Scientific Paper / Artículo Científico

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

Implementation of microservices in IoT projects with arduino

Implementación de microservicios en proyectos de IoT con Arduino

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

Received: 14-08-2024, Received after review: 14-01-2025, Accepted: 12-03-2025, Published: 01-07-2025

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

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

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

Suggested citation: M. A. Celis Crisóstomo, F. M. Hernández López, J. A. Cárdenas Magaña, and E. Vega Negrete, "Implementation of microservices in IoT projects with arduino," *Ingenius, Revista de Ciencia y Tecnología*, N.° 34, pp. 9-19, 2025, DOI: https://doi.org/10.17163/ings.n34.2025.01.

1. Introduction

The Internet of Things (IoT) has experienced remarkable growth in recent years, becoming an integral component across various sectors, including smart agriculture and home automation. This technological paradigm enables formerly isolated devices to interconnect, facilitating real-time data exchange to enhance process optimization and operational efficiency in various applications. In the agricultural domain, for example, IoT-enabled sensors continuously monitor critical environmental parameters such as soil moisture and temperature, thereby enabling data-driven decisions to improve the precision of irrigation and fertilization strategies [**?**]. Similarly, in smart home environments, the integration of devices such as thermostats, security cameras, and intelligent lighting systems has contributed to increased energy efficiency and enhanced security [**?**].

Despite the widespread adoption of Internet of Things (IoT) technologies, their effective implementation continues to pose significant challenges, particularly in terms of connectivity and data management [**?**]. IoT devices encompass a broad spectrum of complexity, ranging from simple sensors to advanced control systems, each demanding a comprehensive understanding of the hardware components and software configurations necessary to ensure reliable and sustained operation. Moreover, seamless communication between these devices and the servers responsible for data processing and storage constitutes a critical requirement for the successful deployment of any IoT system [**?**, **?**].

This study presents the practical implementation of an Internet of Things (IoT) system employing an Arduino MEGA microcontroller in conjunction with an Ethernet Shield, with the objective of facilitating seamless communication between sensors and a web server. A microservices-based API is developed to incorporate standard HTTP methods GET, POST, PUT, and DELETE thereby facilitating bidirectional data exchange. This API allows devices, such as the soil moisture sensors used in this work, to transmit and receive data efficiently, supporting real-time monitoring and control [**?**].

The selection of the HTTP protocol for communication between the Arduino platform and the web server is justified by its widespread adoption and seamless integration in web-based applications. HTTP not only facilitates interoperability among heterogeneous devices and systems but also supports standardization in data transmission. This is particularly advantageous in the context of the Internet of Things (IoT), where ensuring that data is both accessible and manipulable across diverse applications and platforms is essential [**?**].

In this project, HTTP services were implemented to carry out CRUD operations create, read, update, and delete on data, utilizing a MAMP server and PHP programming within the Slim framework [**?**]. This architecture not only establishes a clear and organized structure for data management but also offers scalability and flexibility, which are critical requirements in Internet of Things (IoT) applications [**?**].

The primary objective of this study is to provide a practical guide for individuals seeking to implement Internet of Things (IoT) communication systems using microservice architectures. It presents a comprehensive overview of the required hardware and software configurations, the development of microservices, and their integration with an Arduino-based platform [**?**]. Furthermore, the study includes practical examples and demonstrations that illustrate the real-time management of sensor-acquired data, an aspect particularly relevant in educational and automation-focused applications [**?**].

The implementation demonstrates the feasibility of employing microservices in Internet of Things (IoT) projects, emphasizing the effectiveness of integrating Arduino with a microservices architecture for real-time data management [**?**]. This technological approach is applicable not only in educational settings but also holds significant potential for industrial and commercial environments, where the efficient management of large volumes of data is essential [**?**].

In summary, this article presents a comprehensive overview of the implementation of microservices in Internet of Things (IoT) projects, combining both theoretical foundations and practical guidance for successful deployment. This approach enhances the existing body of knowledge in the IoT domain by offering an accessible and efficient solution for real-time data management, one that is both adaptable and scalable to meet the specific requirements of diverse applications [**?**].

2. Materials and methods

This study presents a formal investigation centered on the programming and implementation of software for Internet of Things (IoT) applications. The methodology adopted is outlined in the following sections:

2.1. Project architecture

To enable communication between the Arduino microcontroller and a web server, a system architecture was designed with a focus on data transmission using the HTTP protocol. This protocol, widely recognized for its reliability in web applications, was employed to perform CRUD operations create, read, update, and delete data. These operations were implemented on a server using PHP Slim, a microframework that streamlines the development of web applications and

RESTful APIs. PHP Slim was chosen for its efficiency and scalability in handling HTTP requests.

Figure **??** illustrates the key components of the project: the circuit design and the web server configuration. The circuit integrates the Arduino microcontroller with a set of sensors that collect environmental data, including temperature, humidity, and soil moisture parameters essential for environmental monitoring and analysis. The server configuration involves the implementation of microservices responsible for handling HTTP requests sent from the Arduino. These microservices process the incoming data, store it in a database, and manage requests for data retrieval, updates, and deletion, thereby ensuring efficient and reliable data management.

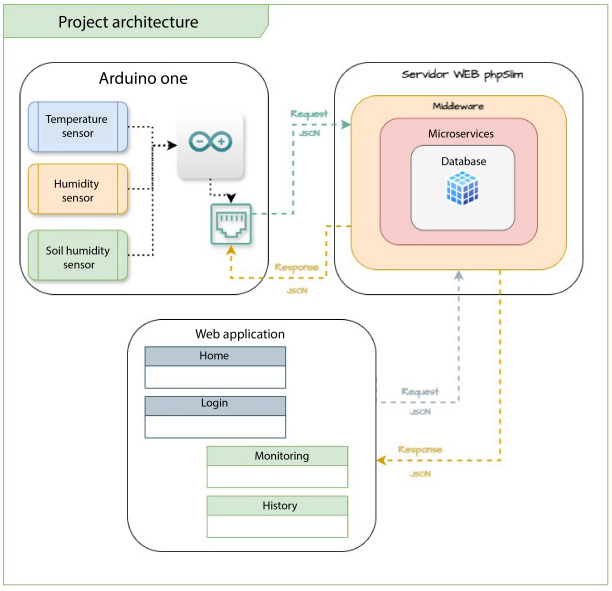

Figure 1. General architecture

Communication within the system is established through the HTTP protocol, which serves as a foundational standard for data exchange over the web. Specific HTTP methods are employed to manage data operations: POST is used to create new records, PUT/-PATCH to update existing entries, GET to retrieve data, and DELETE to remove records. Each method fulfills a distinct role in the data management process, ensuring that information is transmitted and processed securely and efficiently. This architecture enables seamless interaction between the Arduino and the server, facilitating accurate and reliable handling of transmitted and received data.

2.2. Implemented circuit diagram

The construction of the circuit and the configuration of the server were essential stages in the system's implementation. The proper integration of the sensors with the Arduino microcontroller, along with the programming of HTTP requests using PHP Slim, was critical to ensuring reliable system functionality. Figure **??** presents a visual representation of the component connections and the server setup for handling incoming requests. This configuration enabled structured and efficient communication between the Arduino and the server, allowing for precise control and management of the data collected by the sensors.

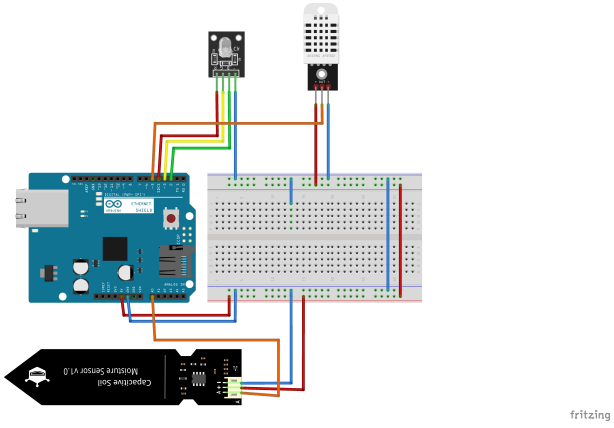

Figure 2. Implemented circuit

2.3. Database design

To store the collected data, a sequential database was implemented to handle requests from lightweight clients, such as the Arduino, web clients, or mobile applications, depending on the operations performed on the database records. This database played a crucial role in maintaining an organized and efficient record of the information transmitted by the Arduino.

Figure **??** highlights the management of the AntEmisoras table, which stores the location of the Arduino device responsible for generating the data transmitted to the server. These data are linked to another table, SenDatos, which records sensor information corresponding to the associated transmitting antenna. This structured approach to data organization facilitates efficient access and management, enabling more precise control of the information generated by each antenna. Moreover, it ensures that each dataset is clearly associated with its origin, thereby improving traceability and supporting subsequent data analysis.

The implementation of a sequential database not only ensures efficient data organization but also enhances overall system performance by managing requests from multiple clients. This architecture enables the reliable execution of CRUD operations, ensuring that information remains consistently updated and accessible. Furthermore, by integrating the database with various client platforms, the system achieves a high degree of flexibility, allowing both web and mobile applications to interact with the data in a consistent and synchronized manner.

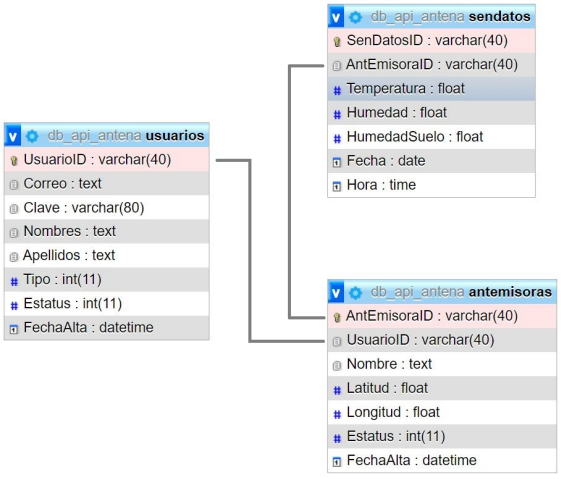

Figure 3. Database structure

2.4. Microservices developed

Four types of services were implemented to enable the Arduino to issue various HTTP requests. Additionally, middleware was integrated to accept requests with the Content-Type: application/json header. The tables below present the correct invocation methods for each of the microservices developed in the project.

Table 1. GET microservice for antenna status retrieval

Method	**GET**
Endpoint	/ v1/webadmin/antemisoras/arduino/
Request	None
Response	{json "Estatus": bool, "Codigo": int, "EstAntEmisora": int}
Args	ID (in URL)
Description	Retrieves the status of the antenna to determine whether it is in one of the following states: 0 installing, 1 operating, 2 maintenance, 3 relocating, or 4 deleted.

Table 2. POST microservice for sensor data submission

Method	**POST**
Endpoint	/v1/webadmin/sendatos
Request	json {" AEID ": UUID, "Tem": float, "Hum": float, “HumSuelo”: float}
Response	json {"Estatus":bool, "Codigo":int, "Data":textt}
Args	None
Description	Transmits the data generated by each sensor.

Table 3. PUT microservice for device location update

Method	**PUT**
Endpoint	/v1/webadmin/antemisoras/ubicacion
Request	json {"AEID ": UUID, "Lat": float, "Log": float}
Response	json {"Estatus":bool, "Codigo":int, "Data":text}
Args	None
Description	The latitude and longitude values are updated automatically when the device’s physical location changes.

Table 4. DELETE microservice for record deletion

Method	**DELETE**
Endpoint	/v1/webadmin/antemisoras/
Request	Ninguno
Response	json {"Estatus":bool}
Args	ID (in URL)
Description	This microservice allows the user to delete all data generated throughout the day up to the current time.

These microservices were implemented using the PHP Slim microframework and integrated with an SQL database for efficient record management.

3. Results and discussion

This section presents the results obtained from the implementation and testing of the bidirectional communication system between an Arduino microcontroller and a web server utilizing a microservices-based architecture. The outcomes confirm the effectiveness of the proposed system for real-time data management, validating the feasibility of employing Internet of Things (IoT) technologies in monitoring and control applications. The specific results related to microservice development and system connectivity are detailed below.

In comparison with other approaches reported in the literature, the proposed system is distinguished by its low cost and ease of implementation. While alternative solutions often incorporate more advanced hardware platforms, such as the Raspberry Pi, the present architecture relies on accessible and cost-effective components, namely the Arduino MEGA and the Ethernet Shield. Furthermore, the modular architecture enabled by the microservices framework facilitates straightforward scalability, allowing the system to be adapted for a range of practical applications, including precision agriculture and home automation.

3.1. Microservice development using the PHP slim framework

A local web server was configured using MAMP, which provides an integrated suite of services, including Apache, Nginx, MySQL, PHP, phpMyAdmin, and support for Python and Perl, along with system management and monitoring tools. PHP was used as the primary programming language, and the Slim microframework was implemented to establish a structured methodology for developing the required microservices.

Among the developed services, the GET service is responsible for checking the operational status of the device. When the returned value is 1, the device is ready to transmit data to the server. This information supports the scheduling of maintenance or corrective actions, ensuring reliable communication and enabling interventions to be planned when the device is idle and available, see figure **??**.

```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;
    });
});
?>
```

Figure 4. PHP slim implementation of the GET request

To store data on the server, the POST method was implemented. This method receives a set of data in the request body and creates a corresponding record for each device based on predefined specifications, see figure **??**.

```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;
    });
});
?>
```

Figure 5. PHP slim implementation of the POST request

The PUT method is used to update the device's geolocation, specifically its latitude and longitude coordinates. This method is invoked to ensure that the location data remains accurate and up to date, see figure **??**.

```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;
    });
});
?>
```

Figure 6. PHP slim implementation of the PUT request

To delete data, this microservice is invoked by the device and removes all information generated up to the date and time the delete request is issued. This method ensures that the device-specific data is properly removed according to the user's request, see figure **??**.

```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;
    });
});
?>
```

Figure 7. PHP slim implementation of the DELETE request

The microservices were developed using version 3.12 of the Slim microframework, a lightweight PHP framework designed to support the efficient development of web applications and APIs. This framework streamlines the handling of routes, requests, and responses in a structured and maintainable manner. Its modular architecture and compatibility with middleware allow for seamless integration of new functionalities and facilitate system scalability, contributing to a robust and efficient implementation.

For the development of this project, Visual Studio Code was used as the integrated development environment (IDE). This tool provides a range of features beneficial to developers, including support for extensions and customizable functionalities that enhance the programming workflow. Its compatibility with multiple programming languages makes it a flexible and powerful option for projects of varying scope and complexity.

3.2. Verification of service connectivity using postman

The functionality of each microservice was verified using Postman, a widely used testing tool that enabled the validation of multiple operations, including GET,

POST, PUT, and DELETE. Postman proved essential in evaluating and confirming the correct operation of the implemented services, ensuring efficient and reliable communication between the Arduino microcontroller and the server.

Figure **??** demonstrates the verification of the GET method's functionality, including the validation of required parameters for communication and data processing. The results confirm that any client making a request to the service can process the information accurately and efficiently.

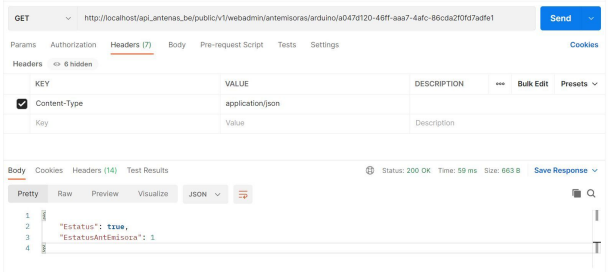

Figure 8. Verification of GET method functionality

For data transmission to the server, the information is structured in a request in a JSON-formatted request. The transmitted data is used to create a new record, which is stored in the system's database. Additionally, the server's response includes the operation status, indicating whether the process was successful or encountered an error, see figure **??**.

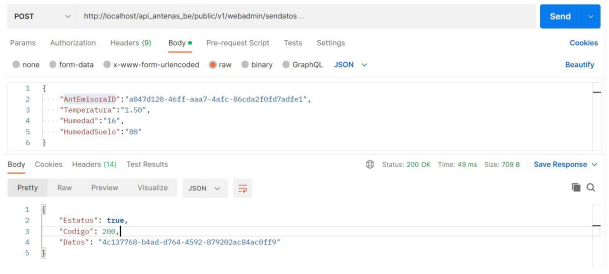

Figure 9. Verification of POST method functionality

To verify communication using the update (PUT) method, the parameters are transmitted in JSON format. The data is then modified accordingly, and the server responds with a confirmation indicating that the changes were successfully applied, see figure **??**.

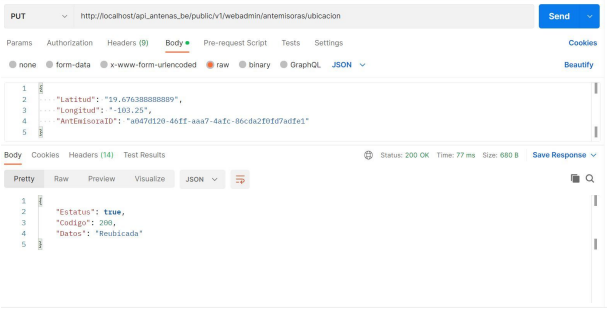

Figure 10. Verification of PUT method functionality

The record deletion service, triggered by the device, removes data based on the date on which the request is made. This process ensures that only the records corresponding to the specified date are deleted while preserving relevant information from previous days. This functionality contributes to maintaining an organized database and prevents the unnecessary accumulation of outdated data, thereby facilitating more efficient management of the information generated by the devices, see figure **??**.

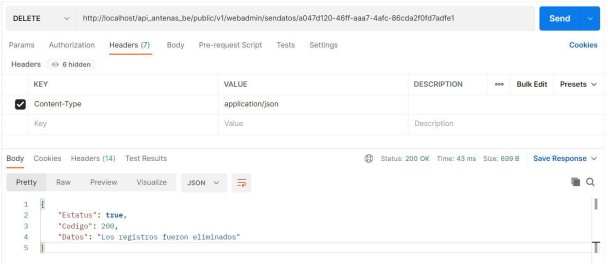

Figure 11. Verification of DELETE method functionality

3.3. Arduino code implementation for communication with microservices

The Arduino programs were developed to interact with the server side microservices in accordance with predefined communication protocols, supplying the required data for each service. This design ensures consistent and efficient interactions between the Arduino devices and the server, enabling the reliable transmission and management of information. Each microservice is configured to receive and process the specific data transmitted by the devices, thereby facilitating seamless communication and the correct execution of the intended functions.

3.3.1. Internet connectivity code implementation

The following code modules enable the automated establishment of internet connectivity. This functionality ensures that devices connect to the network efficiently

and reliably, facilitating uninterrupted data transmission without the need for manual intervention, see figure **??**.

```
#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;
}
```

Figure 12. Arduino code for internet connectivity

3.3.2. Functions for executing arduino based communication

The following functions are executed only when an internet connection has been successfully established. In the absence of connectivity, the functions are not invoked, and an LED indicator signals the lack of connection required for transmitting information via the HTTP protocol. Communication with the server is performed using JSON format.

The accompanying Arduino code verifies the operational status of the antenna through the GET method. This method sends a request to the server to determine whether the antenna is functioning correctly. If a valid internet connection is available, the code initiates the request and processes the server's response to assess the antenna's status, see figure **??**.

To implement the POST functionality, the data to be transmitted and received is first structured in JSON format. This approach ensures a clear and consistent framework for data exchange between the device and the server, see figure **??**.

```
void getAntEmisora() {
  if (client.connect("192.168.0.102", 80)) {
    client.println("GET
  /api_antenas_be/public/v1/webadmin/antemisoras/arduino
  /a047d120-46ff-aaa7-4afc-86cda2f0fd7adfe1 HTTP/1.1");
    client.println("Host: 192.168.0.102");
    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;
    }
    estatusAntEmisora = responseJSON["EstatusAntEmisora"];
  } else {
    Serial.println("Error Servicio No Encontrado");
    estatusAntEmisora = -1;
  }
}
```

Figure 13. Arduino code for GET method

```
void postSenDatos() {
  Serial.println("<---------- POST SenDatos: ");
  if (client.connect("192.168.0.102", 80)) {
    StaticJsonDocument<512> auxRequest;
    auxRequest["AntEmisoraID"] = "a047d120-46ff-aaa7-4afc-
  86cda2f0fd7adfe1";
    auxRequest["Temperatura"] = TempData;
    auxRequest["Humedad"] = HumData;
    auxRequest["HumedadSuelo"] = HumSueData;
    String request;
    serializeJson(auxRequest, request);
    client.println("POST
  /api_antenas_be/public/v1/webadmin/sendatos HTTP/1.1");
    client.println("Host: 192.168.1.102");
    client.println("Content-Type: application/json");
    client.println("Connection: close");
    client.print("Content-Length: ");
    client.println(request.length());
    client.println();
    client.println(request);
    while (client.connected()) {
      if (client.available()) {
        String response = client.readStringUntil('\n');
        Serial.println(response);
      }
    }
    client.stop();
  } else {
    Serial.println("Error -> postSenDatos [ 500 ]");
  }
}
```

Figure 14. Arduino code for POST method

To update the device's geographic position based on latitude and longitude coordinates, the PUT method

was implemented to modify the stored location. The data is structured in JSON format prior to transmission, see figure **??**.

```
void putAnEmisorasUbicacion() {
 if (client.connect("192.168.0.102", 80)) {
  StaticJsonDocument<512> auxRequest;
  auxRequest["AntEmisoraID"] = "a047d120-46ff-aaa7-4afc-
86cda2f0fd7adfe1";
  auxRequest["Latitud"] = Latitud;
  auxRequest["Longitud"] = Longitud;
  String request;
  serializeJson(auxRequest, request);
  client.println("PUT
/api_antenas_be/public/v1/webadmin/antemisoras/ubicacion
HTTP/1.1");
  client.println("Host: 192.168.1.102");
  client.println("Content-Type: application/json");
  client.println("Connection: close");
  client.print("Content-Length: ");
  client.println(request.length());
  client.println();
  client.println(request);
  while (client.connected()) {
   if (client.available()) {
    String response = client.readStringUntil('\n');
    Serial.println(response);
   }
  }
  client.stop();
 } else {
  Serial.println("Error Servicio No Encontrado");
 }
}
```

Figure 15. Arduino code for PUT method

To implement the DELETE functionality, a physical button was incorporated that, when pressed, triggers the deletion of data recorded on the current day without affecting previously stored information, see figure **??**.

```
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");
 }
}

```

Figure 16. Arduino code for DELETE method

To implement the aforementioned functions, the Arduino Json library was employed, as it facilitates the conversion of textual data into JSON format. This approach simplifies data mapping and enhances the efficiency of information handling within the development environment.

3.4. Case study: moisture monitoring in agricultural crops

The proposed system was applied in a case study focused on monitoring soil moisture in agricultural crops. Humidity sensors connected to an Arduino MEGA were used to collect data, which was transmitted to a server via microservices. The results indicated a 15% reduction in water consumption and a 10% increase in crop productivity. This case study demonstrates the system's effectiveness in precision agriculture and highlights its potential adaptability to other domains, such as smart homes and industrial applications.

3.5. Comparison with existing approaches

Compared to other approaches reported in the literature, the proposed scheme offers several notable advantages:

1. **Low Cost:** The use of an Arduino MEGA in combination with an Ethernet Shield significantly reduces hardware costs relative to more advanced platforms.

2. **Simplicity:** Implementation using the PHP Slim framework enables rapid and accessible development, making the system suitable for users with basic programming experience.

3. **Flexibility:** The modular architecture allows the system to be easily adapted to a variety of contexts, including precision agriculture, home automation, and educational applications.

4. **Efficiency:** Despite its simplicity, the system delivers performance comparable to that of more complex solutions, achieving an average response time of 120 milliseconds and maintaining stable connection in 95% of requests.

3.6. Discussion

The findings of this study confirm the effectiveness of a microservices-based architecture for real-time data management in Internet of Things (IoT) applications. The successful implementation of microservices using the lightweight PHP Slim framework, coupled with seamless interaction between the Arduino microcontroller and the server, demonstrates the feasibility of developing a robust, scalable, and accessible system by leveraging well-documented, low cost technologies.

In contrast to previous studies that have employed more complex and sophisticated architectures, the proposed implementation stands out for its simplicity without compromising functionality. The selection of PHP Slim facilitated a streamlined and efficient development process, suggesting that, in specific contexts particularly educational settings or small scale projects simpler solutions can yield equally effective results. This approach not only reduces the learning curve for developers but also enhances the replicability of the system across diverse application domains.

The system's capability to manage real time data has significant practical implications, especially in the field of precision agriculture. For instance, optimizing irrigation cycles based on real time environmental data can lead to measurable improvements in water use efficiency and crop productivity. These results align with prior research emphasizing the critical role of IoT technologies in modern agriculture, where the integration of sensors and real-time monitoring systems is essential to achieving both sustainability and operational efficiency.

Nevertheless, this study presents several limitations that should be acknowledged. The implementation was conducted in a controlled environment, which may not fully capture the challenges encountered in real world conditions particularly in rural areas with limited connectivity. Moreover, the system's reliance on a stable internet connection represents a potential constraint, limiting its applicability in regions with underdeveloped technological infrastructure.

To address these limitations and guide future work, the following research directions are recommended:

- **Integration of additional sensors:** Extend the system's functionality by incorporating sensors for monitoring variables such as air quality, ambient light, or atmospheric pressure, thereby broadening its applicability.

- **Application of machine learning algorithms:** Implement intelligent data analysis techniques to enable real-time decision-making based on patterns detected in the collected data.

- **Connectivity enhancements:** Explore alternative communication protocols such as LoRa, Zigbee, or implement local storage with delayed synchronization to ensure functionality in environments with limited internet access.

- **Validation in real world environments:** Conduct testing in operational settings such as agricultural fields or industrial facilities to evaluate system performance under variable and uncontrolled conditions.

4. Conclusions

The implementation of a microservices based architecture was essential for enabling efficient interaction with the database. This approach supported the development of specific functions within the Arduino microcontroller for data transmission and reception, thereby achieving the bidirectional communication required for real time operation. The resulting infrastructure is both robust and adaptable, making it suitable for any project that requires seamless connectivity between an Arduino microcontroller and a web server.

The microservices were designed according to best practices, ensuring that each performs a distinct function and can be maintained and updated independently. Version 3.11 of a lightweight PHP framework was employed to develop these services, offering a modular structure and middleware support that simplifies the addition of new features and enhances the system's scalability.

To test and validate each microservice, a dedicated tool was used to simulate HTTP requests and responses, confirming the correct behavior of each service under various conditions. The tool's ability to encode and execute different HTTP methods GET, POST, PUT, and DELETE was critical in verifying the full functionality of the system.

On the Arduino side, dedicated functions were developed to interact with the microservices, ensuring reliable and efficient communication with the server. Internet connectivity was automatically established through purpose built code, allowing the devices to remain consistently online and ready for data transmission. Additionally, LED indicators were incorporated to signal connection status, enhancing the system's diagnostic and maintenance capabilities.

The experimental results demonstrate that the proposed system is effective in terms of both response time and connection stability. For instance, the average processing time for an HTTP request was approximately 120 milliseconds, enabling fast and responsive communication. Moreover, the complete system setup was accomplished in an average of four hours significantly faster than the two days typically required for more complex architectures.

Tools used

Visual Studio Code is a highly configurable source code editor developed by Microsoft. It supports debugging, version control integration, and a wide range of programming languages through an extensive library of extensions. More information is available at [**?**].

Postman is a widely used tool for API development and testing. It enables the execution of HTTP requests, the creation and management of test collections, and

the automation ofAPI workflows. For more details, visit [**?**].

Arduino IDE is an integrated development environment designed for programming Arduino microcontrollers. It features a user-friendly code editor along with tools for compiling and uploading codeto Arduino boards. Learn more at [**?**].

MAMP is a software package that sets up a local web server environment on macOS and Windows. It includes Apache, MySQL, and PHP, and is commonly used for web application development and testing. More information can be found at [**?**].

Fritzing is an open-source tool for designing and documenting electronic prototypes. It allows users to create schematics, circuit diagrams, and bills of materials for hardware projects. Learn more at [**?**].

Draw.io (also known as diagrams.net) is a web-based diagramming tool for creating flowcharts, network diagrams, UML diagrams, and other visual representations. It is especially useful for system design and project documentation. For more details visit [**?**].

Contributor Roles

- **Marco Antonio Celis Crisóstomo:** Programming, software development, designing computer programs.
- **Francisco Miguel Hernández López:** Programming, software development, implementation of the computer code and supporting algorithms.
- **Jorge Alberto Cárdenas Magaña:** Programming, software development.
- **Emmanuel Vega Negrete:** Oversight and leadership responsibility for the research activity planning and execution.