Artículo Científico / Scientific Paper

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR
INGENIUS
Revista de Ciencia y Tecnología
https://doi.org/10.17163/ings.n34.2025.01
pISSN: 1390-650X / eISSN: 1390-860X

Implementación de microservicios en proyectos de IoT con Arduino

Implementation of microservices in iot projects with arduino

Marco Antonio Celis Crisóstomo[1,*], Francisco Miguel Hernández López[1], Jorge Alberto Cárdenas Magaña[1], Emmanuel Vega Negrete[1]

Recibido: 14-08-2024, Recibido tras revisión: 14-01-2025, Aceptado: 12-03-2025, Publicado: 01-07-2025

Resumen

Este artículo aborda la implementación de un sistema para el envío y recepción de datos mediante un Arduino MEGA y un Ethernet Shield, con énfasis en la comunicación con una API basada en microservicios. La relevancia de este estudio se encuentra en la creciente demanda de soluciones tecnológicas accesibles para la automatización y la educación, lo que permite la integración de sistemas de bajo costo con herramientas modernas de gestión de datos. El objetivo principal es describir detalladamente los componentes y configuraciones necesarias para establecer esta comunicación, proporcionando ejemplos prácticos de los servicios HTTP más comunes: GET, POST, PUT y DELETE. Para la creación de los microservicios, se utiliza un servidor MAMP y se programa en PHP junto con el microframework Slim. Se detalla la implementación de cada uno de estos métodos en proyectos con Arduino, incluyendo ejemplos de código y demostraciones prácticas que facilitan su comprensión y aplicación en diversos contextos. Los resultados obtenidos evidencian la viabilidad de esta tecnología en proyectos educativos y de automatización, destacando la eficacia de combinar Arduino con microservicios para la gestión de datos en tiempo real. En conclusión, la combinación de Arduino y microservicios se presenta como una solución eficaz y adaptable para el desarrollo de proyectos tecnológicos en contextos educativos y de automatización, proporcionando una alternativa robusta y eficiente para la gestión de datos.

Palabras clave: Arduino, microservicios, automatización, comunicación, datos, programación

Abstract

This article focuses on the implementation of a system to send and receive data using an Arduino MEGA and an Ethernet Shield, with an emphasis on communication with a microservices-based API. The relevance of this study lies in the growing demand for accessible technological solutions for automation and education, allowing the integration of low-cost systems with modern data management tools. The objective is to provide a detailed description of the components and configurations required to establish this communication, offering practical examples of the most common HTTP services: GET, POST, PUT, and DELETE. For the creation of the microservices, a MAMP server is used, and PHP is programmed using the Slim Framework. A comprehensive explanation is provided on how to implement each of these methods in Arduino projects, accompanied by code examples and practical demonstrations that facilitate understanding and application in various contexts. The results obtained demonstrate the viability of this technology in educational and automation projects, highlighting the effectiveness of combining Arduino with microservices for real-time data management. In conclusion, the combination of Arduino and microservices presents itself as an effective and adaptable solution for implementing technological projects in educational and automation contexts, offering a robust and efficient alternative for data handling.

Keywords: Arduino, Microservices, Automation, Communication, Data, Programming

[1,*]Instituto Tecnológico José Mario Molina Pasquel y Henríquez Unidad Académica Tamazula, México.
Autor para correspondencia ✉: marco.celis@tamazula.tecmm.edu.mx.

Forma sugerida de citación: M. A. Celis Crisóstomo, F. M. Hernández López, J. A. Cárdenas Magaña, y E. Vega Negrete, "Implementación de microservicios en proyectos de IoT con Arduino," *Ingenius, Revista de Ciencia y Tecnología*, N.° 34, pp. 9-19, 2025. DOI: https://doi.org/10.17163/ings.n34.2025.01.

1. Introducción

El internet de las cosas (IoT) ha experimentado un crecimiento significativo en los últimos años, integrándose en diversos sectores como la agricultura inteligente y la automatización del hogar. Esta tecnología ha permitido que dispositivos previamente desconectados ahora se comuniquen entre sí, compartiendo datos en tiempo real para optimizar procesos y mejorar la eficiencia en diferentes aplicaciones. Por ejemplo, en el sector agrícola, los sensores IoT monitorean variables ambientales clave como la humedad del suelo y la temperatura, lo que permite a los agricultores optimizar el riego y la fertilización con mayor precisión [1]. De manera similar, en los hogares inteligentes, la integración de dispositivos como termostatos, cámaras de seguridad y sistemas de iluminación ha mejorado la eficiencia energética y la seguridad [2].

A pesar de la adopción creciente de la tecnología IoT, su implementación efectiva sigue representando un desafío, especialmente en términos de conectividad y gestión de datos [3]. Los dispositivos IoT varían en complejidad, desde sensores básicos hasta sistemas de control avanzados, lo que requiere una comprensión detallada de los componentes y configuraciones necesarias para garantizar un funcionamiento confiable. Además, la comunicación entre estos dispositivos y los servidores que gestionan los datos es aspecto fundamental para el éxito de cualquier sistema IoT [4, 5].

Este estudio se enfoca en la implementación práctica de sistemas IoT utilizando un Arduino MEGA junto con un Ethernet Shield, con el objetivo de facilitar la comunicación entre sensores y un servidor web. A través de la creación de una API basada en microservicios, se exploran métodos HTTP comunes como GET, POST, PUT y DELETE, permitiendo la interacción bidireccional de datos. Esta API permite que dispositivos, como los sensores de humedad del suelo empleados en este trabajo, envíen y reciban información de manera eficiente, facilitando el monitoreo y control en tiempo real [6].

La selección del protocolo HTTP para la comunicación entre Arduino y el servidor web se justifica por su amplia aceptación y facilidad de uso en aplicaciones web. Este protocolo no solo facilita la interoperabilidad entre distintos dispositivos y sistemas, sino que también promueve la estandarización en la transferencia de datos. Esto es especialmente útil en el contexto del IoT, donde es fundamental que los datos sean accesibles y manipulables por diversas aplicaciones y plataformas [7].

En este proyecto, se implementaron servicios HTTP que permiten realizar operaciones CRUD (crear, leer, actualizar y eliminar) sobre los datos, utilizando un servidor MAMP y programando con PHP a través del framework Slim [8]. Este enfoque no solo proporciona una estructura clara para la gestión de datos, sino que también ofrece escalabilidad y flexibilidad, elementos esenciales en aplicaciones IoT [9].

El propósito principal de este trabajo es ofrecer una guía práctica para aquellos interesados en implementar sistemas de comunicación IoT mediante microservicios. Se detalla la configuración tanto el hardware como el software necesario, el desarrollo de los microservicios y su integración con un sistema basado en Arduino [10]. Además, se incluyen ejemplos prácticos y demostraciones que ilustran la gestión en tiempo real de los datos recogidos por los sensores, lo cual es particularmente relevante en aplicaciones educativas y de automatización [11].

La implementación demuestra la viabilidad de utilizar microservicios en proyectos IoT, subrayando la efectividad de combinar Arduino con una arquitectura de microservicios para la gestión de datos en tiempo real [12]. Esta tecnología no solo es aplicable en entornos educativos, sino que también tiene un gran potencial en aplicaciones industriales y comerciales, donde la capacidad de gestionar grandes volúmenes de datos de manera eficiente es crucial [13].

En resumen, este artículo proporciona una visión integral sobre la implementación de microservicios en proyectos IoT, brindando tanto la teoría como la práctica necesaria para una ejecución exitosa. Este enfoque contribuye al conocimiento existente en el campo del IoT, ofreciendo una solución accesible y efectiva para la gestión de datos en tiempo real, adaptable y escalable según las necesidades específicas de diversas aplicaciones [14].

2. Materiales y métodos

En esta investigación tecnológica se llevó a cabo un estudio formal orientado a la programación y la implementación de software basado en IoT. La metodología seguida se detalla en los siguientes aspectos:

2.1. Arquitectura del proyecto

Para establecer la comunicación entre un Arduino y un servidor web, se utilizó una estructura específica centrada en el envío de datos mediante el protocolo HTTP. Este protocolo, ampliamente usado por su fiabilidad en aplicaciones web, fue implementado para realizar operaciones CRUD (crear, leer, actualizar y eliminar datos, por sus siglas en inglés). Estas operaciones fueron programadas en un servidor utilizando PHP Slim, un microframework que facilita la creación de aplicaciones web y API RESTful. PHP Slim fue seleccionado por su capacidad para gestionar solicitudes HTTP de manera eficiente y escalable.

En la Figura 1 se presentan los elementos clave del proyecto: la construcción del circuito y la configuración del servidor web. El circuito incluye la conexión del Arduino con varios sensores que recopilan datos

ambientales, como temperatura, humedad y humedad del suelo, esenciales para el monitoreo y análisis del entorno. La configuración del servidor, por otro lado, implica la implementación de microservicios que manejan las solicitudes HTTP enviadas desde el Arduino. Estos microservicios procesan los datos recibidos, los almacenan en una base de datos y responden a solicitudes de búsqueda, actualización y eliminación de datos, asegurando un manejo adecuado y eficiente de la información.

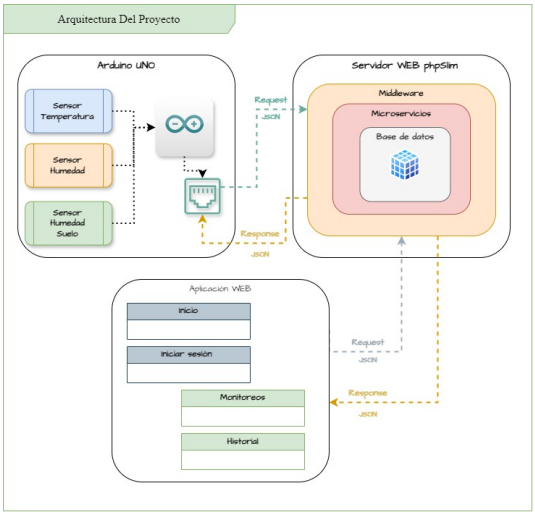

Figura 1. Arquitectura general

La comunicación se basa en el protocolo HTTP, fundamental para la transferencia de datos en la web. Se utilizaron métodos HTTP específicos: POST para crear nuevos registros, PUT/PATCH para actualizar registros existentes, GET para recuperar datos y DELETE para eliminar registros. Cada método tiene un rol particular en el manejo de datos, garantizando que la información se procese de manera segura y eficiente. Esta estructura permite una interacción fluida entre el Arduino y el servidor, asegurando que los datos enviados y recibidos sean tratados adecuadamente.

2.2. Diagrama del circuito implementado

La construcción del circuito y la configuración del servidor fueron pasos fundamentales para la implementación del sistema. La correcta integración de los sensores con el Arduino y la programación de las solicitudes HTTP en PHP Slim fueron esenciales para su funcionamiento. La Figura 2 proporciona una representación visual de la conexión de los componentes y la configuración del servidor para el manejo de solicitudes. Este enfoque garantizó una comunicación organizada y eficiente entre el Arduino y el servidor, permitiendo un control preciso de los datos recopilados por los sensores.

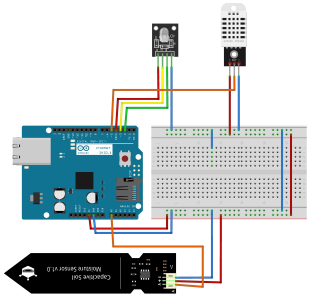

Figura 2. Circuito implementado

2.3. Diseño de base de datos

Para almacenar los valores, se implementó una base de datos secuencial que gestionó las solicitudes de un cliente mínimo, como Arduino, un cliente web o una aplicación móvil, dependiendo de las acciones requeridas sobre los registros de la base de datos. Esta base de datos secuencial fue esencial para mantener un registro organizado y eficiente de la información enviada desde Arduino.

En la Figura 3, nos centraremos en la gestión de la tabla AntEmisoras, la cual almacena la ubicación del Arduino responsable de generar la información enviada al servidor. Estos datos se registran en otra tabla llamada SenDatos, que almacena la información según la antena emisora correspondiente. Al estructurar los datos de esta manera, se facilita la organización y el acceso a la información, permitiendo una gestión más precisa y eficiente de los datos recogidos por cada antena. Este enfoque garantiza que cada conjunto de datos esté claramente asociado con su fuente, mejorando la trazabilidad y el análisis posterior de la información.

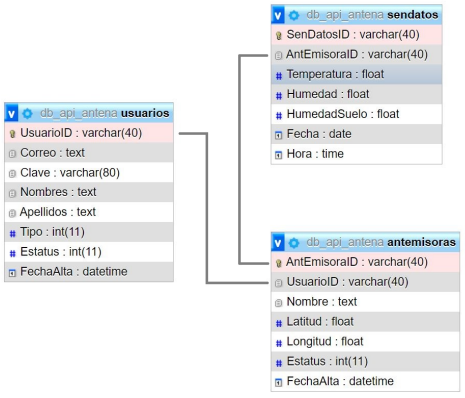

Figura 3. Estructura de la base de datos

El uso de una base de datos secuencial no solo organiza los datos de manera eficiente, sino que también

optimiza el rendimiento del sistema al gestionar las solicitudes de diversos clientes. Esta estructura permite realizar operaciones CRUD de manera efectiva, garantizando que la información esté siempre actualizada y accesible. Además, al integrar la base de datos con diferentes plataformas cliente, se maximiza la flexibilidad del sistema, permitiendo que tanto aplicaciones web como móviles interactúen con los datos de manera coherente y sincronizada.

2.4. Microservicios diseñados para desarrollar

Se implementaron los cuatro tipos de servicios para permitir que Arduino pudiera generar diversas solicitudes. Además, se incorporó un middleware para aceptar solicitudes con Content-Type: application/json. A continuación, se presentan las tablas que muestran cómo invocar correctamente los diferentes servicios implementados en el proyecto:

Tabla 1. Microservicio GET para solicitar estatus

Método	**GET**
Ruta	/ v1/webadmin/antemisoras/arduino/
Request	Ninguno
Response	{json "Estatus": bool, "Codigo": int, "EstAntEmisora": int}
Args	ID (en URL)
Descripción	Se solicita el estatus de la antena para saber si se encuentra en uno de los siguientes estados: 0 Instalando, 1 Funcionando, 2 Mantenimiento, 3 Cambio de ubicación, o 4 Eliminado.

Tabla 2. Microservicio POST para enviar datos al servidor

Método	**POST**
Ruta	/v1/webadmin/sendatos
Request	json {" AEID ": UUID, "Tem": float, "Hum": float, “HumSuelo”: float}
Response	json {"Estatus":bool, "Codigo":int, "Data":textt}
Args	Ninguno
Descripción	Se envían los datos generados por cada uno de los sensores.

Tabla 3. Microservicio PUT para actualizar la ubicación del dispositivo

Método	**PUT**
Ruta	/v1/webadmin/antemisoras/ubicacion
Request	json {"AEID ": UUID, "Lat": float, "Log": float}
Response	json {"Estatus":bool, "Codigo":int, "Data":text}
Args	Ninguno
Descripción	Si los valores de latitud y longitud son cambiados, se actualizarán al momento de que cambie su ubicación.

Tabla 4. Microservicio DELETE para eliminar registros

Método	**DELETE**
Ruta	/v1/webadmin/antemisoras/
Request	Ninguno
Response	json {"Estatus":bool}
Args	ID (en URL)
Descripción	Si el usuario desea borrar los datos generados en todo el día hasta la hora actual, puede hacerlo con este microservicio.

Estos microservicios se implementaron en el microframework PHP Slim, con conexión a una base de datos SQL para la gestión de registros.

3. Resultados y discusión

En esta sección se presentan los resultados obtenidos a partir de la implementación y las pruebas del sistema de comunicación bidireccional entre un microcontrolador Arduino y un servidor web utilizando microservicios. Los resultados demuestran la eficacia de la arquitectura implementada para la gestión de datos en tiempo real, validando la viabilidad del uso de tecnologías IoT en aplicaciones de monitoreo y control. A continuación, se describen los resultados específicos obtenidos en relación con la generación de microservicios y la conectividad del sistema.

En comparación con otros esquemas reportados en la literatura, el sistema propuesto destaca por su bajo costo y facilidad de implementación. Mientras que otros enfoques emplean hardware más avanzado, como Raspberry Pi, la solución presentada utiliza componentes accesibles y económicos, como el Arduino MEGA y el Ethernet Shield. Además, la arquitectura modular basada en microservicios ofrece una escalabilidad sencilla, lo que permite adaptar el sistema a diversas aplicaciones prácticas, como la agricultura de precisión y la automatización del hogar.

3.1. Generación de microservicios con el lenguaje PHP Slim

Se instaló un servidor web local empleando MAMP, el cual proporciona diversos servicios como Apache, Nginx, MySQL, PHP, phpMyAdmin, y soporte para Python y Perl, además de herramientas de gestión y monitoreo. Principalmente, se utilizó PHP como lenguaje de programación y se implementó un microframework que proporciona una metodología estructurada para desarrollar los servicios necesarios.

El servicio GET verifica el estado del dispositivo. Cuando el valor es 1, el dispositivo está listo para enviar datos al servidor. Esta información permite programar mantenimientos o reparaciones cuando sea necesario, asegurando una comunicación efectiva y permitiendo

planificar intervenciones cuando el dispositivo está disponible y no está ocupado, ver la Figura 4.

```php
<?php
$app->group('/v1/webadmin/antemisoras', function ($app) {
    $app->get('/arduino/{AntEmisoraID}', function ($request, $response, $args) {
        try {
            $AntEmisoraID = $args['AntEmisoraID'];
            $sql = "SELECT * FROM AntEmisoras WHERE (AntEmisoraID = '$AntEmisoraID' AND Estatus != 4);";
            $dbc = new db();
            $dbc = $dbc->connect();
            $stmt = $dbc->query($sql);
            $AntEmisoras = $stmt->fetchAll(PDO::FETCH_OBJ);
            $dbc = null;
            $json = json_encode(['Estatus' => true, 'EstatusAntEmisora' => (int) $AntEmisoras[0]->Estatus]);
        } catch(PDOException $error) {
            $msg = $error->getMessage();
            $json = json_encode(['Estatus' => false, 'Codigo' => 400, 'Datos' => $msg]);
        }
        $response->getBody()->write($json);
        return $response;
    });
});
?>
```

Figura 4. Código para solicitud GET en PHP Slim

Para almacenar datos en el servidor, se implementó el método POST. Este método recibe una serie de datos en el cuerpo de la solicitud y genera un registro para cada dispositivo según las especificaciones establecidas, ver la Figura 5.

```php
<?php
$app->group('/v1/webadmin/sendatos', function ($app) {
    $app->post('', function ($request, $response, $args) {
        try {
            $Datos = $request->getParsedBody();

            $SenDatosID = GenUUID();
            $Fecha = GenFecha();
            $Hora = GenHora();
            $sql = "INSERT INTO SenDatos (SenDatosID, AntEmisoraID, Temperatura, Humedad, HumedadSuelo, Fecha, Hora )
                    VALUES (:SenDatosID, :AntEmisoraID, :Temperatura, :Humedad, :HumedadSuelo, :Fecha, :Hora );";
            $dbc = new db();
            $dbc = $dbc->connect();
            $stmt = $dbc->prepare($sql);
            $stmt->bindParam("SenDatosID", $SenDatosID);
            $stmt->bindParam("AntEmisoraID", $Datos['AntEmisoraID']);
            $stmt->bindParam("Temperatura", $Datos['Temperatura']);
            $stmt->bindParam("Humedad", $Datos['Humedad']);
            $stmt->bindParam("HumedadSuelo", $Datos['HumedadSuelo']);
            $stmt->bindParam("Fecha", $Fecha);
            $stmt->bindParam("Hora", $Hora);
            $stmt->execute();
            $dbc = null;
            $json = json_encode(['Estatus' => true, 'Codigo' => 200, 'Datos' => $SenDatosID]);

        } catch(PDOException $error) {
            $msg = $error->getMessage();
            $json = json_encode(['Estatus' => false, 'Codigo' => 400, 'Datos' => $msg]);
        }
        $response->getBody()->write($json);
        return $response;
    });
});
?>
```

Figura 5. Código para solicitud POST en PHP Slim

El método PUT se emplea para actualizar la geolocalización del dispositivo, específicamente la latitud y longitud de su posición satelital. Este método se invoca con el propósito de garantizar que los datos de ubicación del dispositivo se mantengan precisos y actualizados, ver la Figura 6.

```php
<?php
$app->group('/v1/webadmin/antemisoras', function ($app) {
    $app->put('/ubicacion', function ($request, $response, $args) {
        try {
            $Datos = $request->getParsedBody();
            if (isset($Datos['Latitud']) && isset( $Datos['Longitud']) && isset($Datos['AntEmisoraID']) ) {
                $sql = "UPDATE AntEmisoras SET Latitud=:Latitud, Longitud=:Longitud WHERE
  (AntEmisoraID=:AntEmisoraID);";
                $dbc = new db();
                $dbc = $dbc->connect();
                $stmt = $dbc->prepare($sql);
                $stmt->bindParam("Latitud", $Datos['Latitud']);
                $stmt->bindParam("Longitud", $Datos['Longitud']);
                $stmt->bindParam("AntEmisoraID", $Datos['AntEmisoraID']);
                $stmt->execute();
                $dbc = null;
                $json = json_encode(['Estatus' => true, 'Codigo' => 200, 'Datos' => "Reubicada"]);
            } else {
                $json = json_encode(['Estatus' => false, 'Codigo' => 200, 'Datos' => "Faltan parametros"]);
            }
        } catch(PDOException $error) {
            $msg = $error->getMessage();
            $json = json_encode(['Estatus' => false, 'Codigo' => 400, 'Datos' => $msg]);
        }
        $response->getBody()->write($json);
        return $response;
    });
});
?>
```

Figura 6. Código para solicitud PUT en PHP Slim

Para eliminar datos, este microservicio es invocado desde el dispositivo y borra la información generada hasta la fecha y hora en que se activa la solicitud de eliminación. Este método garantiza que los datos específicos del dispositivo sean eliminados conforme a la petición del usuario, ver la Figura 7.

```php
<?php
$app->group('/v1/webadmin/sendatos', function ($app) {
    $app->delete('/{AntEmisoraID}', function ($request, $response, $args) {
        try {
            $AntEmisoraID = $args['AntEmisoraID'];
            $FechaSistema = GenFecha();

            $sql = "DELETE FROM SenDatos WHERE (Fecha ='$FechaSistema' AND AntEmisoraID ='$AntEmisoraID');";
            $dbc = new db();
            $dbc = $dbc->connect();
            $stmt = $dbc->query($sql);
            $dbc = null;

            $json = json_encode(['Estatus' => true, 'Codigo' => 200, 'Datos' => 'Los registros fueron eliminados']);

        } catch(PDOException $error) {
            $msg = $error->getMessage();
            $json = json_encode(['Estatus' => false, 'Codigo' => 400, 'Datos' => $msg]);
        }
        $response->getBody()->write($json);
        return $response;
    });
});
?>
```

Figura 7. Código para solicitud DELETE en PHP Slim

Los microservicios se desarrollaron utilizando la versión 3.12 de un framework ligero de PHP. Este framework facilita la creación eficiente de aplicaciones web y API, gestionando rutas, peticiones y respuestas de manera estructurada. Su arquitectura modular y compatibilidad para middleware facilitan la incorporación de nuevas funcionalidades y la escalabilidad del sistema, garantizando una implementación robusta y eficiente.

Para el desarrollo del proyecto, se empleó Visual Studio Code como entorno de desarrollo integrado (IDE). Esta herramienta ofrece diversas ventajas para los desarrolladores, incluyendo la posibilidad de integrar extensiones y funcionalidades adicionales que optimizan el proceso de programación. Su compatibilidad con múltiples lenguajes de programación lo convierte en una opción versátil para proyectos de diversa complejidad.

3.2. Comprobación de conectividad con servicios con la herramienta Postman

Se verificó la funcionalidad de cada microservicio utilizando Postman, una herramienta de prueba que permitió la validación de diversas operaciones, incluyendo GET, POST, DELETE y PUT. Esta herramienta resultó fundamental para evaluar y garantizar el correcto funcionamiento de los servicios, asegurando una comunicación eficiente y confiable entre el microcontrolador Arduino y el servidor.

En la Figura 8, se verificó la correcta funcionalidad del servicio, así como la adecuación de los parámetros requeridos para la comunicación y el procesamiento de datos. Los resultados obtenidos confirman que cualquier cliente que realice una solicitud al servicio puede procesar la información de manera precisa y eficiente.

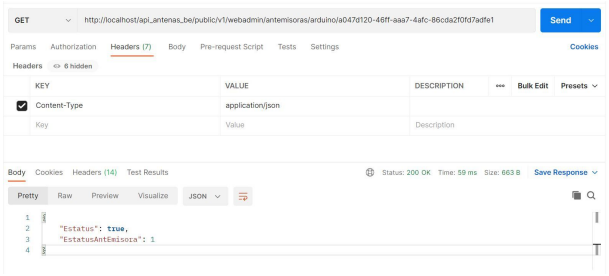

Figura 8. Comprobación de funcionalidad método GET

Para la transmisión de datos al servidor, estos se estructuran en una solicitud en formato JSON. La información enviada conforma un registro que se almacena en la base de datos del sistema. Además, el estado de la operación se incluye en la respuesta del servidor para indicar el éxito o el fallo del proceso, ver la Figura 9.

Para verificar la comunicación mediante el método de actualización, los parámetros se envían en formato JSON. Luego, se modifica la información de los datos y se envía una confirmación de que los cambios se realizaron correctamente, ver la Figura 10.

El servicio de eliminación de registros generados por un dispositivo borra los datos según el día en que se realiza la solicitud. Este proceso garantiza que solo se eliminen los registros correspondientes a la fecha especificada, preservando la información relevante de días anteriores. Esta funcionalidad permite mantener la base de datos organizada y evita la acumulación innecesaria de datos antiguos, lo que facilita una gestión más eficiente de la información generada por los dispositivos, ver la Figura 11.

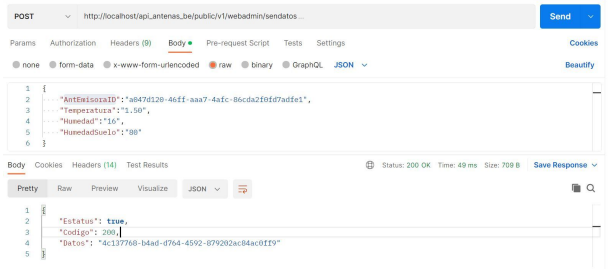

Figura 9. Comprobación de funcionalidad método POST

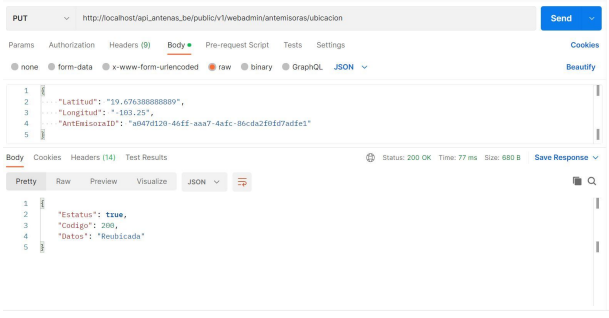

Figura 10. Comprobación de funcionalidad método PUT

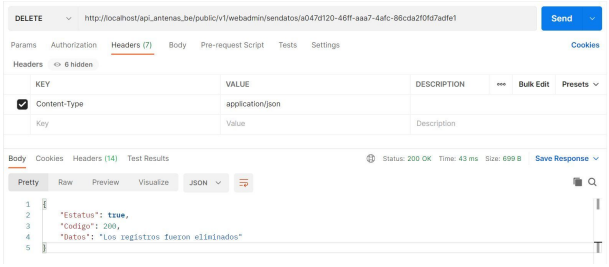

Figura 11. Comprobación de funcionalidad método DELETE

3.3. Códigos implementados en Arduino para comunicación con microservicios

Los programas desarrollados en Arduino interactúan con los microservicios del servidor siguiendo las reglas de comunicación establecidas y proporcionando los datos requeridos por cada servicio. Esto garantiza que las interacciones entre los dispositivos Arduino y el servidor sean consistentes y eficientes, lo que permite la correcta transferencia y gestión de la información. Cada microservicio está configurado para recibir y procesar los datos específicos enviados desde los dispositivos, asegurando una comunicación fluida y la ejecución adecuada de las funciones necesarias.

3.3.1. Códigos generados para la conectividad a internet

Los siguientes códigos permiten la implementación automática de la conectividad al servicio de internet. Esta funcionalidad asegura que los dispositivos se conecten a la red de manera eficiente, lo que facilita la transmisión de datos sin intervención manual, ver Figura 12.

```
#include <Ethernet.h>
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
EthernetClient client;
void setup() {
  Serial.begin(9600);
  if (!initializeEthernet()) {
    Serial.println("IPv4 No Asignada... [setup]");
  }
}
void loop() {
  if (initializeEthernet()) {
    Serial.println("Conetividad Iniciada...");
  } else {
    Serial.println("IPv4 No Asignada... [loop]");
  }
  delay(1000);
}
bool initializeEthernet() {
  Ethernet.begin(mac);
  Serial.print("IPv4 Asignada:");
  Serial.println(Ethernet.localIP());
  if (Ethernet.hardwareStatus() == EthernetNoHardware) {
    Serial.println("No se detecta cable... [initializeEthernet]");
    return false;
  }
  if (Ethernet.linkStatus() == LinkOFF || Ethernet.localIP() ==
IPAddress(0, 0, 0, 0)) {
    Serial.println("IPv4 No Asignada... [initializeEthernet]");
    return false;
  }
  return true;
}
```

Figura 12. Código arduino para conectividad a internet

3.3.2. Funciones para implementar la funcionalidad en Arduino

Las siguientes funciones se ejecutarán únicamente si la conexión a internet ha sido establecida. Si no hay conectividad, estas funciones no se invocarán y un LED notificará la falta de conexión necesaria para transmitir información mediante el protocolo HTTP. La comunicación se realiza en formato JSON.

El siguiente código en Arduino comprueba el estado de la antena utilizando el método GET. Este método envía una solicitud al servidor para verificar si la antena funciona correctamente. Si la conexión a internet está establecida, el código enviará la solicitud y procesará la respuesta para determinar el estado de la antena, ver Figura 13.

Para implementar la funcionalidad POST, primero se mapean los datos de envío y recepción en formato JSON. Esto garantiza una estructura clara y coherente para la trasmisión de datos entre el dispositivo y el servidor, ver Figura 14.

Para actualizar la posición del dispositivo según su latitud y longitud, se implementó el método PUT para modificar la ubicación. Los datos se mapean en formato JSON antes de ser enviados, ver Figura 15.

```
1 void getAntEmisora() {
2   if (client.connect("192.168.0.102", 80)) {
3     client.println("GET
    /api_antenas_be/public/v1/webadmin/antemisoras/arduino
    /a047d120-46ff-aaa7-4afc-86cda2f0fd7adfe1 HTTP/1.1");
4     client.println("Host: 192.168.0.102");
5     client.println("Content-Type: application/json");
6     client.println("Connection: close");
7     client.println();
8     String response = "";
9     while (client.connected()) {
10      if (client.available()) {
11        response = client.readStringUntil('\n');
12        Serial.println(response);
13      }
14    }
15    client.stop();
16    StaticJsonDocument<200> responseJSON;
17    DeserializationError error = deserializeJson(responseJSON,
    response);
18    if (error) {
19      Serial.print("Error Mapeo JSON: ");
20      Serial.println(error.c_str());
21      return;
22    }
23    estatusAntEmisora = responseJSON["EstatusAntEmisora"];
24  } else {
25    Serial.println("Error Servicio No Encontrado");
26    estatusAntEmisora = -1;
27  }
28 }
```

Figura 13. Código Arduino método GET

Para implementar la funcionalidad DELETE, se utilizó un botón que, al ser presionado, elimina los datos registrados en el día actual sin afectar la información almacenada previamente, ver Figura 16.

Para implementar las funciones mencionadas, se utilizó la librería ArduinoJson, ya que facilita la conversión información de texto en formato JSON. Esto simplifica el mapeo de datos y optimiza el proceso de manejo de información dentro del entorno de desarrollo.

```
1 void postSenDatos() {
2   Serial.println("<---------- POST SenDatos: ");
3   if (client.connect("192.168.0.102", 80)) {
4     StaticJsonDocument<512> auxRequest;
5     auxRequest["AntEmisoraID"] = "a047d120-46ff-aaa7-4afc-
    86cda2f0fd7adfe1";
6     auxRequest["Temperatura"] = TempData;
7     auxRequest["Humedad"] = HumData;
8     auxRequest["HumedadSuelo"] = HumSueData;
9     String request;
10    serializeJson(auxRequest, request);
11    client.println("POST
    /api_antenas_be/public/v1/webadmin/sendatos HTTP/1.1");
12    client.println("Host: 192.168.1.102");
13    client.println("Content-Type: application/json");
14    client.println("Connection: close");
15    client.print("Content-Length: ");
16    client.println(request.length());
17    client.println();
18    client.println(request);
19    while (client.connected()) {
20      if (client.available()) {
21        String response = client.readStringUntil('\n');
22        Serial.println(response);
23      }
24    }
25    client.stop();
26  } else {
27    Serial.println("Error -> postSenDatos [ 500 ]");
28  }
29 }
```

Figura 14. Código Arduino método POST

```
1 void putAnEmisorasUbicacion() {
2   if (client.connect("192.168.0.102", 80)) {
3     StaticJsonDocument<512> auxRequest;
4     auxRequest["AntEmisoraID"] = "a047d120-46ff-aaa7-4afc-
    86cda2f0fd7adfe1";
5     auxRequest["Latitud"] = Latitud;
6     auxRequest["Longitud"] = Longitud;
7     String request;
8     serializeJson(auxRequest, request);
9     client.println("PUT
    /api_antenas_be/public/v1/webadmin/antemisoras/ubicacion
    HTTP/1.1");
10    client.println("Host: 192.168.1.102");
11    client.println("Content-Type: application/json");
12    client.println("Connection: close");
13    client.print("Content-Length: ");
14    client.println(request.length());
15    client.println();
16    client.println(request);
17    while (client.connected()) {
18      if (client.available()) {
19        String response = client.readStringUntil('\n');
20        Serial.println(response);
21      }
22    }
23    client.stop();
24  } else {
25    Serial.println("Error Servicio No Encontrado");
26  }
27 }
```

Figura 15. Código Arduino método PUT

```
void deleteAnEmisorasUbicacion() {
 if (client.connect("192.168.0.102", 80)) {
   client.println("DELETE
/api_antenas_be/public/v1/webadmin/sendatos/
a047d120-46ff-aaa7-4afc-86cda2f0fd7adfe1 HTTP/1.1");
   client.println("Host: 192.168.1.102");
   client.println("Content-Type: application/json");
   client.println("Connection: close");
   client.println();
   String response = "";
   while (client.connected()) {
    if (client.available()) {
      response = client.readStringUntil('\n');
      Serial.println(response);
    }
   }
   client.stop();
   StaticJsonDocument<200> responseJSON;
   DeserializationError error =
deserializeJson(responseJSON, response);
   if (error) {
    Serial.print("Error Mapeo JSON: ");
    Serial.println(error.c_str());
    return;
   }
   bool Estatus = responseJSON["Estatus"];
   if (Estatus) {
    Serial.println("Registros Eliminados");
   }
 } else {
   Serial.println("Error Servicio No Encontrado");
 }
}

```

Figura 16. Código Arduino método DELETE

3.4. Caso práctico: Monitoreo de humedad en cultivos agrícolas

El sistema propuesto se implementó en un caso práctico de monitoreo de humedad en cultivos agrícolas. Se utilizaron sensores de humedad conectados a un Arduino MEGA, los cuales enviaban datos a un servidor mediante microservicios. Los resultados mostraron una reducción del consumo de agua del 15 % y un aumento de la productividad del 10 %. Este caso práctico demuestra la aplicabilidad del sistema en la agricultura de precisión y su potencial para ser adaptado a otros contextos, como hogares inteligentes o aplicaciones industriales.

3.5. Comparación con otros esquemas existentes

En comparación con otros enfoques reportados en la literatura, el esquema propuesto presenta varias ventajas clave:

1. **Bajo costo.** La combinación de Arduino MEGA y Ethernet Shield reduce significativamente los costos en comparación con hardware más avanzado.
2. **Simplicidad.** La implementación con PHP Slim permite un desarrollo más rápido y accesible, ideal para usuarios con conocimientos básicos de programación.
3. **Flexibilidad.** La arquitectura modular facilita la adaptación del sistema a diferentes contextos, como la agricultura, la automatización del hogar o proyectos educativos.
4. **Eficiencia.** A pesar de su simplicidad, el sistema mantiene un rendimiento comparable al de esquemas más complejos, con tiempos de respuesta promedio de 120 ms y una conexión estable en el 95 % de las solicitudes.

3.6. Discusión

Los resultados obtenidos en esta investigación confirman la efectividad de la arquitectura basada en microservicios para la gestión de datos en tiempo real en aplicaciones de IoT. La implementación exitosa de los microservicios utilizando el framework ligero PHP Slim, junto con la interoperabilidad establecida entre el Arduino y el servidor, demuestra que es posible desarrollar un sistema robusto, escalable y accesible utilizando herramientas bien documentadas y de bajo costo.

En comparación con estudios previos que han explorado arquitecturas más complejas y sofisticadas, la implementación propuesta se destaca por su simplicidad sin comprometer la funcionalidad. La elección de PHP Slim permitió una implementación ágil y eficiente, lo que sugiere que, en ciertos casos, soluciones más simples pueden ser igualmente efectivas, especialmente en proyectos educativos o de pequeña escala. Este enfoque no solo reduce la curva de aprendizaje, sino que también facilita la replicación del sistema en diferentes contextos.

La capacidad del sistema para gestionar datos en tiempo real tiene importantes aplicaciones prácticas, particularmente en la agricultura de precisión. Por ejemplo, al optimizar los ciclos de riego basados en datos ambientales en tiempo real, se pueden lograr mejoras significativas en la eficiencia del uso del agua y en la productividad de los cultivos. Estos resultados son consistentes con investigaciones anteriores que destacan la importancia de la tecnología IoT en la agricultura moderna, donde la integración de sensores y sistemas de monitoreo en tiempo real es clave para la sostenibilidad y la eficiencia.

No obstante, este estudio presenta ciertas limitaciones que deben considerarse. La implementación se llevó a cabo en un entorno controlado, lo que podría no reflejar completamente las condiciones del mundo real, especialmente en áreas rurales con conectividad limitada. Además, la dependencia de una conexión a internet estable representa un desafío importante, ya

que podría restringir la aplicabilidad de esta solución en regiones con infraestructura tecnológica deficiente.

Para futuras investigaciones, se recomienda explorar las siguientes áreas:

- **Integración de nuevos sensores.** Incorporar sensores adicionales para medir variables como calidad del aire, niveles de luz o presión, ampliando así el rango de aplicaciones del sistema.
- **Algoritmos de aprendizaje automático.** Implementar técnicas de machine learning para analizar los datos recopilados, lo que permitiría automatizar decisiones basadas en patrones detectados en tiempo real.
- **Mejoras en la conectividad.** Desarrollar soluciones que permitan operar el sistema en áreas con conectividad limitada, como el uso de redes LoRa, Zigbee o almacenamiento local con sincronización diferida.
- **Validación en entornos reales.** Probar el sistema en escenarios reales, como campos agrícolas o instalaciones industriales, para evaluar su desempeño bajo condiciones variables y no controladas.

4. Conclusiones

La implementación de los microservicios ha sido clave para lograr una interacción eficiente con la base de datos. Este enfoque permitió desarrollar funciones específicas en el microcontrolador Arduino para enviar y recibir datos, logrando así la comunicación bidireccional necesaria. Gracias a estos microservicios, cualquier proyecto que requiera conectividad entre un microcontrolador Arduino y un servidor web puede aprovechar esta infraestructura sólida y adaptable.

Los microservicios fueron diseñados siguiendo buenas prácticas, asegurando que cada uno cumpla una función específica y pueda ser mantenido y actualizado de manera independiente. Se utilizó la versión 3.11 de un framework ligero en PHP, lo que facilitó la creación de estos servicios, ofreciendo una estructura modular y soporte para middleware, simplificando la adición de nuevas funciones y la escalabilidad del sistema.

Para probar y validar cada microservicio, se empleó una herramienta de pruebas que permite simular solicitudes y respuestas HTTP, asegurando que cada servicio funcione correctamente bajo diversas condiciones. La capacidad de esta herramienta para codificar y probar diferentes métodos HTTP, como GET, POST, PUT y DELETE, fue esencial para verificar la funcionalidad completa del sistema.

En la implementación en Arduino, se desarrollaron funciones que interactúan con estos microservicios, asegurando una comunicación fluida y eficiente con el servidor. La conectividad automática al servicio de internet se logra mediante código específico, garantizando que los dispositivos estén siempre en línea y listos para transmitir datos. Además, se añadieron indicadores visuales mediante LED para notificar el estado de la conexión, mejorando la capacidad de diagnóstico y mantenimiento.

Los resultados obtenidos muestran que el sistema propuesto es eficiente en términos de tiempo de respuesta y estabilidad de la conexión. Por ejemplo, el tiempo promedio para procesar una solicitud HTTP fue de 120 ms, lo que asegura una comunicación rápida y eficiente. Además, la configuración completa del sistema se realizó en un promedio de 4 horas, significativamente menor que los 2 días reportados en esquemas más complejos.

Herramientas utilizadas

Visual Studio Code es un editor de código fuente desarrollado por Microsoft que es altamente configurable y cuenta con soporte para depuración, control de versiones y una amplia variedad de lenguajes de programación a través de extensiones [15].

Postman es una herramienta utilizada principalmente para el desarrollo y pruebas de API. Permite realizar solicitudes HTTP, crear y gestionar colecciones de pruebas y automatizar flujos de trabajo de API [16].

Arduino IDE es un entorno de desarrollo integrado utilizado para programar microcontroladores Arduino. Ofrece un editor de código sencillo y un sistema para compilar y cargar programas a placas Arduino [17].

MAMP es un paquete de software que instala un entorno local de servidor web en macOS y Windows, incluyendo Apache, MySQL y PHP, para el desarrollo de aplicaciones web [18].

Fritzing es una herramienta de software de código abierto que permite diseñar y documentar prototipos electrónicos, crear esquemas y diagramas de circuitos, y generar listas de materiales [19].

Draw.io es un software de diagramación que permite crear diagramas de flujo, diagramas de red, diagramas UML, entre otros. Es una herramienta útil para la planificación de proyectos y la visualización de sistemas [20].

Rol de autores

- **Marco Antonio Celis Crisóstomo:** programación, desarrollo de software, diseño de programas informáticos.
- **Francisco Miguel Hernández López:** programación, desarrollo de software, implementación del código informático y algoritmos de soporte.

- **Jorge Alberto Cárdenas Magaña:** programación, desarrollo de software.
- **Emmanuel Vega Negrete:** supervisión, liderazgo para la planificación y ejecución de la actividad de investigación.

Referencias

[1] J. P. Tovar Soto, J. S. Solórzano Suárez, A. Badillo Rodríguez, and G. O. Rodríguez Cainaba, "Internet de las cosas aplicado a la agricultura: estado actual," *Lámpsakos*, no. 22, pp. 86–105, Nov. 2019. [Online]. Available: https://doi.org/10.21501/21454086.3253

[2] B. A. Quintana G., V. R. Pereira Poveda, and C. N. Vega S., "Automatización en el hogar: un proceso de diseño para viviendas de interés social," *Revista Escuela de Administración de Negocios*, no. 78, pp. 108–121, Jul. 2015. [Online]. Available: https://doi.org/10.21158/01208160.n78.2015.1193

[3] C. A. Laura Mamani, "Pruebas de software para microservicios," *Innovación y Software*, vol. 4, no. 1, pp. 151–160, Mar. 2023. [Online]. Available: http://dx.doi.org/10.48168/innosoft.s11.a86

[4] P. Gutiérrez Prada, G. B. de Corso-Sicilia, and W. G. Jiménez-Barbosa, "Impacto social del internet de las cosas (IdC): una reflexión conceptual," *Jangwa Pana*, vol. 21, no. 3, pp. 254–270, Nov. 2022. [Online]. Available: https://doi.org/10.21676/16574923.4719

[5] L. G. Viera Molina and C. O. Guilcaso Molina, "Utilización de sensores iot para la automatización de sistemas de riego," *Dominio de las Ciencias*, vol. 9, no. 4, pp. 1731–1748, 2023. [Online]. Available: https://upsalesiana.ec/ing34ar1r5

[6] L. García, L. Parra, J. M. Jimenez, J. Lloret, and P. Lorenz, "IoT-based smart irrigation systems: An overview on the recent trends on sensors and iot systems for irrigation in precision agriculture," *Sensors*, vol. 20, no. 4, p. 1042, Feb. 2020. [Online]. Available: https://doi.org/10.3390/s20041042

[7] S. K. Mousavi, A. Ghaffari, S. Besharat, and H. Afshari, "Improving the security of internet of things using cryptographic algorithms: a case of smart irrigation systems," *Journal of Ambient Intelligence and Humanized Computing*, vol. 12, no. 2, pp. 2033–2051, Jul. 2020. [Online]. Available: https://doi.org/10.1007/s12652-020-02303-5

[8] B. Parvathi Sangeetha, N. Kumar, A. P. Ambalgi, S. L. Abdul Haleem, K. Thilagam, and P. Vijayakumar, "IoT based smart irrigation management system for environmental sustainability in India," *Sustainable Energy Technologies and Assessments*, vol. 52, p. 101973, Aug. 2022. [Online]. Available: https://doi.org/10.1016/j.seta.2022.101973

[9] J. Yang, A. Sharma, and R. Kumar, "IoT-based framework for smart agriculture," *International Journal of Agricultural and Environmental Information Systems*, vol. 12, no. 2, pp. 1–14, Apr. 2021. [Online]. Available: http://doi.org/10.4018/ijaeis.20210401.oa1

[10] J. Ruan, Y. Wang, F. T. S. Chan, X. Hu, M. Zhao, F. Zhu, B. Shi, Y. Shi, and F. Lin, "A life cycle framework of green IoT-based agriculture and its finance, operation, and management issues," *IEEE Communications Magazine*, vol. 57, no. 3, pp. 90–96, Mar. 2019. [Online]. Available: http://doi.org/10.1109/mcom.2019.1800332

[11] J. M. Talavera, L. E. Tobón, J. A. Gómez, M. A. Culman, J. M. Aranda, D. T. Parra, L. A. Quiroz, A. Hoyos, and L. E. Garreta, "Review of IoT applications in agro-industrial and environmental fields," *Computers and Electronics in Agriculture*, vol. 142, pp. 283–297, Nov. 2017. [Online]. Available: http://doi.org/10.1016/j.compag.2017.09.015

[12] R. Kumar, R. Mishra, H. P. Gupta, and T. Dutta, "Smart sensing for agriculture: Applications, advancements, and challenges," *IEEE Consumer Electronics Magazine*, vol. 10, no. 4, pp. 51–56, Jul. 2021. [Online]. Available: https://doi.org/10.1109/MCE.2021.3049623

[13] M. S. Farooq, S. Riaz, A. Abid, T. Umer, and Y. B. Zikria, "Role of IoT technology in agriculture: A systematic literature review," *Electronics*, vol. 9, no. 2, p. 319, Feb. 2020. [Online]. Available: http://doi.org/10.3390/electronics9020319

[14] V. K. Patil, A. Jadhav, S. Gavhane, and V. Kapare, "IoT based real time soil nutrients detection," in *2021 International Conference on Emerging Smart Computing and Informatics (ESCI)*. IEEE, Mar. 2021, pp. 737–742. [Online]. Available: http://doi.org/10.1109/esci50559.2021.9396860

[15] Visual Studio Code. (2024) *Your code editor. Redefined with AI.* Visual Studio Code. [Online]. Available: https://upsalesiana.ec/ing34ar1r15

[16] Postman. (2024) AI is powered by APIs. APIs are powered by Postman,. Postman. Inc. [Online]. Available: https://upsalesiana.ec/ing34ar1r16

[17] Arduino. (2024) Arduino IDE 2.3.4. Arduino. [Online]. Available: https://upsalesiana.ec/ing34ar1r17

[18] MAMP. (2024) MAMP PRO & MAMPYour web development tool. MAMP GmbH. [Online]. Available: https://upsalesiana.ec/ing34ar1r18

[19] Fritzing. (2024) Fritzing home page. Fritzing Electronics Made Easy. [Online]. Available: https://upsalesiana.ec/ing34ar1r19

[20] Draw.io. (2024) Diagrams on line. J Graph. Ltd. [Online]. Available: https://upsalesiana.ec/ing34ar1r20