSQL distribuida, escalable y diseñada para manejar grandes cantidades de datos.
- No está diseñada específicamente para series temporales, pero puede adaptarse.

5. **MongoDB:**

- Base de datos NoSQL que admite almacenamiento de datos de series temporales.
- Ofrece flexibilidad en el esquema y puede ser escalado horizontalmente.

6. **Elasticsearch:**

- Originalmente diseñado para búsqueda y análisis de texto completo, pero puede utilizarse para datos de series temporales.
- Escalabilidad y capacidades de búsqueda avanzadas.

La elección entre estas alternativas depende de los requisitos específicos de tu aplicación, como el volumen de datos, la complejidad de las consultas, la escalabilidad, entre otros factores. Incluso en algunos casos, podrías combinar varias tecnologías para satisfacer diferentes necesidades dentro de tu arquitectura de IoT.

2 Preparación del entorno

2.1 La pila MING

Como punto de partida usaremos el siguiente repositorio para montar nuestro ecosistema de servicios necesario para el proyecto: <https://github.com/emield12/docker-influxdb-grafana-nodered-mqtt>. Recuerda que existe una copia en local en el repositorio de estos apuntes, en la carpeta **iot-stack**.

Arrancamos los contenedores con (en Windows abrimos una Git Bash terminal en la carpeta **iot-stack**, previamente hemos abierto Docker Desktop para que arranque el servicio):

```
1 bash run.sh
```

Para parar los contenedores:

```
1 docker-compose down
```

Una vez creados, podemos arrancarlos todos a la vez desde esta misma carpeta con:

```
1 docker-compose start
```

Ya podemos abrir la página de InfluxDB: <http://localhost:8086/>. Pulsemos en “Get Started”. Ahora configuramos InfluxDB con estos parámetros:

| Parámetro | Valor |
|---------------------------|-------------|
| Username | root |
| Password | Secreto_123 |
| Confirm Password | Secreto_123 |
| Initial Organization Name | IES VDC |
| Initial Bucket Name | Ambiente |

En la siguiente pantalla, nos aseguramos de copiar el token y guardarla en un fichero de texto. He creado una carpeta “**credentials**” que añado al **.gitignore** y ahora en un archivo influx.token guardo el token copiado.

El **.gitignore** quedaría así de momento:

```
1 credentials/
2 node_modules/
```

La barra al final quiere decir que en cualquier subcarpeta del proyecto que contenga una carpeta con estos nombres, será ignorada y no subida al repositorio.

Para confirmar los cambios y subirlos al repo, recuerda que puedes hacer (previamente debiste haber creado el repositorio en Github/Gitlab):

```
1 git add .
2 git commit -m "creado gitignore"
3 git push origin main
```

Si nos habíamos equivocado y añadido el “node_modules” y el “credentials” puedo hacer:

```
1 git rm --cached -r .
```

Para empezar de nuevo.

De los tres botones que salen en la ventana de bienvenida, ahora vamos a hacer el tutorial pulsando en “Quick Start”.

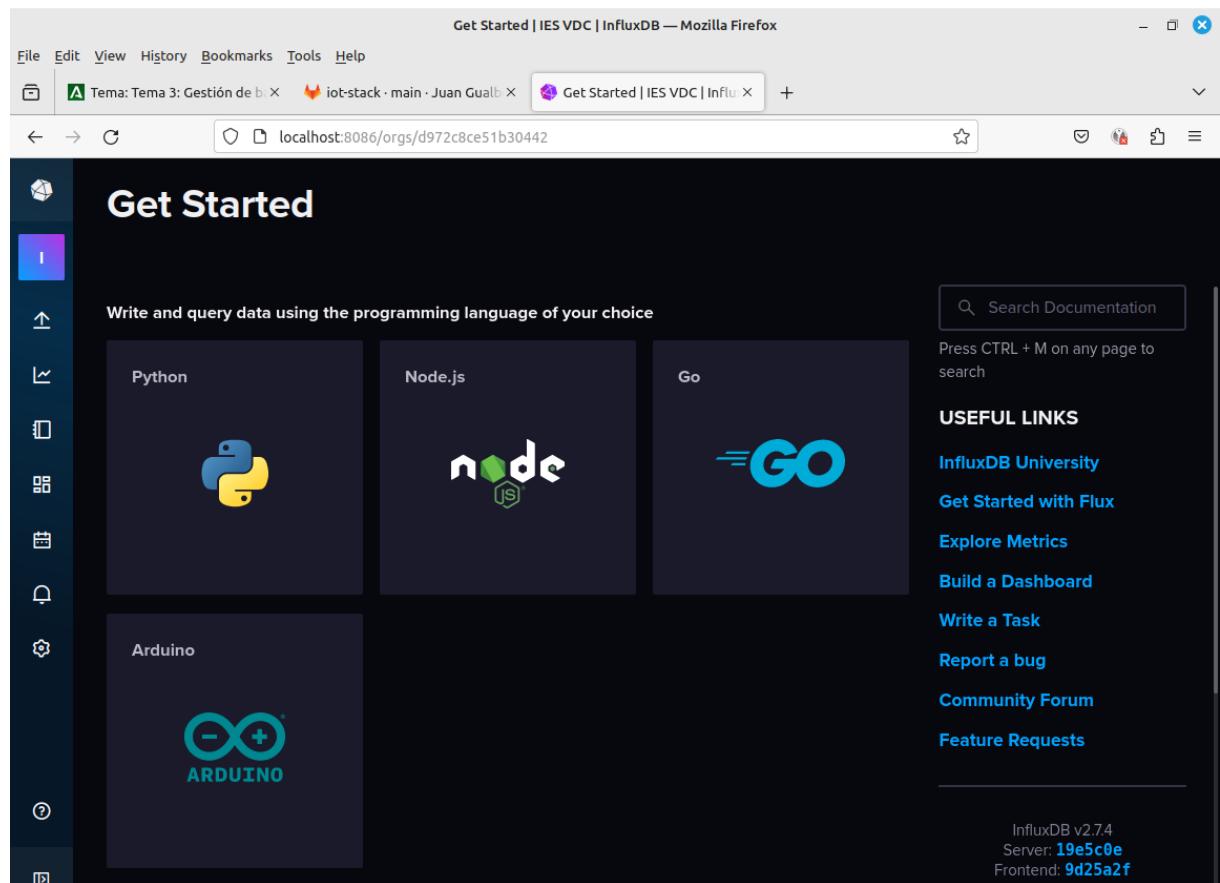


Figura 1: Ventana de “Get Started”

En la ventana pulsamos en **Node.js**.

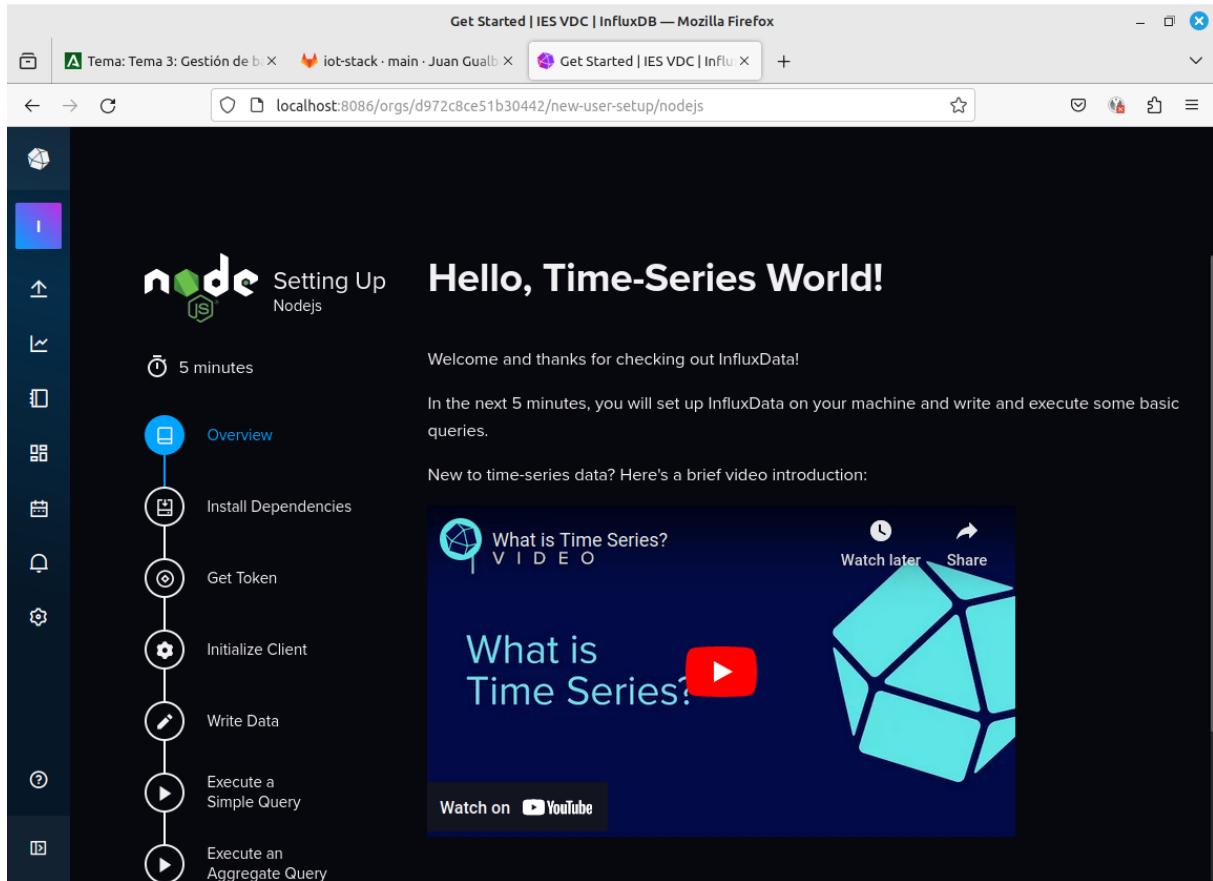


Figura 2: Ventana de bienvenida de NodeJS con el Vídeo de Youtube

Creamos en la raíz de nuestro proyecto una carpeta “**node**”, en esa carpeta vamos a crear el código y a completar el tutorial.

Paso 1: Instalar las dependencias

```
1 npm install --save @influxdata/influxdb-client
```

Paso 2: variables de entorno

En Windows usamos git bash para ejecutar este comando:

```
1 export INFLUXDB_TOKEN=$(cat ./credentials/influx.token)
```

Paso 3: Conexión a la BBDD

Ejecuto el comando “node” desde la carpeta que hicimos npm install:

```

1 const {InfluxDB, Point} = require('@influxdata/influxdb-client')
2 const token = process.env.INFLUXDB_TOKEN
3 const url = 'http://localhost:8086'
4 const client = new InfluxDB({url, token})

```

Paso 4: inserción de datos

Copia el contenido de este ejemplo a un archivo con extensión **js** y puedes ejecutarlo desde NodeJS en la carpeta donde hicimos el comando “npm install” para que estén todas las librerías necesarias.

```

1 const {InfluxDB, Point} = require('@influxdata/influxdb-client')
2
3 const org= 'IES VDC';
4 let bucket = `ambiente`
5 const token = process.env.INFLUXDB_TOKEN
6 const url = 'http://localhost:8086'
7
8 const client = new InfluxDB({url, token})
9
10
11 let writeClient = client.getWriteApi(org, bucket, 'ns')
12
13 for (let i = 0; i < 5; i++) {
14   let point = new Point('measurement1')
15     .tag('tagname1', 'tagvalue1')
16     .intField('field1', i)
17
18   void setTimeout(() => {
19     writeClient.writePoint(point)
20   }, i * 1000) // separate points by 1 second
21
22   void setTimeout(() => {
23     writeClient.flush()
24   }, 5000)
25 }

```

Paso 5: Consulta a la BBDD

```

1 const {InfluxDB, Point} = require('@influxdata/influxdb-client')
2
3 const org= 'IES VDC';
4 const token = process.env.INFLUXDB_TOKEN
5 const url = 'http://localhost:8086'
6
7 const client = new InfluxDB({url, token})
8
9 let queryClient = client.getQueryApi(org)
10
11 let fluxQuery = `from(bucket: "ambiente")
12   |> range(start: -30d)
13   |> filter(fn: (r) => r._measurement == "measurement1")`
```

```
14
15
16 queryClient.queryRows(fluxQuery, {
17   next: (row, tableMeta) => {
18     const tableObject = tableMeta.toObject(row)
19     console.log(tableObject)
20   },
21   error: (error) => {
22     console.error('\nError', error)
23   },
24   complete: () => {
25     console.log('\nSuccess')
26   },
27 })
```

3 Introducción al IoT con ESP32

3.1 Objetivos:

En este apartado vamos a ver lo siguiente:

1. Introducción a IoT y ESP32

- Definición de IoT y su importancia.
- Breve introducción al ESP32: características, ventajas y casos de uso comunes.

2. Configuración del Entorno de Desarrollo

- Instalación de la herramienta de desarrollo (IDE) para ESP32 (por ejemplo, PlatformIO en VSCode).
- Configuración del entorno de desarrollo.
- Conexión y configuración del ESP32.

3. Programación Básica con Arduino

- Estructura básica de un programa Arduino para ESP32.
- Manipulación de pines: entrada y salida digital/análoga.
- Lectura de sensores básicos (puede ser un sensor de temperatura, por ejemplo).

4. Comunicación Inalámbrica y Conectividad

- Configuración y uso de Wi-Fi en el ESP32.
- Enviar y recibir datos a través de Wi-Fi.
- Breve introducción a MQTT para comunicación IoT.

5. Sensores y Actuadores Avanzados

- Integración de sensores más avanzados (por ejemplo, sensores de movimiento, cámaras).
- Control de actuadores (por ejemplo, relés, motores).

6. Manejo de Energía y Optimización

- Estrategias para gestionar el consumo de energía.
- Uso de modos de bajo consumo en ESP32.

7. Seguridad en IoT

- Breve introducción a los principios de seguridad en IoT.
- Consejos para proteger dispositivos y datos.

8. Proyecto Práctico: Construcción de una Aplicación IoT Simple

- Desarrollo de un proyecto práctico que integre varios conceptos aprendidos durante la clase.
- Solución de problemas y resolución de dudas.

9. Recursos y Siguientes Pasos

- Recursos adicionales para aprender más.
- Siguientes pasos para el desarrollo de proyectos más avanzados.
- Preguntas y respuestas.

3.1.1 Metodología:

- **Teoría y Demostración:** Comienza con una breve explicación teórica de cada concepto, seguido de demostraciones prácticas en vivo.
- **Práctica Guiada:** Proporciona ejercicios prácticos después de cada sección para que los participantes puedan aplicar lo que han aprendido.
- **Proyecto Final:** La masterclass debe culminar con la construcción de un proyecto práctico que integre múltiples aspectos del desarrollo con ESP32.
- **Participación Activa:** Fomenta preguntas y participación activa. Puedes incorporar discusiones grupales para compartir experiencias y enfoques.
- **Recursos Adicionales:** Proporciona a los participantes materiales de lectura, tutoriales y enlaces a recursos en línea para el aprendizaje continuo.

3.2 Introducción a IoT y ESP32

3.2.1 Internet de las Cosas (IoT)

3.2.1.1 Definición

- **IoT:** Interconexión de dispositivos físicos a través de Internet.
-
- **Dispositivos:** Desde electrodomésticos hasta sensores industriales.

3.2.1.2 Importancia

- **Transformación:** Impacto en diversas industrias.
- **Eficiencia:** Mejora en procesos y servicios.

3.2.2 ESP32

3.2.2.1 Introducción al ESP32

- **Microcontrolador:** Bajo costo y consumo de energía.
- **Conectividad:** Wi-Fi y Bluetooth integrados.
- **Versatilidad:** Programación con Arduino y otros entornos.

3.2.2.2 Características Clave

- **Dual-core y velocidad de reloj.**
- **Conectividad Wi-Fi y Bluetooth.**
- **Amplia variedad de puertos de entrada/salida.**
- **Flexibilidad para programar con Arduino y otros entornos.**

3.2.3 Casos de Uso Comunes

3.2.3.1 Ejemplos Prácticos

- **Monitoreo y Control Remoto.**
- **Sensores Ambientales.**
- **Automatización del Hogar.**
- **Proyectos de IoT Educativos.**

3.2.4 Desafíos y Consideraciones en IoT

3.2.4.1 Seguridad La seguridad es un tema crucial en el IoT debido a la amplia variedad de dispositivos conectados a la red. Los dispositivos IoT pueden ser vulnerables a ataques si no se implementan medidas de seguridad adecuadas. El ESP32 aborda este desafío proporcionando funciones de seguridad incorporadas, como la capacidad de establecer conexiones seguras a través de Wi-Fi (WPA2) y la posibilidad de implementar algoritmos de cifrado.

Al estar conectados a Internet, debemos proteger siempre todos nuestros dispositivos detrás de un cortafuegos, en una red exclusiva, nunca exponerlos directamente ni en redes compartidas - especialmente WiFi - con otros usuarios.

3.2.4.2 Escalabilidad Al desarrollar soluciones IoT, es importante considerar la capacidad de escalar desde un prototipo pequeño hasta una implementación a gran escala. El ESP32 es versátil y puede adaptarse a una amplia variedad de proyectos, desde pequeños dispositivos de prototipo hasta implementaciones más grandes.

3.2.4.3 Consumo de Energía En muchos casos, los dispositivos IoT deben funcionar con fuentes de energía limitadas, como baterías. La gestión eficiente del consumo de energía es esencial para garantizar una vida útil prolongada de la batería. El ESP32 aborda este desafío al ofrecer modos de bajo consumo que permiten reducir significativamente el consumo de energía cuando el dispositivo no está en pleno funcionamiento, lo que es crucial para aplicaciones alimentadas por batería.

Estos desafíos y consideraciones son solo algunos ejemplos de los muchos factores que los desarrolladores deben tener en cuenta al trabajar en proyectos de IoT. Abordar estos aspectos desde el principio puede ayudar a crear soluciones más robustas y eficientes. La comprensión de cómo el ESP32 maneja estos desafíos es esencial para aprovechar al máximo sus capacidades en el desarrollo de aplicaciones IoT.

Claro, en el punto “5. Futuro del IoT y Desarrollo con ESP32” se trata de proporcionar una visión general de las tendencias emergentes en el ámbito del Internet de las cosas (IoT) y cómo el ESP32 está posicionado para abordar esas tendencias. Aquí hay una explicación más detallada:

3.2.5 Futuro del IoT y Desarrollo con ESP32

3.2.5.1 Tendencias Emergentes El Internet de las cosas es un campo dinámico que evoluciona constantemente. Algunas tendencias emergentes que podrían afectar el desarrollo de aplicaciones IoT en el futuro incluyen:

- **Edge Computing:** El procesamiento de datos en el borde de la red para reducir la latencia y mejorar la eficiencia.
- **5G y Conectividad Mejorada:** La implementación generalizada de redes 5G que proporcionan velocidades de conexión más rápidas y una mayor capacidad.
- **Inteligencia Artificial y Machine Learning en Dispositivos IoT:** La integración de capacidades de inteligencia artificial y aprendizaje automático directamente en dispositivos IoT para un procesamiento más rápido y decisiones más inteligentes.

3.2.5.2 Comunidad y Recursos Para el desarrollo continuo en IoT y específicamente con el ESP32, es vital destacar la importancia de la comunidad de desarrolladores y disponer de recursos para el aprendizaje continuo. El ESP32 dispone de:

- **Comunidades en línea:** Foros, grupos de discusión y redes sociales donde los desarrolladores pueden compartir conocimientos, hacer preguntas y colaborar.
- **Documentación y Tutoriales:** Recursos escritos y multimedia que ofrecen guías detalladas sobre el uso del ESP32 y cómo abordar desafíos comunes.

- **Proyectos de Código Abierto:** La participación en proyectos de código abierto relacionados con el ESP32 puede ser una excelente manera de aprender y contribuir a la comunidad.

3.3 Primer proyecto de programación con ESP32

Vamos cuáles serían los primeros pasos para crear un programa simple para el ESP32, compilarlo y cargarlo en la placa. Para este ejemplo, usaremos el entorno de desarrollo PlatformIO en Visual Studio Code, que es una opción popular para programar el ESP32.

1. Instalar Visual Studio Code y PlatformIO:

- Descarga e instala Visual Studio Code.
- Abre Visual Studio Code y ve a la pestaña de extensiones. Busca e instala la extensión “PlatformIO IDE”. Pulsa en la extensión en la barra de la izquierda para forzar la descarga de todos los componentes. Reinicia VS Code cuando se pida.

2. Crear un Nuevo Proyecto:

- En Visual Studio Code, ve a la pestaña de “PlatformIO” en el menú lateral y selecciona “New Project”.
- Selecciona el entorno “Espressif 32” y elige el modelo específico de tu placa ESP32.

3. Escribir el Programa:

- Abre el archivo `src/main.cpp`. Este archivo contiene el código principal de tu programa.

```
1 #include <Arduino.h>
2
3 void setup() {
4     // Inicialización, se ejecuta una vez al inicio
5     Serial.begin(115200);
6     Serial.println("Hola, ESP32!");
7 }
8
9 void loop() {
10    // Código que se ejecuta repetidamente
11    Serial.println("Hola de nuevo!");
12    delay(1000); // Espera 1 segundo
13 }
```

Este programa simplemente imprime mensajes a través del puerto serie cada segundo.

4. Compilar el Programa:

- Guarda tus cambios y presiona el icono de “Check” en la barra inferior para compilar el programa.

5. Configurar el Puerto de la Placa:

- Conecta tu placa ESP32 al ordenador mediante un cable USB.
- Selecciona el puerto correcto en la esquina inferior derecha de Visual Studio Code.

6. Cargar el Programa:

- Presiona el icono de “flecha” (Upload) en la barra inferior para cargar el programa en la placa.

7. Verificar la Salida:

- Abre el monitor serial (puedes hacerlo desde la pestaña de “PlatformIO” en Visual Studio Code) para ver la salida del programa.

4 Configurando un hardware de ejemplo

4.1 ESP32 Wroom

La placa de desarrollo o DEVKIT V1 NodeMCU-32 es una herramienta muy potente para el prototipado rápido de proyectos con IoT (Internet de la cosas). Integra en una placa el SoM ESP-WROOM-32 que tiene como base al SoC ESP32, el convertidor USB-serial Ch340G necesario para programar por USB el ESP32, reguladores de voltaje y leds indicadores.

La plataforma ESP32 es la evolución del ESP8266 mejorando sus capacidades de comunicación y procesamiento computacional. A nivel de conectividad permite utilizar diversos protocolos de comunicación inalámbrica como: WiFi, Bluetooth y BLE.

En cuanto a procesamiento su CPU 32-bit de dos núcleos de hasta 240Mhz que se pueden controlar independientemente. Además incluye internamente una gran cantidad de periféricos para la conexión con: sensores táctiles capacitivos, sensor de efecto Hall, amplificadores de bajo ruido, interfaz para tarjeta SD, Ethernet, SPI de alta velocidad, UART, I2S e I2C. Aplicado en Mini Servidores Web, Procesamiento digital, Webcams, Cámara IP, Robótica móvil, Domótica y más.

NodeMCU-32 está diseñada especialmente para trabajar montado en protoboard. Puede alimentarse directamente del puerto micro-USB o utilizando una fuente externa de 5V o 3V pues posee regulador de voltaje en placa, recomendamos utilizar una fuente de 5VDC/1A y colocar un capacitor de 100uF en paralelo con la fuente de alimentación para filtrar los picos de corriente.

Los pines de entradas/salidas (GPIO) trabajan a 3.3V por lo que para conexión a sistemas de 5V es necesario utilizar conversores de nivel como: Conversor de nivel 3.3-5V 4CH o Conversor de nivel bidireccional 8CH - TXS0108E.

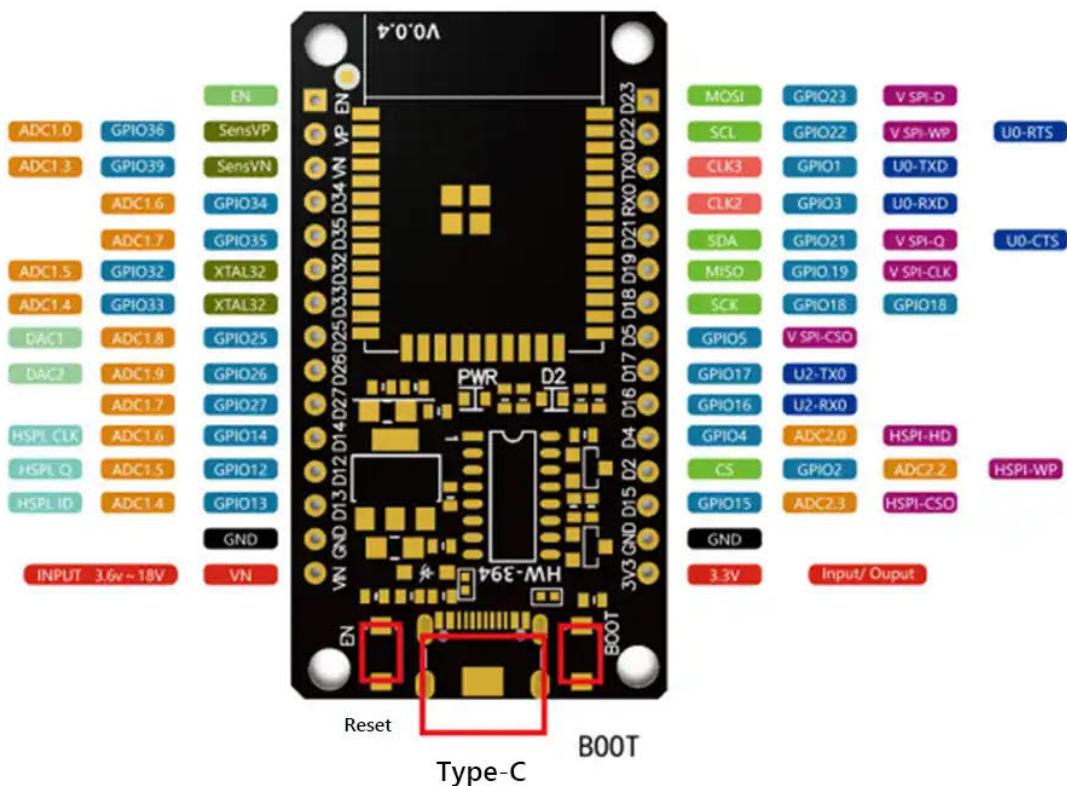
El SoC(System On a Chip) ESP32 de Espressif Systems es la evolución del ESP8266, diseñado para superar a su antecesor en capacidad de procesamiento y conectividad, integra un potente microcontrolador con arquitectura de 32 bits, conectividad Wi-Fi y Bluetooth. El SoM(System on Module) ESP-WROOM-32 fabricado por Espressif integra en un módulo el SoC ESP32, memoria FLASH, cristal oscilador y antena WiFi en PCB.

La plataforma ESP32 permite el desarrollo de aplicaciones en diferentes lenguajes de programación, frameworks, librerías y recursos diversos. Los más comunes a elegir son: Arduino(en lenguaje C++), MicroPython, LUA, Esp-idf(Espressif IoT Development Framework) desarrollado por el fabricante del chip, Simba Embedded Programming Platform(en lenguaje Python), RTOS's (como Zephyr Project, Mongoose OS, NuttX RTOS), Javascript (Espruino, Duktape, Mongoose JS), Basic. Al trabajar dentro del entorno Arduino podremos utilizar un lenguaje de programación conocido y hacer uso de un IDE sencillo de utilizar, además de hacer uso de toda la información sobre proyectos y librerías disponibles en internet.

La comunidad de usuarios de Arduino es muy activa y da soporte a plataformas como el ESP32 y ESP8266. Dentro de las principales placas de desarrollo o módulos basados en el ESP32 tenemos: ESP32-WROOM-32, NodeMCU-32 ESP32 y ESP32-CAM y de la familia ESP8266 tenemos: ESP-01, ESP-12E, Wemos D1 mini y NodeMCU v2.

ESPECIFICACIONES Y CARACTERÍSTICAS

- Voltaje de Alimentación (USB): 5V DC
- Voltaje de Entradas/Salidas: 3.3V DC
- Placa: ESP32 DEVKIT V1 (Espressif)
- SoM: ESP-WROOM-32 (Espressif)
- SoC: ESP32 (ESP32-D0WDQ6)
- CPU: Dual-Core Tensilica Xtensa LX6 (32 bit)
- Frecuencia de Reloj: hasta 240Mhz
- Desempeño: Hasta 600 DMIPS
- Procesador secundario: Permite hacer operaciones básica en modo de ultra bajo consumo Wifi: 802.11 b/g/n/e/i (802.11n @ 2.4 GHz hasta 150 Mbit/s)
- Bluetooth: v4.2 BR/EDR and Bluetooth Low Energy (BLE) Memoria: 448 KByte ROM 520 KByte SRAM 16 KByte SRAM in RTC QSPI Flash/SRAM, 4 MBytes
- Pines: 30
- Pines Digitales GPIO: 24 (Algunos pines solo como entrada)
- Pines PWM: 16
- Pines Analógicos ADC: 18 (3.3V, 12bit: 4095, tipo SAR, ganancia programable)
- Conversor Digital a Analógico DAC: 2 (8bit)
- UART: 2
- Chip USB-Serial: CH340G
- Antena en PCB
- Seguridad: Estandares IEEE 802.11 incluyendo WFA, WPA/WPA2 and WAPI 1024-bit OTP, up to 768-bit for customers
- Aceleración criptográfica por hardware: AES, HASH (SHA-2), RSA, ECC, RNG
- Dimensiones: 55*28 mm

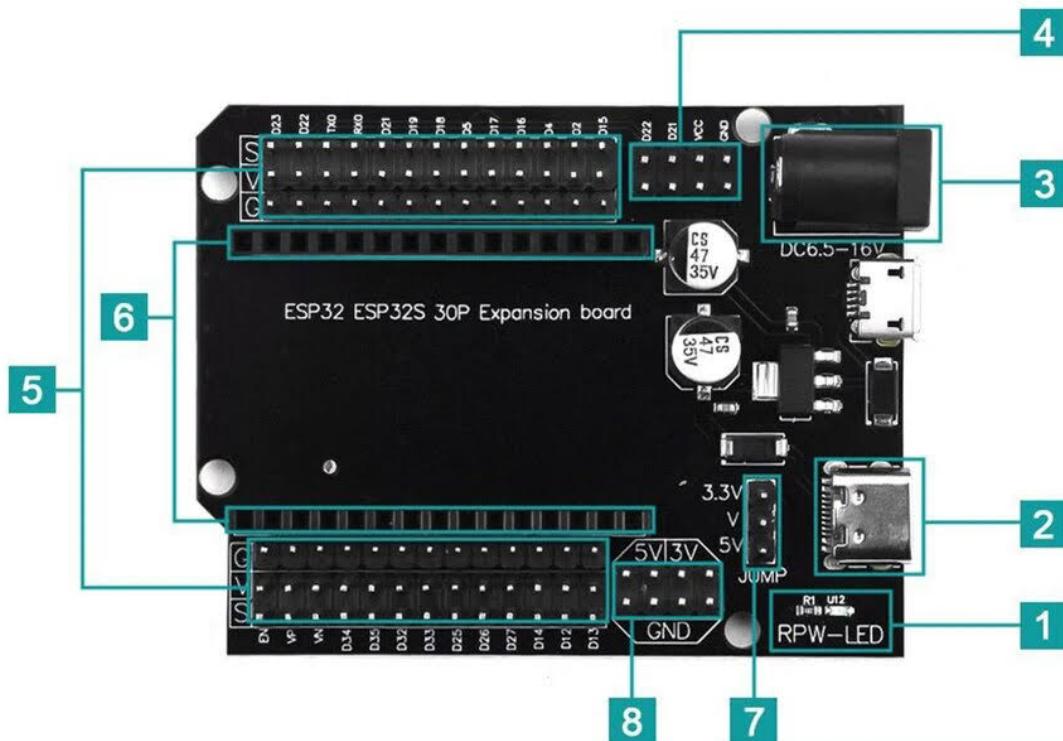
**Figura 3:** Entradas y salidas del ESP32

4.2 Placa expansión

Descripción de la interfaz:

1. Indicador de PWR
2. Interfaz de entrada de fuente de alimentación USB de 5 V
3. Puerto de entrada de alimentación de 6.5-16 V CC

4. Interfaz extendida I2C (dos grupos)
5. Interfaz de extensión IO de placa de desarrollo (interfaz GVS, la línea G es de tierra, la línea V de voltaje y la línea S de salida de los pines indicados a un lado de la línea)
6. Interfaz de instalación de la placa de desarrollo (ESP32-30 Pines)
7. Interfaz de salida de 5 V/3.3 V
8. Interfaz Jumper [seleccionar V, voltaje de 5 V o 3.3 V]



Interface description:

- | | |
|---|--|
| 1. Power indicator light PWR | 5. Development board extension interface (GVS interface) |
| 2. USB5V power supply input interface | 6. Development board installation interface |
| 3. DC6.5-16V power supply input interface | 7.5V/3.3V output interface |
| 4. I2C extension interface (two sets) | 8. Jump interface (select 5V or 3V voltage) |

Figura 4: Placa de expansión usada

5 Ejercicios de Arduino

Para esta toma de contacto, descargamos e instalamos **ARDUINO IDE**. En el caso de Linux es simplemente descargar y descomprimir, aunque debes asegurarte que tu usuario está en el grupo `serial` para poder leer el dispositivo `/dev/ttyUSB0`.

Para añadir placas adicionales de ESP32 como la que usamos nosotros (la WROOM-32), tenemos que ir al menú archivo, preferencias y en el administrador de placas adicionales, añadimos estas líneas:

```
1 https://espressif.github.io/arduino-esp32/package_esp32_index.json
2 https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/
    package_esp32_index.json
```

Ahora seleccionamos nuestra placa, **ESP32 DEV MODULE**, le decimos que está en `/dev/ttyUSB0` (en linux) y ya podemos empezar a programar: Menú Tools → Boards → ESP32 → ESP32 Dev Module.

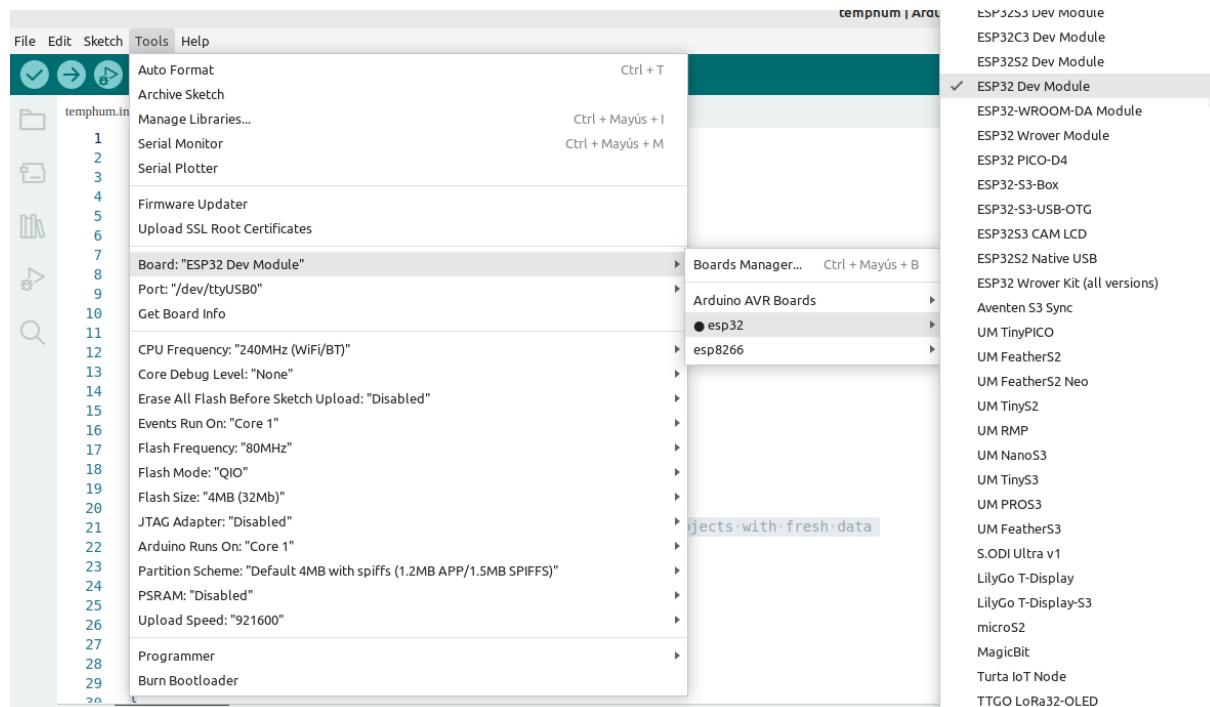


Figura 5: Selección de placa desde el IDE

5.1 Parpadeo con un LED

Para encender y apagar un led, vamos a configurar la señal 4 como salida.

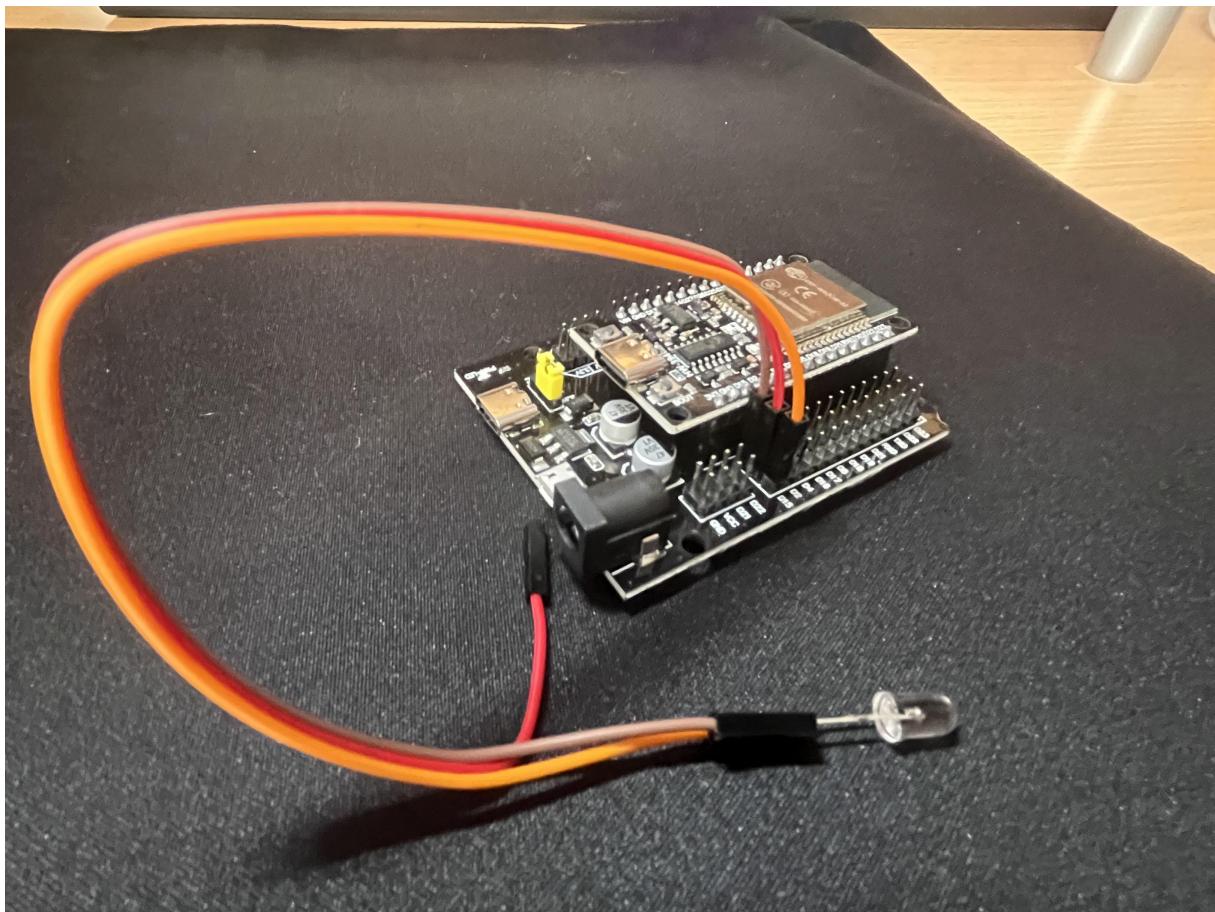


Figura 6: Ejemplo de cómo conectar el led.

```
1 // the setup function runs once when you press reset or power the board
2 void setup() {
3     // initialize digital pin GPIO18 as an output.
4     pinMode(18, OUTPUT);
5 }
6
7 // the loop function runs over and over again forever
8 void loop() {
9     digitalWrite(18, HIGH); // turn the LED on
10    delay(500);           // wait for 500 milliseconds
11    digitalWrite(18, LOW); // turn the LED off
12    delay(500);           // wait for 500 milliseconds
13 }
```

5.2 Sensor de luz

Conectamos el sensor de luz al puerto D34, pero lo vamos a usar en analógico. El cable que va a S en la placa de expansión, la conectamos al pin A0 del sensor.

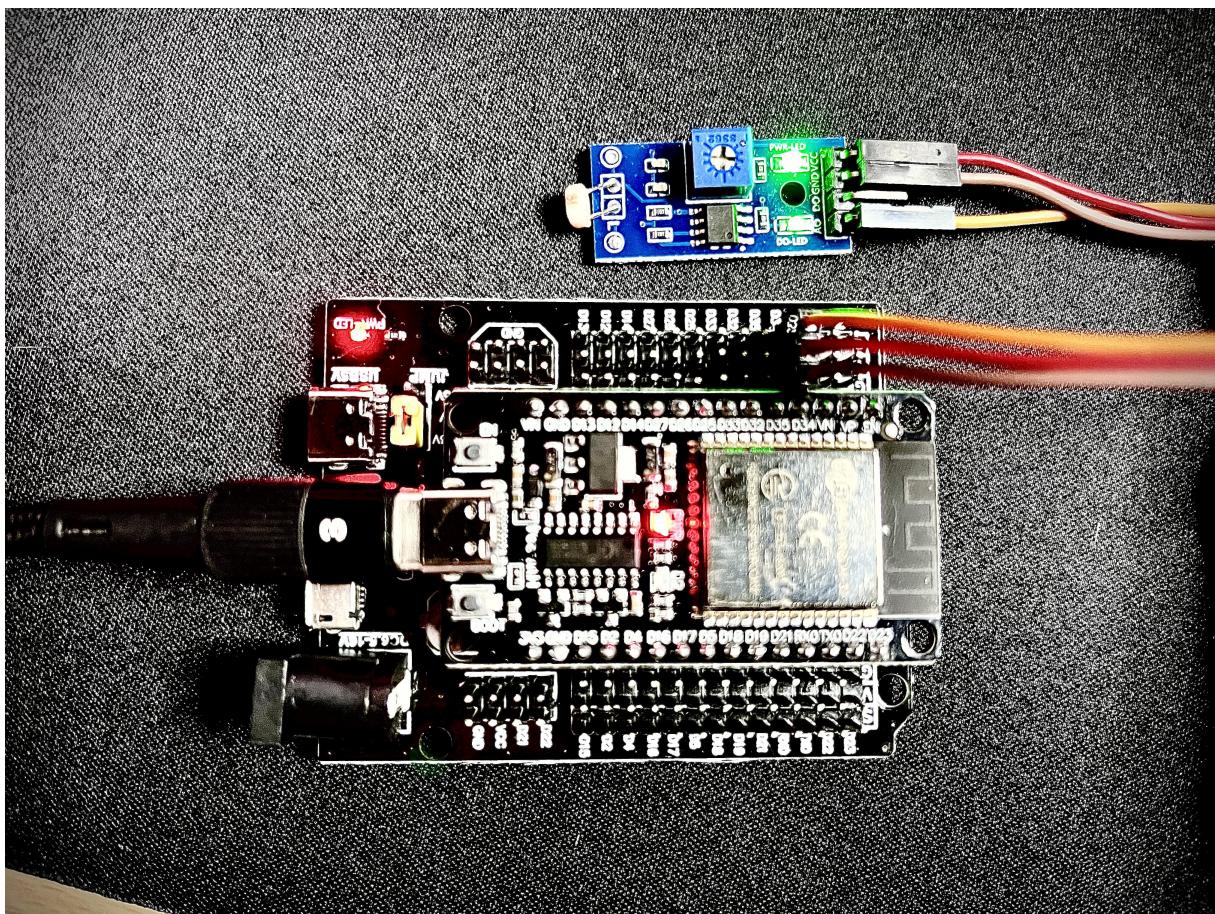


Figura 7: Ejemplo de cómo conectar el sensor de luz.

```
1 int count = 1;
2
3 void setup() {
4     Serial.begin(9600);
5 }
6
7 void loop() {
8     // reads the input on analog pin A0 (value between 0 and 1023)
9     int analogValue = analogRead(34);
10    Serial.print(count);
11    count++;
12    Serial.print(" th valor analógico = ");
13    Serial.print(analogValue); // the raw analog reading
```

```
14
15  if (analogValue < 100) {
16      Serial.println(" - Muy brillante");
17  } else if (analogValue < 200) {
18      Serial.println(" - Brillante");
19  } else if (analogValue < 500) {
20      Serial.println(" - Hay luz");
21  } else if (analogValue < 800) {
22      Serial.println(" - Casi sin luz");
23  } else {
24      Serial.println(" - Oscuridad");
25  }
26
27  delay(1000);
28 }
```

5.3 Sensor de temperatura y humedad

Para poder usar el sensor de temperatura y humedad (AHT10) y el sensor de presión atmosférica (BMP280) necesitamos añadir unas librerías desde el gestor de librerías del IDE. Abrimos el menú contextual de la izquierda que tiene como unos libros dibujados y buscamos “ahtx”. El primer resultado es la librería **Adafruit AHTX0** que es la que vamos a usar. También es posible hacerlo desde Sketch → Include Library → Manage Libraries. Seguidamente añadimos también **Adafruit Unified Sensor**.

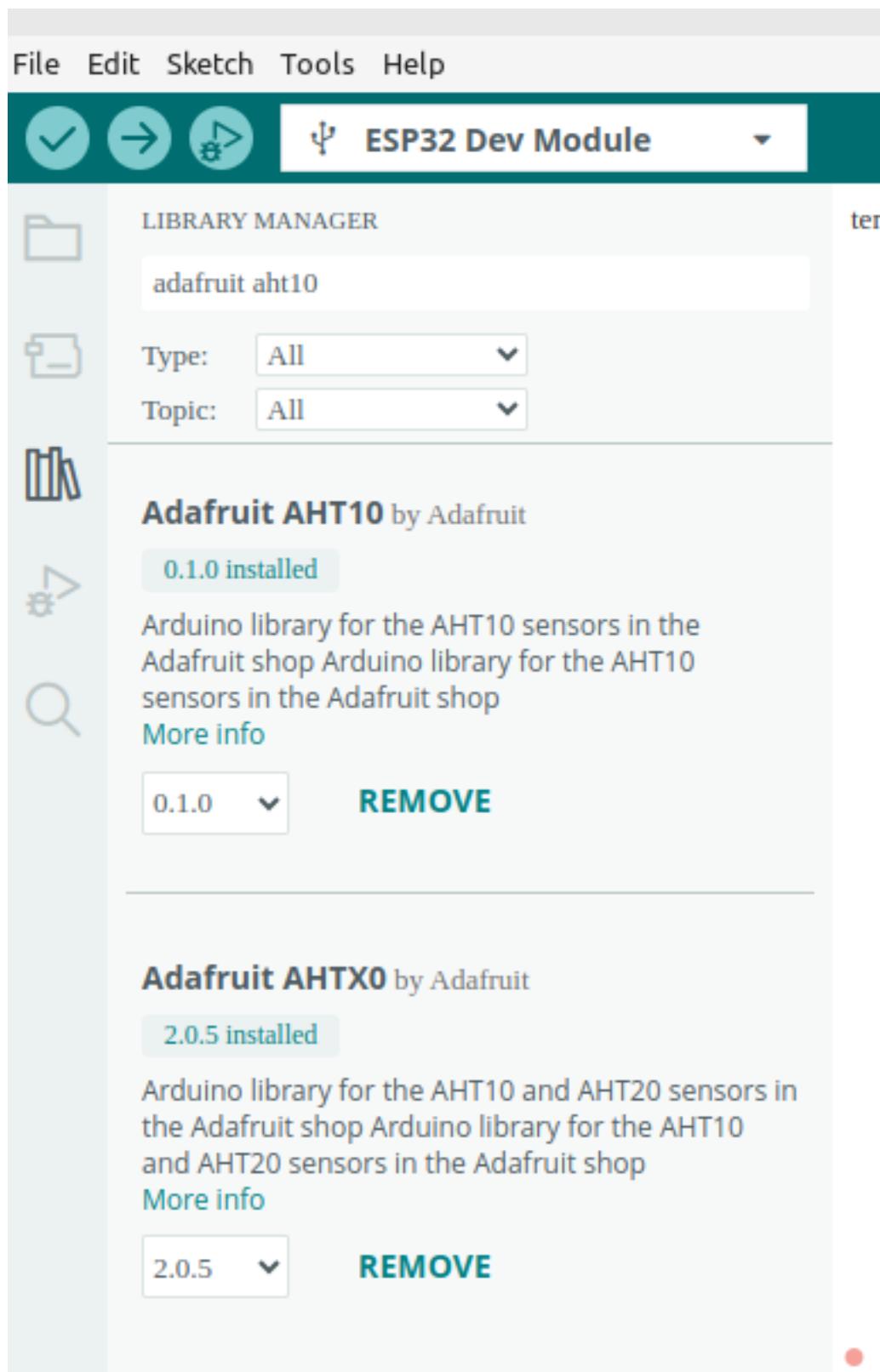


Figura 8: Cómo instalar librerías adicionales desde el menú contextual de la izquierda

Estos chips usan el I2C para comunicarse con el controlador, luego necesitaremos conectarlo a los pines SDA y SCL que están en D21 y D22 en nuestro ESP32 WROOM como vimos en la imagen de pinout del microcontrolador en el apartado anterior. Conectamos VCC y GND cada uno en su lugar en el chip y en la placa a V y G respectivamente.

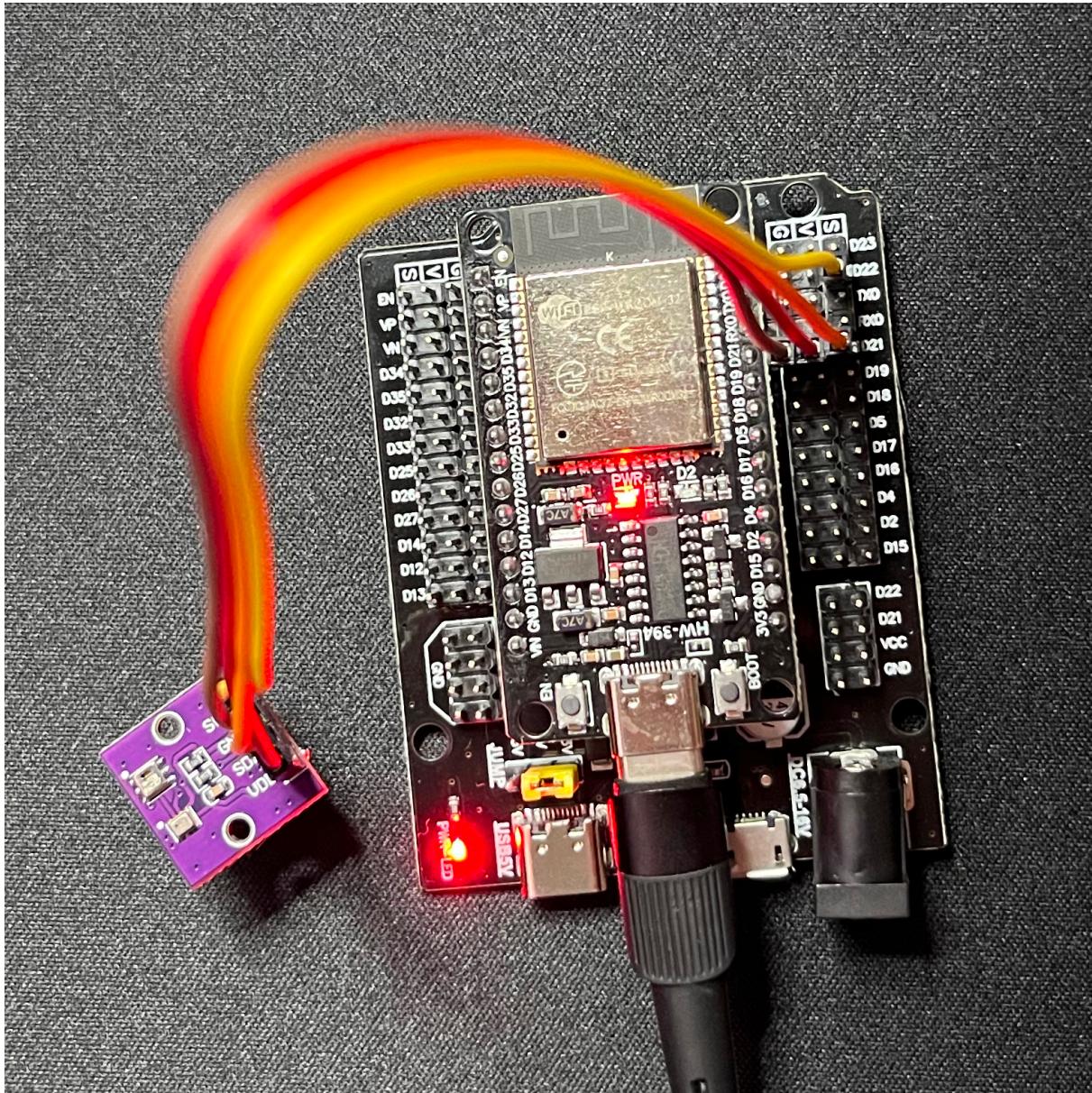


Figura 9: Conexión del sensor de humedad, temperatura y presión atmosférica al ESP32

Ejemplo de lectura de humedad y temperatura:

```
1 #include <Adafruit_AHTX0.h>
```

```

2
3 Adafruit_AHTX0 aht;
4
5 void setup()
6 {
7     Serial.begin(9600);
8     Serial.println("Adafruit AHT10/AHT20 demo!");
9
10    if (! aht.begin())
11    {
12        Serial.println("Could not find AHT? Check wiring");
13        while (1) delay(10);
14    }
15    Serial.println("AHT10 or AHT20 found");
16 }
17
18 void loop() {
19     sensors_event_t humidity, temp;
20     aht.getEvent(&humidity, &temp); // leer datos de humedad y
21     temperatura
22     Serial.print("Temperature: ");
23     Serial.print(temp.temperature);
24     Serial.println(" degrees C");
25     Serial.print("Humidity: ");
26     Serial.print(humidity.relative_humidity);
27     Serial.println("% rH");
28     delay(1000);
29 }
```

Ejemplo de lectura de presión atmosférica, altitud y temperatura (en el IDE puedes encontrarlo en File
-> Examples -> Adafruit BMP280 Library -> bmp280test):

```

1 #include <Wire.h>
2 #include <SPI.h>
3 #include <Adafruit_BMP280.h>
4
5 #define BMP_SCK (13)
6 #define BMP_MISO (12)
7 #define BMP_MOSI (11)
8 #define BMP_CS (10)
9
10 Adafruit_BMP280 bmp; // I2C
11 //Adafruit_BMP280 bmp(BMP_CS); // hardware SPI
12 //Adafruit_BMP280 bmp(BMP_CS, BMP_MOSI, BMP_MISO, BMP_SCK);
13
14 void setup() {
15     Serial.begin(9600);
16     while ( !Serial ) delay(100); // wait for native usb
17     Serial.println(F("BMP280 test"));
18     unsigned status;
```

```
19 //status = bmp.begin(BMP280_ADDRESS_ALT, BMP280_CHIPID);
20 status = bmp.begin();
21 if (!status) {
22     Serial.println(F("Could not find a valid BMP280 sensor, check
23                 wiring or "));
24     Serial.print("SensorID was: 0x"); Serial.println(bmp.sensorID(),16)
25     ;
26     Serial.print("           ID of 0xFF probably means a bad address, a
27                 BMP 180 or BMP 085\n");
28     Serial.print("           ID of 0x56-0x58 represents a BMP 280,\n");
29     Serial.print("           ID of 0x60 represents a BME 280.\n");
30     Serial.print("           ID of 0x61 represents a BME 680.\n");
31     while (1) delay(10);
32 }
33 /* Default settings from datasheet. */
34 bmp.setSampling(Adafruit_BMP280::MODE_NORMAL,          /* Operating Mode.
35                  */
36                  Adafruit_BMP280::SAMPLING_X2,        /* Temp.
37                  oversampling */
38                  Adafruit_BMP280::SAMPLING_X16,       /* Pressure
39                  oversampling */
40                  Adafruit_BMP280::FILTER_X16,         /* Filtering. */
41                  Adafruit_BMP280::STANDBY_MS_500); /* Standby time. */
42
43 void loop() {
44     Serial.print(F("Temperature = "));
45     Serial.print(bmp.readTemperature());
46     Serial.println(" *C");
47
48     Serial.print(F("Pressure = "));
49     Serial.print(bmp.readPressure());
50     Serial.println(" Pa");
51
52     Serial.print(F("Approx altitude = "));
53     /* barómetro ajustado: fuente http://www.ujaen.es/dep/fisica/
54     estacion/estacion3.htm */
55     Serial.print(bmp.readAltitude(1022.5));
56     Serial.println(" m");
57
58     Serial.println();
59     delay(2000);
60 }
```

5.4 Buzzer (zumbador)

Para este ejemplo vamos a usar la línea 15. Conectamos tres cables en la placa de expansión (GVS) en la línea 15. Ponemos el negativo del buzzer en G, el positivo en V y la señal en S. Configuramos como salida la línea 15. Prueba este sketch. ¿Cuál es la melodía oculta?

```
1 const int c = 261;
2 const int d = 294;
3 const int e = 329;
4 const int f = 349;
5 const int g = 391;
6 const int gS = 415;
7 const int a = 440;
8 const int aS = 455;
9 const int b = 466;
10 const int cH = 523;
11 const int cSH = 554;
12 const int dH = 587;
13 const int dSH = 622;
14 const int eH = 659;
15 const int fH = 698;
16 const int fSH = 740;
17 const int gH = 784;
18 const int gSH = 830;
19 const int aH = 880;
20
21 const int buzzerPin = 15;
22 int counter = 0;
23
24 void setup() {
25     //Setup pin modes
26     pinMode(buzzerPin, OUTPUT);
27 }
28
29 void loop() {
30
31     //Play first section
32     firstSection();
33
34     //Play second section
35     secondSection();
36
37     //Variant 1
38     beep(f, 250);
39     beep(gS, 500);
40     beep(f, 350);
41     beep(a, 125);
42     beep(cH, 500);
43     beep(a, 375);
44     beep(cH, 125);
```

```
45     beep(eH, 650);
46
47     delay(500);
48
49     //Repeat second section
50     secondSection();
51
52     //Variant 2
53     beep(f, 250);
54     beep(gS, 500);
55     beep(f, 375);
56     beep(cH, 125);
57     beep(a, 500);
58     beep(f, 375);
59     beep(cH, 125);
60     beep(a, 650);
61
62     delay(650);
63 }
64
65 void beep(int note, int duration) {
66     //Play tone on buzzerPin
67     tone(buzzerPin, note, duration);
68
69     //Play different LED depending on value of 'counter'
70     if (counter % 2 == 0) {
71
72         delay(duration);
73
74     } else {
75
76         delay(duration);
77     }
78
79     //Stop tone on buzzerPin
80     noTone(buzzerPin);
81
82     delay(50);
83
84     //Increment counter
85     counter++;
86 }
87
88 void firstSection() {
89     beep(a, 500);
90     beep(a, 500);
91     beep(a, 500);
92     beep(f, 350);
93     beep(cH, 150);
94     beep(a, 500);
95     beep(f, 350);
```

```
96     beep(cH, 150);
97     beep(a, 650);
98
99     delay(500);
100
101    beep(eH, 500);
102    beep(eH, 500);
103    beep(eH, 500);
104    beep(fH, 350);
105    beep(cH, 150);
106    beep(gS, 500);
107    beep(f, 350);
108    beep(cH, 150);
109    beep(a, 650);
110
111    delay(500);
112 }
113 void secondSection() {
114     beep(aH, 500);
115     beep(a, 300);
116     beep(a, 150);
117     beep(aH, 500);
118     beep(gSH, 325);
119     beep(gH, 175);
120     beep(fSH, 125);
121     beep(fH, 125);
122     beep(fSH, 250);
123
124     delay(325);
125
126     beep(aS, 250);
127     beep(dSH, 500);
128     beep(dH, 325);
129     beep(cSH, 175);
130     beep(cH, 125);
131     beep(b, 125);
132     beep(ch, 250);
133
134     delay(350);
135 }
```

5.5 PIR (Sensor Infrarrojo Pasivo)

Conectamos el PIN del sensor GROUND al pin G de la placa de expansión en la señal 15. Hacemos lo mismo con VCC y la línea V de la placa de expansión y finalmente la señal (línea S de la placa de expansión) va al pin de salida del sensor como se ve en las figuras:

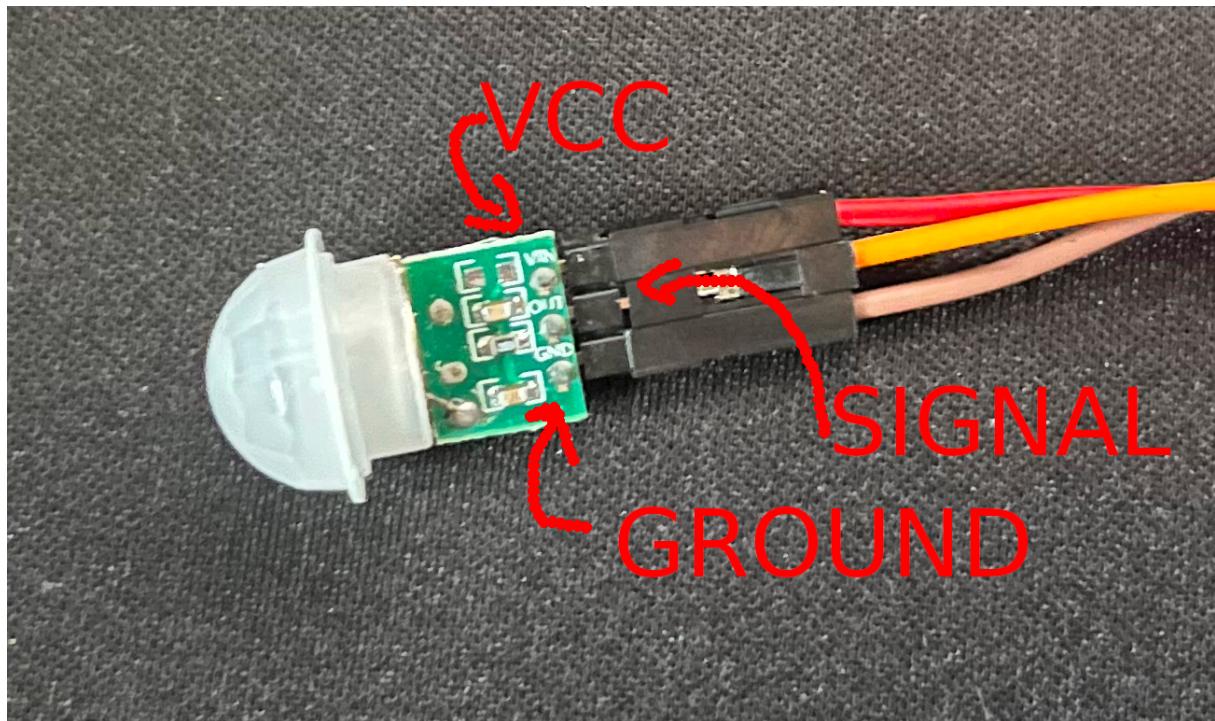


Figura 10: Cableado del sensor

En nuestro caso si ponemos la cabeza del sensor hacia arriba y las patillas para atrás, de izquierda a derecha tenemos:

- **GND:** Ground o masa, el polo negativo.
- **V. IN:** V.IN o VCC, el polo positivo.
- **OUT:** La salida de la señal.

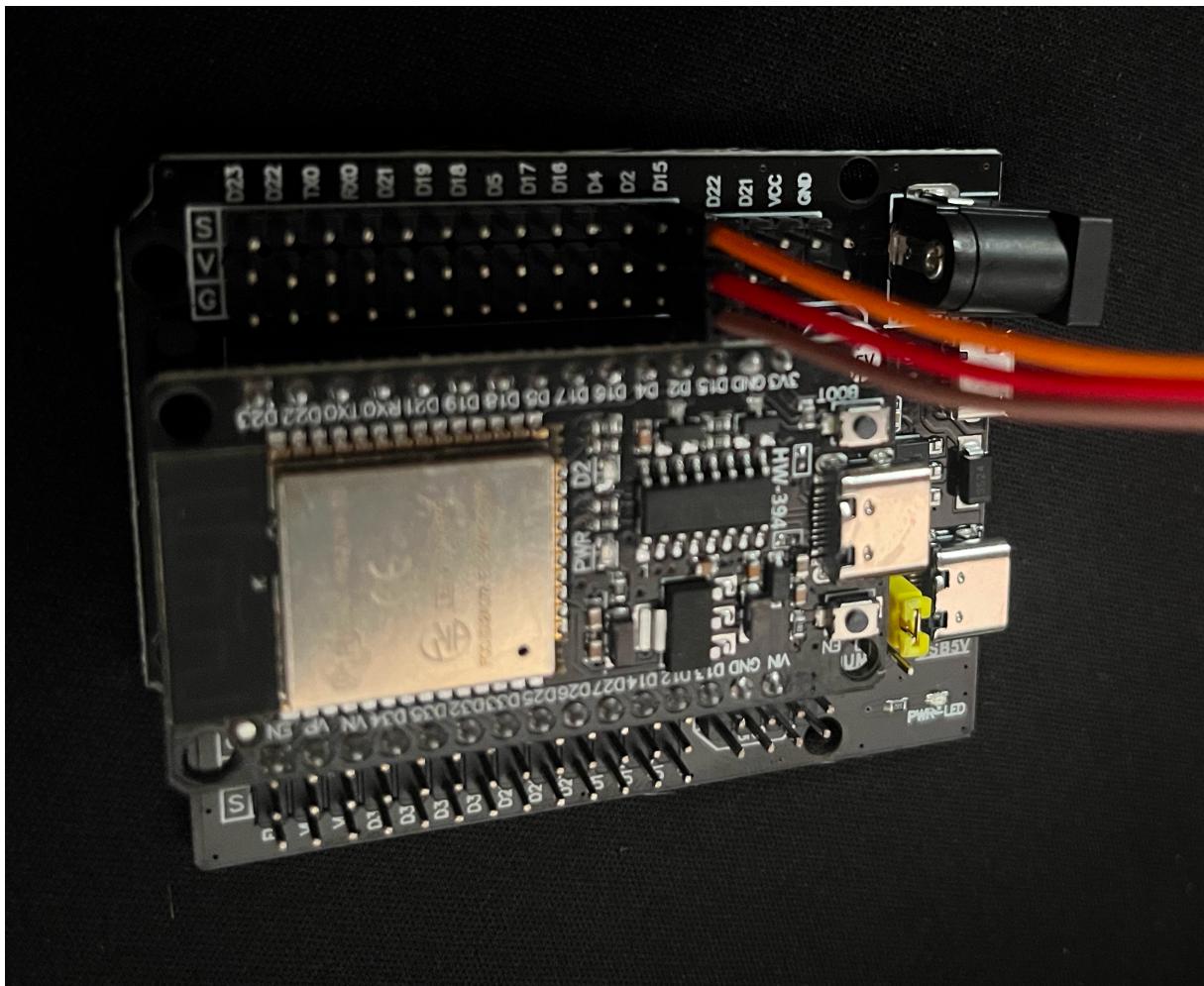


Figura 11: Cableado de la placa de expansión

```
1  /*
2   * PIR sensor
3   */
4
5  int inputPin = 15;           // choose the input pin (for PIR
6  sensor)
7  int pirState = LOW;         // we start, assuming no motion
detected
8  int val = 0;                // variable for reading the pin status
9
10 void setup() {
11   pinMode(inputPin, INPUT); // declare sensor as input
12   Serial.begin(9600);
13 }
14 void loop(){
```

```

15  val = digitalRead(inputPin); // read input value
16  if (val == HIGH) {           // check if the input is HIGH
17
18      if (pirState == LOW) {
19          // we have just turned on
20          Serial.println("Motion detected!");
21          // We only want to print on the output change, not state
22          pirState = HIGH;
23      }
24  } else {
25
26      if (pirState == HIGH){
27          // we have just turned off
28          Serial.println("Motion ended!");
29          // We only want to print on the output change, not state
30          pirState = LOW;
31      }
32  }
33 }
```

5.6 Conexión a WiFi

Como ya sabemos, para conectar dos dispositivos de red vía WiFi, disponemos de tres opciones:

1. **modo estación (conexión a un punto de acceso)**: Conectamos a un router o punto de acceso que a su vez nos comunica con otros equipos conectados a la LAN o a Internet.
2. **modo punto de acceso (creación de un punto de acceso)**: El ESP32 es quien nos *da* conexión, es decir el resto de dispositivos se conectan a él. Como es obvio, en este caso no tendremos conexión a Internet.
3. **modo ad-hoc (conexión directa entre dispositivos)**: Esta opción es la más restrictiva, pues permite la comunicación punto a punto entre dispositivos configurados exactamente igual solamente (normalmente sólo dos dispositivos).

Veamos cómo sería en cada caso:

5.6.1 1. Configurar ESP32 como Estación (Conexión a un Punto de Acceso):

```

1 #include <WiFi.h>
2
3 const char *ssid = "NombreDelPuntoDeAcceso";
4 const char *password = "ContraseñaDelPuntoDeAcceso";
5
6 void setup() {
7     Serial.begin(115200);
```

```

8
9 // Conectar el ESP32 a un punto de acceso
10 WiFi.begin(ssid, password);
11
12 while (WiFi.status() != WL_CONNECTED) {
13     delay(1000);
14     Serial.println("Conectando a WiFi...");
15 }
16
17 Serial.println("Conectado a WiFi");
18 }
19
20 void loop() {
21     // Tu código aquí
22 }
```

Este código configura el ESP32 para conectarse a un punto de acceso WiFi específico utilizando el nombre (SSID) y la contraseña proporcionados. Si el punto de acceso está conectado a Internet, todos los dispositivos tienen la posibilidad de acceder a la red de redes.

5.6.2 2. Configurar ESP32 como Punto de Acceso:

```

1 #include <WiFi.h>
2
3 const char *ssid = "NombreDelPuntoDeAcceso";
4 const char *password = "ContraseñaDelPuntoDeAcceso";
5
6 void setup() {
7     Serial.begin(115200);
8
9     // Configurar el ESP32 como punto de acceso
10    WiFi.softAP(ssid, password);
11
12    Serial.println("Punto de Acceso configurado con éxito");
13 }
14
15 void loop() {
16     // Tu código aquí
17 }
```

Este código configura el ESP32 para crear un punto de acceso WiFi con el nombre (SSID) y la contraseña proporcionados. Los dispositivos pueden conectarse a este punto de acceso utilizando las credenciales especificadas. No hay posibilidad de usar Internet.

5.6.3 3. Configurar ESP32 en Modo Ad-Hoc:

```
1 #include <WiFi.h>
2
3 const char *ssid = "NombreDeRedAdHoc";
4 const char *password = "ContraseñaAdHoc";
5
6 void setup() {
7     Serial.begin(115200);
8
9     // Configurar el ESP32 en modo ad-hoc
10    WiFi.begin(ssid, password, 1, NULL, false);
11
12    while (WiFi.status() != WL_CONNECTED) {
13        delay(1000);
14        Serial.println("Conectando al modo Ad-Hoc...");
15    }
16
17    Serial.println("Conectado al modo Ad-Hoc");
18}
19
20 void loop() {
21     // Tu código aquí
22 }
```

Este código configura el ESP32 para unirse a una red WiFi ad-hoc con el nombre y la contraseña especificados. Puedes configurar varios ESP32 para unirse a la misma red ad-hoc y comunicarse directamente entre sí.

El modo ad-hoc es más complejo y puede tener limitaciones dependiendo de los dispositivos y las configuraciones específicas. Todos los dispositivos que deseas que se comuniquen estén en el mismo canal y compartan la misma configuración de red (nombre y contraseña).

6 Introducción a NodeJS

6.1 Instalación de NodeJS

Desde que **node** es a su vez un paquete instalable con el gestor de paquetes de **npm**, lo podemos instalar globalmente así:

en Debian/Ubuntu/Mint:

```
1 sudo npm install -g n
2 sudo n 20
```

en Windows 10/11 (con elevación):

```
1 PS> winget install openjs.nodejs.lts
```

6.2 Modo interactivo

6.3 Backend de ejemplo

Uno de los puntos fuertes de Node.JS es su capacidad para actuar como servidor o backend de aplicaciones, pues en apenas unas pocas líneas podremos implementar un servicio. A continuación vamos a ver como hacer un servidor de ejemplo en poco más de 50 líneas de código.

1. Configuración del servidor (Node.js con Express):

En el directorio raíz de nuestro proyecto, nos vamos a la carpeta **node**, que ya tenía la dependencia del cliente InfluxDB (mira el fichero package.json) y ejecuta los siguientes comandos en la terminal:

```
1 cd node
2 npm install express
```

Luego, creamos un archivo llamado **server.js**:

```
1 const express = require('express');
2 const app = express();
3
4 // enrutador
5 app.get('/', (req, res) => {
6   res.send('Has hecho una petición GET.');
7 });
8
9 app.post('/', (req, res) => {
10   res.send('Has hecho una petición POST.');
11});
```

```

12
13 app.delete('/', (req, res) => {
14   res.send('Has hecho una petición DELETE.');
15 });
16
17 app.put('/', (req, res) => {
18   res.send('Has hecho una petición PUT.');
19 });
20
21 // los primeros 1024 puertos se pueden usar sólo por el sistema
22 // operativo (elevación)
22 app.listen(8000, () => console.log('Example app is listening on port
23   8000.'));

```

Para lanzar el servidor, desde la carpeta *node* escribimos:

```
1 node server.js
```

Ahora podemos con Postman o la extensión de Firefox REST client hacer las peticiones para probar que funciona. También se puede hacer con CURL:

```

1 curl -X GET    -i http://localhost:8000
2 curl -X POST   -i http://localhost:8000
3 curl -X PUT    -i http://localhost:8000
4 curl -X DELETE -i http://localhost:8000

```

Ten cuidado si pruebas con cURL de usar en tus POST/PUT/PATCH el parámetro `-H 'Content-Type : application/json'` y `-H 'Accept: application/json'` para añadir en las cabeceras de la petición que estamos mandando o esperamos los datos en formato JSON. Ejemplo:

```

1 curl -X GET -H 'Accept: application/json' -i http://localhost:8000
2 curl -X POST -H 'Content-Type: application/json' -i http://localhost
:8000/todos --data '{"id": 10, "task":"Estudiar para el examen de
NodeJS"}'
3 curl -X PUT -H 'Content-Type: application/json' -i http://localhost
:8000/todos/10 --data '{"id": 10, "task":"Estudiar para el examen de
Acceso a Datos"}'
4 curl -X DELETE -i http://localhost:8000/10

```

Vamos a crear una aplicación (API REST) que consume una lista de tareas:

```

1 const express = require('express');
2 const bodyParser = require('body-parser');
3 const app = express();
4 const PORT = 8000;
5
6 app.use(bodyParser.json());
7
8 let todos = [
9   { id: 1, task: 'Nos vamos de excursión a Oracle' },

```

```

10     { id: 2, task: 'Nos volvemos muy aburridos de Oracle' },
11   ];
12
13 // GET - Obtener todas las tareas
14 app.get('/todos', (req, res) => {
15   res.json(todos);
16 });
17
18 // POST - Agregar una nueva tarea
19 app.post('/todos', (req, res) => {
20   const newTodo = req.body;
21   todos.push(newTodo);
22   res.status(201).json(newTodo);
23 });
24
25 // PUT - Actualizar una tarea existente
26 app.put('/todos/:id', (req, res) => {
27   const todoId = parseInt(req.params.id);
28   const updatedTodo = req.body;
29   todos = todos.map(todo => (todo.id === todoId ? updatedTodo : todo));
30   res.json(updatedTodo);
31 });
32
33 // DELETE - Eliminar una tarea
34 app.delete('/todos/:id', (req, res) => {
35   const todoId = parseInt(req.params.id);
36   todos = todos.filter(todo => todo.id !== todoId);
37   res.sendStatus(204);
38 });
39
40 app.listen(PORT, () => {
41   console.log(`El servidor está corriendo en http://localhost:${PORT}`)
42 });

```

Fíjate cómo hemos leído el parámetro **id** en el *DELETE* de la tarea:

```

1 app.delete('/todos/:id', (req, res) => {
2   const todoId = parseInt(req.params.id);
3   todos = todos.filter(todo => todo.id !== todoId);
4   res.sendStatus(204);
5 });

```

6.3.1 Paso de parámetros:

Al tratarse de una API REST el parámetro va directamente en la ruta sin más, así si queremos hacer un borrado de la tarea 2 hacemos una petición *DELETE* a la URL: <http://localhost:8000/todos/2>. Sin embargo, estamos acostumbrados a ver en la URL del navegador algo como: <http://localhost:8000/deleteTodo?id=2>,

en este caso no sería una API REST y entonces los parámetros van asignados con el símbolo '='. Veamos otro ejemplo con dos parámetros para entenderlo:

| Tipo | Verbo URL | Explicación |
|--------|--|-------------|
| REST | PATCH http://localhost:8000/todos/2/task/no%20hacer%20nada | |
| normal | POST | |

6.4 Frontend

Vamos a dar un paso más, vamos a crear ahora una Web, para ello añadimos una dependencia más:

```
1 npm install --save body-parser
```

1. Cliente HTML5 y JavaScript:

Preparamos el servidor para que podamos servir contenido estático. Para ello creamos una carpeta **public** donde va a estar el contenido estático y añadimos esta línea al enrutador (antes del primer **app.get**):

```
1 // permitimos servir contenido estático (carpeta public)
2 app.use('/', express.static('public'));
```

Creamos un archivo llamado **index.html**:

```
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Programador de tareas</title>
7 </head>
8 <body>
9
10 <h1>TO-DO App</h1>
11
12 <!-- Formulario para agregar una nueva tarea -->
13 <form id="addTodoForm">
14   <label for="task">Nueva tarea:</label>
15   <input type="text" id="task" required>
16   <button type="submit">Agregar tarea</button>
17 </form>
18
19 <!-- Lista de tareas -->
```

```
20 <ul id="todoList"></ul>
21
22 <script>
23 // Cliente JavaScript para interactuar con la API REST
24
25 const apiUrl = 'http://localhost:8000/todos';
26 const todoList = document.getElementById('todoList');
27 const addTodoForm = document.getElementById('addTodoForm');
28
29 // Función para cargar las tareas
30 async function fetchTodos() {
31     const response = await fetch(apiUrl);
32     const todos = await response.json();
33     displayTodos(todos);
34 }
35
36 // Función para mostrar las tareas en el cliente
37 function displayTodos(todos) {
38     todoList.innerHTML = '';
39     todos.forEach(todo => {
40         const listItem = document.createElement('li');
41         listItem.innerHTML = `${todo.task}
42             <button onclick="updateTodo(${todo.id})">Actualizar</button>
43             <button onclick="deleteTodo(${todo.id})">Eliminar</button>`;
44         todoList.appendChild(listItem);
45     });
46 }
47
48 // Función para agregar una nueva tarea
49 async function addTodo(event) {
50     event.preventDefault();
51     const taskInput = document.getElementById('task');
52     const task = taskInput.value;
53
54     const response = await fetch(apiUrl, {
55         method: 'POST',
56         headers: {
57             'Content-Type': 'application/json',
58         },
59         body: JSON.stringify({ task }),
60     });
61
62     if (response.ok) {
63         fetchTodos();
64         taskInput.value = '';
65     }
66 }
67
68 // Función para actualizar una tarea
69 async function updateTodo(id) {
70     const response = await fetch(`${apiUrl}/${id}`);
```

```
71     const todo = await response.json();
72     const updatedTask = prompt('Actualizar tarea:', todo.task);
73
74     if (updatedTask !== null) {
75         await fetch(`/${apiUrl}/${id}`, {
76             method: 'PUT',
77             headers: {
78                 'Content-Type': 'application/json',
79             },
80             body: JSON.stringify({ task: updatedTask }),
81         });
82
83         fetchTodos();
84     }
85 }
86
87 // Función para eliminar una tarea
88 async function deleteTodo(id) {
89     const confirmDelete = confirm('¿Seguro que quieres eliminar esta
90     tarea?');
91
92     if (confirmDelete) {
93         await fetch(`/${apiUrl}/${id}`, { method: 'DELETE' });
94         fetchTodos();
95     }
96
97 // Event Listener para el formulario de agregar tarea
98 addTodoForm.addEventListener('submit', addTodo);
99
100 // Cargar las tareas al cargar la página
101 fetchTodos();
102 </script>
103
104 </body>
105 </html>
```

7 Protegiendo el servicio

A continuación vamos a ver cómo crear un sistema de inicio de sesión con Node.js, Express, Pug y Mysql.

Para crear un sistema de inicio de sesión con Node.js, Express, Pug y almacenamiento de datos de usuario y contraseñas en MySQL, sigue estos pasos:

7.1 1. Configuración del Proyecto

1. Crea un nuevo directorio para tu proyecto.
2. Inicializa un nuevo proyecto de Node.js ejecutando `npm init -y` en la terminal.
3. Instala las dependencias necesarias:

```
1 npm install express pug mysql2 express-session body-parser
```

7.2 2. Configuración de MySQL

Asegúrate de tener un servidor MySQL en ejecución y crea una base de datos junto con una tabla para almacenar los datos del usuario. Aquí hay un ejemplo simple:

```
1 CREATE DATABASE IF NOT EXISTS mydatabase;
2 USE mydatabase;
3
4 CREATE TABLE IF NOT EXISTS users (
5     id INT AUTO_INCREMENT PRIMARY KEY,
6     username VARCHAR(255) NOT NULL,
7     password VARCHAR(255) NOT NULL
8 );
```

7.3 3. Configuración del Servidor Express

Crea un archivo llamado `app.js` para tu servidor Express:

```
1 const express = require('express');
2 const session = require('express-session');
3 const mysql = require('mysql2');
4 const bodyParser = require('body-parser');
5 const path = require('path');
6
7 const app = express();
8 const port = 8000;
```

```
9
10 // Configuración de MySQL
11 const connection = mysql.createConnection({
12   host: 'localhost',
13   port: 33306,
14   user: 'root',
15   password: '238fmvmM',
16   database: 'reservas',
17 });
18
19 // Conexión a MySQL
20 connection.connect(err => {
21   if (err) {
22     console.error('Error al conectar a MySQL:', err);
23     return;
24   }
25   console.log('Conectado a MySQL');
26 });
27
28 // Configuración de sesiones
29 app.use(
30   session({
31     secret: 'your-secret-key', // se puede generar con bcrypt
32     resave: false,
33     saveUninitialized: true,
34   })
35 );
36
37 // Configuración de Pug
38 app.set('view engine', 'pug');
39 app.set('views', path.join(__dirname, 'views'));
40
41 // Middleware para analizar el cuerpo de las solicitudes
42 app.use(bodyParser.urlencoded({ extended: true }));
43
44 // Rutas
45 app.get('/', (req, res) => {
46   res.render('index', { user: req.session.user });
47 });
48
49 app.get('/login', (req, res) => {
50   res.render('login');
51 });
52
53 app.post('/login', (req, res) => {
54   const { username, password } = req.body;
55
56   // Verificación de credenciales en MySQL
57   const query = 'SELECT * FROM users WHERE username = ? AND password = ?';
58   connection.query(query, [username, password], (err, results) => {
```

```

59     if (err) {
60         console.error('Error al verificar las credenciales:', err);
61         res.redirect('/login');
62     } else {
63         if (results.length > 0) {
64             req.session.user = username;
65             res.redirect('/');
66         } else {
67             res.redirect('/login');
68         }
69     }
70 });
71 });
72
73 app.get('/logout', (req, res) => {
74     req.session.destroy(err => {
75         if (err) {
76             console.error(err);
77         } else {
78             res.redirect('/login');
79         }
80     });
81 });
82
83 // Iniciar el servidor
84 app.listen(port, () => {
85     console.log(`Server is running on http://localhost:${port}`);
86 });

```

7.4 4. Crear las Vistas Pug

Crea un directorio llamado `views` en el directorio raíz de tu proyecto. Dentro de este directorio, crea dos archivos Pug: `index.pug` y `login.pug`.

7.4.1 Contenido de `views/index.pug`

Este archivo contiene la vista principal:

```

1 doctype html
2 html(lang="en")
3     head
4         meta(charset="UTF-8")
5         meta(name="viewport", content="width=device-width, initial-scale
6             =1.0")
7         title Sistema de Login
8         body
9             h1 Bienvenido, #{user || 'Invitado'}

```

```
9  if user
10    p
11      a(href='/logout') Cerrar Sesión
12  else
13    p
14      a(href='/login') Iniciar Sesión
```

7.4.2 Contenido de views/login.pug

Formulario de login:

```
1 doctype html
2 html(lang="en")
3   head
4     meta(charset="UTF-8")
5     meta(name="viewport", content="width=device-width, initial-scale
6       =1.0")
7     title Iniciar Sesión
8   body
9     h1 Iniciar Sesión
10    form(action='/login', method='post')
11      label(for='username') Usuario:
12        input(type='text', id='username', name='username', required)
13        br
14      label(for='password') Contraseña:
15        input(type='password', id='password', name='password', required)
16        br
17        button(type='submit') Iniciar Sesión
```

7.5 Ejecución del Proyecto

1. Asegúrate de tener un servidor MySQL en ejecución.
2. Ejecuta `node app.js` para iniciar el servidor Express.
3. Accede a tu aplicación en `http://localhost:8000`.

Este ejemplo básico utiliza una conexión a MySQL para verificar las credenciales durante el inicio de sesión. Asegúrate de modificar las configuraciones de MySQL (`tu_usuario_mysql`, `tu_contraseña_mysql`) según tu entorno. Además, en un entorno de producción, deberías manejar las contraseñas de manera más segura y considerar la seguridad en todas las capas de tu aplicación.

8 Ejemplo: Almacenamiento de datos de temperatura y humedad en InfluxDB

Para enviar datos desde tu dispositivo Arduino a InfluxDB a través de un broker Mosquitto, necesitarás agregar la lógica MQTT al ejemplo que ya vimos anteriormente.

Primero, debemos asegurarnos de tener la biblioteca PubSubClient instalada en tu entorno de desarrollo Arduino. Puedes instalarlo desde el Gestor de bibliotecas de Arduino.

A continuación, modificamos el ejemplo que ya vimos de captura de datos de humedad, temperatura y presión y los mandamos también al bróker:

```
1 #include <Wire.h>
2 #include <SPI.h>
3 #include <Adafruit_BMP280.h>
4 #include <WiFi.h>
5 #include <PubSubClient.h>
6
7 #define BMP_SCK  (13)
8 #define BMP_MISO (12)
9 #define BMP_MOSI (11)
10 #define BMP_CS   (10)
11
12 const char *ssid = "TuSSID";           // Cambia por tu SSID
13 const char *password = "TuClave";      // Cambia por tu contraseña
14 const char *mqtt_server = "IP_de_Tu_Broker"; // Cambia por la dirección
                                                IP de tu broker MQTT
15
16 WiFiClient espClient;
17 PubSubClient client(espClient);
18
19 Adafruit_BMP280 bmp; // I2C
20
21 void setup() {
22     Serial.begin(9600);
23     while (!Serial)
24         delay(100); // wait for native usb
25     Serial.println(F("BMP280 test"));
26     unsigned status;
27     status = bmp.begin();
28     if (!status) {
29         Serial.println(F("Could not find a valid BMP280 sensor, check
29             wiring or "));
30             "try a different address!"));
31     while (1)
32         delay(10);
33 }
34
35 /* Default settings from datasheet. */
```

```
36 bmp.setSampling(Adafruit_BMP280::MODE_NORMAL,          /* Operating Mode. */
37             Adafruit_BMP280::SAMPLING_X2,           /* Temp. oversampling
38             */
39             Adafruit_BMP280::SAMPLING_X16,          /* Pressure
40             oversampling */
41             Adafruit_BMP280::FILTER_X16,           /* Filtering. */
42             Adafruit_BMP280::STANDBY_MS_500);    /* Standby time. */
43
44 // Conectar a la red WiFi
45 WiFi.begin(ssid, password);
46 while (WiFi.status() != WL_CONNECTED) {
47     delay(1000);
48     Serial.println("Connecting to WiFi...");
49 }
50
51 // Conectar al broker MQTT
52 client.setServer(mqtt_server, 1883);
53 while (!client.connected()) {
54     Serial.println("Connecting to MQTT...");
55     if (client.connect("ESP32Client")) {
56         Serial.println("Connected to MQTT");
57     } else {
58         Serial.print("Failed with state ");
59         Serial.println(client.state());
60         delay(2000);
61     }
62 }
63
64 void loop() {
65     Serial.print(F("Temperature = "));
66     Serial.print(bmp.readTemperature());
67     Serial.println(" *C");
68     Serial.print(F("Pressure = "));
69     Serial.print(bmp.readPressure());
70     Serial.println(" Pa");
71     Serial.print(F("Approx altitude = "));
72     Serial.print(bmp.readAltitude(1022.5));
73     Serial.println(" m");
74
75 // Enviar datos a InfluxDB a través de MQTT
76 char temperature[10];
77 char pressure[10];
78 char altitude[10];
79
80 sprintf(temperature, "%.*f", bmp.readTemperature());
81 sprintf(pressure, "%.*f", bmp.readPressure());
82 sprintf(altitude, "%.*f", bmp.readAltitude(1022.5));
```

```
85 client.publish("influxdb/temperature", temperature);
86 client.publish("influxdb/pressure", pressure);
87 client.publish("influxdb/altitude", altitude);
88
89 delay(2000);
90 }
```

Fíjate como usamos al final la biblioteca PubSubClient para establecer una conexión MQTT y publicar datos en los temas “influxdb/temperature”, “influxdb/pressure” y “influxdb/altitude”. Asegúrate de cambiar las variables `ssid`, `password` y `mqtt_server` con los valores correspondientes a tu red WiFi y tu broker MQTT. Además, debes tener un servidor InfluxDB configurado para recibir y almacenar estos datos. Ajusta los temas MQTT según tus preferencias.

Si ya creaste los contenedores Docker, ya dispones de un broker MQTT funcionando. Para tomar datos de Mosquitto (broker MQTT) y pasarlos a InfluxDB utilizando Node-RED, puedes seguir estos pasos:

8.0.1 1. Instalar Paquetes Necesarios:

En Node-RED, instala los paquetes necesarios para MQTT y InfluxDB. Puedes hacerlo a través de la interfaz de usuario de Node-RED (puedes acceder a ella en tu navegador en <http://localhost:1880>).

- Haz clic en el botón en la esquina superior derecha para abrir el menú.
- Selecciona “Manage palette”.
- Ve a la pestaña “Install”.
- Busca e instala los paquetes `node-red-contrib-mqtt-broker` para MQTT y `node-red-contrib-influxdb` para InfluxDB.

8.0.2 2. Configurar el Nodo MQTT:

- Agrega un nodo MQTT input (mqtt in) desde la paleta de nodos a tu flujo.
- Configura el nodo con los detalles del servidor MQTT, como el broker, el puerto y el tema al que se subscribirá para recibir datos.

8.0.3 3. Configurar el Nodo InfluxDB:

- Agrega un nodo InfluxDB output (influxdb out) a tu flujo.
- Configura el nodo con los detalles de tu servidor InfluxDB, como la dirección, el puerto, la base de datos y las credenciales si es necesario.

8.0.4 4. Conectar los Nodos:

- Conecta el nodo MQTT input al nodo InfluxDB output arrastrando un cable entre ellos en el flujo.
- Asegúrate de configurar correctamente los campos de datos para que coincidan con los datos que estás enviando a través de MQTT.

8.0.5 5. Desplegar el Flujo:

- Haz clic en el botón “Deploy” para aplicar los cambios en tu flujo.

Ahora, Node-RED debería estar suscrito a los datos enviados por tu dispositivo a través de MQTT y los insertará en la base de datos InfluxDB según la configuración que hayas establecido.

Este es un flujo básico, y puedes ajustarlo según tus necesidades específicas. Node-RED es muy flexible y te permite realizar transformaciones, filtrados y otras operaciones en los datos antes de almacenarlos en InfluxDB.

9 Apéndice

9.1 Un poco de redes

Como estamos trabajando con varios contenedores y diferentes direcciones IP, una interesante idea es dar nombre a nuestros equipos (aunque sea una referencia a localhost) para simular un entorno aún más real:

En nuestros sistemas operativos podemos añadir hosts desde los ficheros:

| Sistema operativo | Fichero |
|-------------------|---------------------------------------|
| Linux/UNIX | /etc/hosts |
| Windows | C:/Windows/system32/drivers/etc/hosts |

Recuerda que para trabajar con equipos de nuestra red LAN (o loopbak local) tenemos las siguientes direcciones:

| IP inicial | máscara | IP final | tipo | clase |
|-----------------|---------------|-------------------|----------------|-------|
| 127.0.0.1/8 | 255.0.0.0 | 127.255.255.254/8 | loopback | A |
| 10.0.0.1/8 | 255.0.0.0 | 10.255.255.254/8 | Direc. privada | A |
| 172.16.0.0/16 | 255.255.0.0 | 172.31.255.254/16 | Direc. privada | B |
| 192.168.xx.1/24 | 255.255.255.0 | 192.168.xx.254/24 | Direc. privada | C |

Es importante recordar que debemos proteger las redes de los dispositivos IoT todo lo posible, y la recomendación es añadir un recibimiento SSL a las comunicaciones usando certificados para evitar fugas, manipulación o robo de datos.

9.2 Bibliografía

- Ejemplo de MING Stack para acelerar el desarrollo de aplicaciones: <https://github.com/JoePortilla/MING-stack>.