

IoT-Based Seaweed Cultivation Monitoring System with Data Transmission Protection Using JSON Web Token (JWT) on ESP32

Iwan Fitrianto Rahmad^{a,1}, Yudho Andika^{b,2}, Vidi Agung Fragastia^{c,3}, Elsa Aditya^{d,4}

^{a,c,d} Universitas Potensi Utama, Jl. KL Yos Sudarso Km 6,5 No. 3A, Medan, 20241, Indonesia

^b Universitas Malikussaleh, Cot Tengku Nie Reuleut Muara Batu, Aceh Utara, 24351, Indonesia

iwanditrah@yahoo.com ¹, yudhoandika@unimal.ac.id ², vidiagoeng@gmail.com ³,

elsaaditya15@gmail.com ⁴

ABSTRACT

Security in the communication of IoT devices is a critical concern in seaweed cultivation monitoring systems, particularly when operating over open networks. This study presents the design and implementation of an IoT-based monitoring system for seaweed cultivation, incorporating data transmission protection on the ESP32 device through the use of JSON Web Token (JWT). The system monitors key parameters influencing seaweed growth—such as temperature, salinity, pH, and turbidity—and transmits this data to a server via HTTP/HTTPS protocols. JWT is employed for authentication and authorization, ensuring that only legitimate devices are permitted to send data to the server. The research methodology encompasses requirement analysis, system architecture design, hardware and software implementation, followed by functionality, security, and performance testing. Experimental results demonstrate that the proposed system successfully acquires sensor data and transmits it in real time to the server, while effectively rejecting unauthorized attempts lacking valid tokens. The application of JWT significantly enhances communication security with minimal impact on ESP32 performance. Overall, the proposed IoT-based Seaweed Cultivation Monitoring System with JWT protection provides an effective solution for safeguarding data integrity and authenticity in seaweed farming applications.

Keywords : internet of things, esp32, json web token, data security, seaweed monitoring, water quality

Submitted:

Article Info :
Reviewed:

Accepted :

Copyright © 2026 – CSRID Journal. All rights reserved.

1. INTRODUCTION

Internet of Things has emerged as one of the main technological solutions to enable M2M communications, where physical devices connected to the internet are able to autonomously collect and exchange information [1]. The number of IoT devices is increasing and is predicted to reach 39.6 billion by 2033 [2]. This indicates that IoT plays a vital role in life and has become an essential component, from home automation systems to natural resource management and industrial sectors [3], [4]. Yet, ensuring security about data transmission in between devices is one of the challenges in IoT implementation [5]. This challenge is faced when devices perform their operations on unstable networks or under limited capacity conditions [6]. Being able to efficiently transmit small-sized data [7], [8], performing very well over low-bandwidth networks, and being of low power consumption are just a few reasons why MQTT is chosen to be an IoT system's communication protocol [9].

Conventional IoT systems are usually prone to security vulnerabilities in the case of authentication based on sensor data only. Eavesdropping [10] or SQL injection attacks [11] may lead to unauthorized access to sensitive data. In order to enhance security, therefore, JSON Web Token-based authentication has been considered more secure [12], [13]. JWT allows the

verification of devices' identities without repeatedly transmitting sensitive information [14], while providing a mechanism to expire tokens and prevent their further reuse. Therefore, each request for data or each command is guarded with an encrypted token with a specific period of validity. Upon expiration, the system would automatically reject such a request and seek a new token, thereby enhancing security in communication as well as sensor data management in IoT infrastructures [15]. JWT is used not only for device authentication, but also to reduce credential exposure risk and prevent replay attacks, which are common vulnerabilities in conventional IoT authentication mechanisms.

IoT technology is increasingly being used in aquaculture to enable remote monitoring and control of environmental conditions. In seaweed farming, the large potential of IoT-based monitoring systems in monitoring water quality parameters in real-time significantly influences their optimal growth [16]. Integrated with sensors and an ESP32 microcontroller, the system can measure variables automatically: pH, salinity, turbidity, and water temperature, just like its implementation in the cultivation of shrimp ponds and other aquaculture commodities [17], [18]. The ESP32 microcontroller has also been proven to be effective in environmental monitoring systems for oyster mushroom cultivation and hydroponic plants, integrating temperature, humidity, and pH sensors for accurate regulation of environmental conditions [19], [20]. Similar implementations of IoT in fish and mangrove crab farming have been carried out by using TDS sensors, DS18B20 temperature sensors, and NodeMCU ESP32 for real-time data acquisition [21], [22]. These works demonstrate how IoT-based monitoring can raise efficiency and productivity in aquaculture by means of continuous data exchange and remote supervision [23], [24]. It can find anomalies in environmental parameters earlier and conduct interventions on time to reduce losses and optimize production [25].

Present development trends in IoT for aquaculture are also oriented to the use of predictive analytics and real-time automatic control to optimize resource use [26]. However, most of the existing systems are focused on data acquisition and visualization, while security-related aspects of data transmission, in particular authentication and protection of sensor data sent from field devices to the server, are not adequately addressed. This paper therefore presents an IoT-based seaweed cultivation monitoring system using ESP32 with data transmission protection based on JWT. The system is designed to make sure only authenticated devices can send data and to guarantee secure and reliable communication between nodes and a server. Experimental results demonstrate that the proposed system can acquire sensor data and transmit it in real time, while rejecting access attempts without a token or with an invalid token. The performance test has shown that the implementation of JWT can enhance the security of communication with minimal influence on the ESP32 performance and is an effective solution to maintain the integrity and authenticity of data in an IoT-based seaweed cultivation application.

The main novelty of this research lies in the implementation of a JSON Web Token (JWT)-based authentication mechanism directly on a resource-constrained ESP32 device for IoT-based aquaculture monitoring, specifically seaweed cultivation. Unlike most existing IoT monitoring systems in aquaculture that focus primarily on sensor data acquisition and visualization, this study emphasizes secure data transmission through token-based authentication with expiration control.

Furthermore, previous studies generally rely on static credentials or unsecured HTTP communication, whereas this research integrates dynamic token validation, token expiration handling, and rejection of unauthorized devices at the application layer. The proposed approach demonstrates that JWT can be practically applied on low-power microcontrollers with minimal performance overhead, making it a viable lightweight security solution for real-world IoT deployments in aquaculture environments.

Several studies have implemented IoT-based monitoring systems in aquaculture, focusing on parameters such as temperature, pH, salinity, and turbidity using ESP32 or similar microcontrollers. These systems successfully improve monitoring efficiency and enable real-time data visualization.

However, most existing works emphasize sensing and data transmission aspects, while security mechanisms are either limited or not explicitly discussed. Authentication is often implemented using static credentials or unsecured communication protocols, which are vulnerable to unauthorized access and replay attacks.

Although some studies address IoT security, they primarily focus on cloud-side solutions or computationally intensive encryption techniques, which are not suitable for low-resource devices. Therefore, there is a research gap in implementing a lightweight, token-based authentication mechanism directly on ESP32-based IoT nodes, particularly in aquaculture applications. This study addresses this gap by integrating JWT-based authentication and evaluating its impact on system performance.

2. RESEARCH METHODOLOGY

This study is an applied research that focuses on the design and implementation of a secure data transmission system for IoT-based water quality monitoring using ESP32 and JSON Web Token (JWT). The stages of the research are illustrated in Figure 1.

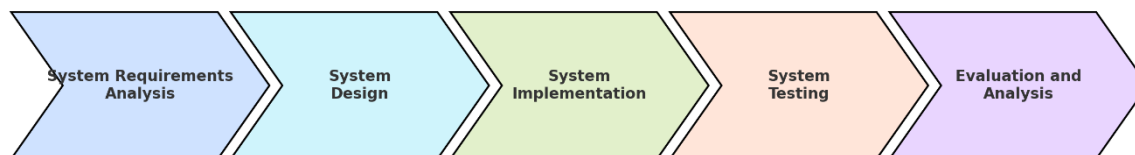


Figure 1. Research Stages

The first step of this research is system requirements analysis, which aims to identify all components needed in the development of a secure IoT data transmission system based on ESP32 and JWT. This analysis includes determining the type of data to be transmitted, the hardware and software to be used, communication requirements, and data security aspects. The system is designed to transmit data from water quality sensors, namely temperature, humidity, pH, and turbidity.

The second stage is system design. The main device used is the ESP32 as an IoT node connected to the aforementioned sensors. For the software, Arduino IDE is used for microcontroller programming, while Node.js with the Express framework is used on the server side. The overall block diagram of the data transmission system is shown in Figure 2.

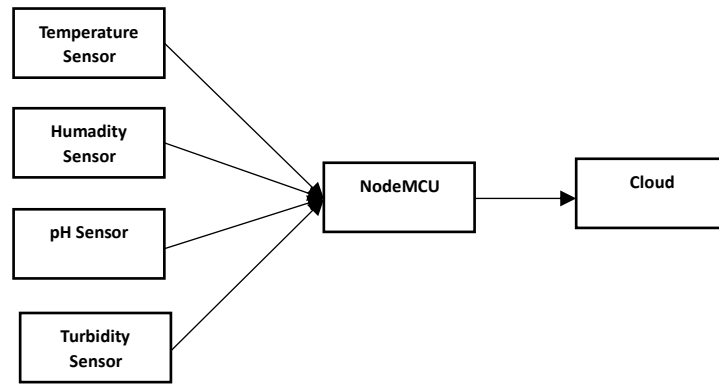


Figure 2. Block Diagram of the Data Transmission System

In the third stage, JWT is employed to secure data transmission between the ESP32 and the server as an authentication mechanism. The communication protocol used is HTTP or HTTPS, with JSON-based data structures sent using the POST method. The server is designed to verify the JWT token received via the Authorization header and to store only valid data in the storage system. This requirement analysis and security design form the basis for the detailed system design. The flowchart of the secure data transmission process is shown in Figure 3.

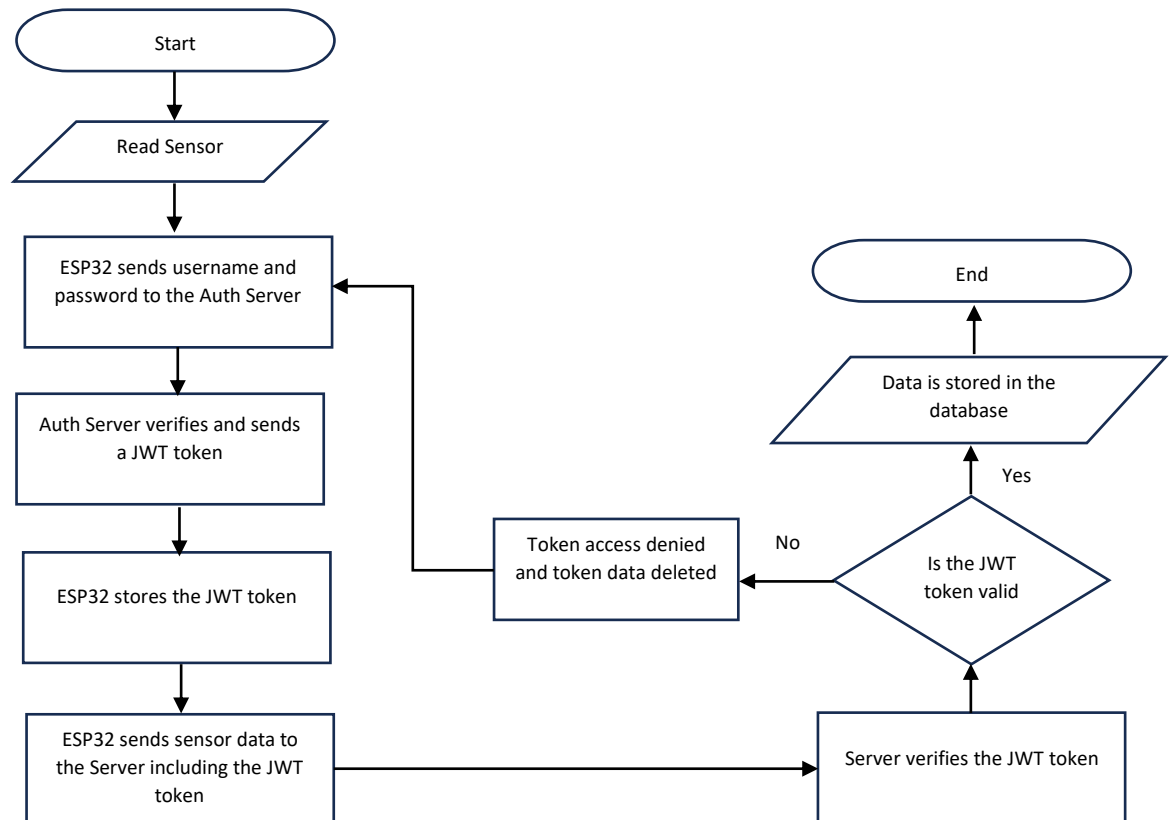


Figure 3. Flowchart of Secure Data Transmission

System evaluation in this study is carried out through three main types of testing: functional testing, security testing, and performance testing. Each test has a specific objective to ensure that the JWT-based authentication system operates correctly, securely, and efficiently in an IoT device environment, particularly on the ESP32.

A. Functional Testing

Functional testing is conducted to ensure that all system features and functional requirements are implemented and operate as expected. The test cases are derived from the previously defined functional specifications. The main objective is to verify that all core processes—from token request, token validation, to sensor data publication to the server—can be executed consistently without errors.

B. Security Testing

Security testing focuses on evaluating the accuracy and robustness of the system in performing authentication and validation of the entities involved. The main goal is to observe how the system handles token-based authentication scenarios under various conditions. Security testing is divided into several scenarios:

a. Transmission of an Invalid Token

In this scenario, the ESP32 prepares sensor data along with a forged or invalid token (for instance, a token that has been modified or obtained from an unauthorized source). The server is expected to validate the token, reject the request, and return an appropriate error response that clearly indicates the failure of authentication.

b. Incorrect Handshake between ESP32 and Server

This test simulates situations where the handshake process between the ESP32 and the server does not proceed correctly, due to issues such as incorrect request headers, improper token format, or skipped authentication steps. The server should be able to detect these inconsistencies, reject the connection or incoming data, and provide an error message indicating that the handshake process has failed.

c. Transmission with a Valid Token

This scenario ensures that the system behaves correctly when the ESP32 sends sensor data using a valid token. The server is expected to accept and process the data properly and respond with a success message. This test confirms that the authentication and authorization processes run normally when all components are configured correctly.

In addition, security testing also includes scenarios involving invalid sensor data and expired tokens, both with and without prior disconnection, to ensure that the server can reliably reject expired or invalid tokens and enforce re-authentication.

C. Performance Testing

Performance testing is conducted to compare the system behavior under two configurations: without JWT and with JWT. In the configuration without JWT, authentication and data transmission are carried out using conventional methods such as username and password, without additional security layers such as token expiration. In the configuration with JWT, authentication is performed using tokens with stronger security characteristics, including a token expiration mechanism that

limits access after a certain period and forces periodic token renewal. The objective of this test is to evaluate and compare the performance of both configurations, with particular attention to response time and system efficiency in handling authentication requests and sensor data publication.

The comparison highlights the overhead introduced by the use of JWT and its impact on overall system performance and security. Performance testing results show that the use of JWT adds only a slight overhead to response time and system efficiency, even on devices with limited memory capacity such as ESP32-based NodeMCU boards. Token management testing also confirms that token renewal works effectively when tokens expire, ensuring that the system remains both secure and operationally feasible for IoT-based seaweed cultivation monitoring applications.

3. RESULTS AND DISCUSSION

A. Functionality Testing

Functionality testing was carried out to evaluate the extent to which the system fulfills the predefined functional requirements and operates correctly. On the ESP32 side, the program was developed to perform a handshake with the server according to the scenario shown in Figure 4.

```

46 void performLogin() {
47     HTTPClient http;
48     http.begin(serverUrl + "/login");
49     http.addHeader("Content-Type", "application/json");
50
51     StaticJsonDocument<200> doc;
52     doc["username"] = "admin";
53     doc["password"] = "admin";
54
55     String requestBody;
56     serializeJson(doc, requestBody);
57
58     int httpCode = http.POST(requestBody);
59
60     if(httpCode == HTTP_CODE_OK) {
61         String payload = http.getString();
62         DynamicJsonDocument response(1024);
63         deserializeJson(response, payload);
64
65         jwtToken = response["token"].as<String>();
66         Serial.println("Token received: " + jwtToken);
67     } else {
68         Serial.println("Login failed");
69     }
70
71     http.end();
72 }

```

Figure 4. ESP32 code connecting to the server using the HTTP protocol

On the server side, a corresponding scenario was implemented to ensure that the handshake process can be executed correctly, as illustrated in Figure 5.

Figure 5. Server response when the handshake is successful

```

10121:41.961 -> ...Connected to WiFi.
10121:49.403 -> Token received: syN8h3K0J1D1I1n1k3C1G1e1m1WJ9J...eY1c1v1Ym2t121e1m1f3k1u1w1m1a1n1e1m1W1u1n1e1m1Q1u1Q1W1Q1T1A1L1C1O1E1S1H1D1O1T1W1D19.1n1C1W1
10121:53.789 -> Server response: {"status":"success", "receivedData":{"temperature":128.6, "humidity":74.7, "ph":9.8, "turbidity":32.2, "deviceId":"ESP32-1", "serverTimestamp":16292239.880}}
10121:54.121 -> Server response: {"status":"success", "receivedData":{"temperature":128.6, "humidity":74.7, "ph":9.8, "turbidity":32.2, "deviceId":"ESP32-1", "serverTimestamp":16292239.880}}
10121:54.616 -> Server response: {"status":"success", "receivedData":{"temperature":120.2, "humidity":71.0, "ph":11.6, "turbidity":61.1, "deviceId":"ESP32-1", "serverTimestamp":16292239.880}}
10121:58.626 -> Server response: {"status":"success", "receivedData":{"temperature":126.8, "humidity":77.6, "ph":6.8, "turbidity":78.9, "deviceId":"ESP32-1", "serverTimestamp":16292239.880}}
10121:58.644 -> Server response: {"status":"success", "receivedData":{"temperature":126.8, "humidity":77.6, "ph":6.8, "turbidity":78.9, "deviceId":"ESP32-1", "serverTimestamp":16292239.880}}
10121:59.791 -> Token received: syN8h3K0J1D1I1n1k3C1G1e1m1WJ9J...eY1c1v1Ym2t121e1m1f3k1u1w1m1a1n1e1m1W1u1n1e1m1Q1u1Q1W1Q1T1A1L1C1O1E1S1H1D1O1T1W1D19.1n1C1W1
10121:126.267 -> Server response: {"status":"success", "receivedData":{"temperature":123.5, "humidity":76.3, "ph":8.3, "turbidity":49.3, "deviceId":"ESP32-1", "serverTimestamp":16292239.880}}
10121:137.135 -> Server response: {"status":"success", "receivedData":{"temperature":120.6, "humidity":74.6, "ph":8.3, "turbidity":74.3, "deviceId":"ESP32-1", "serverTimestamp":16292239.880}}
10121:179.759 -> Server response: {"status":"success", "receivedData":{"temperature":120.6, "humidity":77.7, "ph":6.7, "turbidity":50.7, "deviceId":"ESP32-1", "serverTimestamp":16292239.880}}
10121:228.880 -> ERROR RESPONSE:
10121:228.880 -> Status Code = 411
10121:239.880 -> Bean Error Default read Timeout
10121:232.434 -> Server response: {"status":"success", "receivedData":{"temperature":132.7, "humidity":66.1, "ph":9.4, "turbidity":51.6, "deviceId":"ESP32-1", "serverTimestamp":16292239.880}}
10121:303.790 -> Server response: {"status":"success", "receivedData":{"temperature":130.0, "humidity":77.0, "ph":8.1, "turbidity":68.9, "deviceId":"ESP32-1", "serverTimestamp":16292239.880}}
10121:303.850 -> Server response: {"status":"success", "receivedData":{"temperature":126.6, "humidity":120.6, "ph":7.6, "turbidity":101.6, "deviceId":"ESP32-1", "serverTimestamp":16292239.880}}
10121:303.944 -> Server response: {"status":"success", "receivedData":{"temperature":124.7, "humidity":130.7, "ph":8.9, "turbidity":67.4, "deviceId":"ESP32-1", "serverTimestamp":16292239.880}}
10121:326.124 -> Server response: {"status":"success", "receivedData":{"temperature":131.5, "humidity":109.7, "ph":8.9, "turbidity":68.1, "deviceId":"ESP32-1", "serverTimestamp":16292239.880}}
10121:336.817 -> Server response: {"status":"success", "receivedData":{"temperature":129.3, "humidity":136.6, "ph":13.3, "turbidity":68.7, "deviceId":"ESP32-1", "serverTimestamp":16292239.880}}
10121:340.150 -> Server response: {"status":"success", "receivedData":{"temperature":124.6, "humidity":120.6, "ph":7.6, "turbidity":101.6, "deviceId":"ESP32-1", "serverTimestamp":16292239.880}}
10121:337.655 -> Server response: {"status":"success", "receivedData":{"temperature":125.1, "humidity":166.1, "ph":9.6, "turbidity":148.7, "deviceId":"ESP32-1", "serverTimestamp":16292239.880}}
10121:338.689 -> Server response: {"status":"success", "receivedData":{"temperature":120.2, "humidity":73.6, "ph":12.8, "turbidity":38.6, "deviceId":"ESP32-1", "serverTimestamp":16292239.880}}
10121:339.144 -> Server response: {"status":"success", "receivedData":{"temperature":129.1, "humidity":107.7, "ph":9.6, "turbidity":140.2, "deviceId":"ESP32-1", "serverTimestamp":16292239.880}}
10121:449.139 -> Server response: {"status":"success", "receivedData":{"temperature":130.7, "humidity":78.4, "ph":6.6, "turbidity":146.7, "deviceId":"ESP32-1", "serverTimestamp":16292239.880}}

```

The server then responds to the sensor data sent by the ESP32, as depicted in Figure 7.

```

Received sensor data at: {
  temperature: 30,
  humidity: 74.7,
  ph: 9.5,
  turbidity: 32.2,
  deviceId: 'ESP32-1',
  serverTimestamp: '17/05/2025 10:21:52:948'
}
Received sensor data at: {
  temperature: 28.6,
  humidity: 39.4,
  ph: 12.4,
  turbidity: 53.7,
  deviceId: 'ESP32-1',
  serverTimestamp: '17/05/2025 10:21:54:374'
}
Received sensor data at: {
  temperature: 28.2,
  humidity: 71.8,
  ph: 11.6,
  turbidity: 61.1,
  deviceId: 'ESP32-1',
  serverTimestamp: '17/05/2025 10:21:54:740'
}
Received sensor data at: {
  temperature: 26.8,
  humidity: 36,
  ph: 6.5,
  turbidity: 75.9,
  deviceId: 'ESP32-1',
  serverTimestamp: '17/05/2025 10:21:58:395'
}
Received sensor data at: {
  temperature: 22.6,
  humidity: 52.8,

```

Figure 7. Server response when the ESP32 sends sensor data

B. Security Testing

In the first security test scenario, the ESP32 sends an invalid token to the server. This causes the server to perform authentication and return an error message to the ESP32, as shown in Figure 8.

```

09:48:39.149 -> Error code: 403
09:48:44.733 -> Error code: 403
09:48:50.444 -> Error code: 403
09:49:09.245 -> ..Connected to WiFi
09:49:18.610 -> Login failed
09:49:18.610 -> Connection problem. Reconnecting...
09:49:19.232 -> ...Connected to WiFi
09:49:27.194 -> Login failed
09:49:27.194 -> Connection problem. Reconnecting...
09:49:27.791 -> ....Connected to WiFi
09:49:31.775 -> Token received: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImFkbWw1Iiwicm9zeSI6ImFkbWw1IiwiaWF0IjoxNzQ3MzU0MjEwLjE3CjE4a10jE3NDcONTM3MzF9.m2Grj
09:49:32.316 ->
09:49:32.316 -> ERROR RESPONSE:
09:49:32.316 -> Status Code: 403
09:49:32.316 -> Pesan Server: Token tidak valid atau kadaluarsa
09:49:32.316 -> Error Code: INVALID_TOKEN
09:49:37.748 ->
09:49:37.748 -> ERROR RESPONSE:
09:49:37.748 -> Status Code: 403
09:49:37.748 -> Pesan Server: Token tidak valid atau kadaluarsa
09:49:37.748 -> Error Code: INVALID_TOKEN
09:49:43.230 ->
09:49:43.230 -> ERROR RESPONSE:
09:49:43.230 -> Status Code: 403
09:49:43.230 -> Pesan Server: Token tidak valid atau kadaluarsa
09:49:43.230 -> Error Code: INVALID_TOKEN

```

Figure 8. Server response to the ESP32

On the server side, after verifying the invalid token sent by the ESP32, the server displays an error message stating “Token is invalid or expired”, as illustrated in Figure 9.

```
[nodemon] starting node index.js
[nodemon] restarting due to changes...
Server running on port 3000
[nodemon] starting node index.js
Server running on port 3000
Error: {
  errorCode: 'INVALID_TOKEN',
  message: 'Token tidak valid atau kadaluarsa'
}
Error: {
  errorCode: 'INVALID_TOKEN',
  message: 'Token tidak valid atau kadaluarsa'
}
Error: {
  errorCode: 'INVALID_TOKEN',
  message: 'Token tidak valid atau kadaluarsa'
}
Error: {
  errorCode: 'INVALID_TOKEN',
  message: 'Token tidak valid atau kadaluarsa'
}
Error: {
  errorCode: 'INVALID_TOKEN',
  message: 'Token tidak valid atau kadaluarsa'
}
Error: {
  errorCode: 'INVALID_TOKEN',
  message: 'Token tidak valid atau kadaluarsa'
}
Error: {
  errorCode: 'INVALID_TOKEN',
  message: 'Token tidak valid atau kadaluarsa'
}
Error: {
  errorCode: 'INVALID_TOKEN',
  message: 'Token tidak valid atau kadaluarsa'
}
Error: {
  errorCode: 'INVALID_TOKEN',
  message: 'Token tidak valid atau kadaluarsa'
}
Error: {
  errorCode: 'INVALID_TOKEN',
  message: 'Token tidak valid atau kadaluarsa'
}
Error: {
  errorCode: 'INVALID_TOKEN',
  message: 'Token tidak valid atau kadaluarsa'
}
```

Figure 9. Server response for an invalid token

The next test was carried out by creating an incorrect handshake between the ESP32 and the server. The result of this test on the ESP32 side is shown in Figure 10.

```
13:42:02.461 -> ...Connected to WiFi
13:42:10.453 -> Login failed
13:42:10.453 -> Connection problem. Reconnecting...
13:42:10.986 -> ...Connected to WiFi
13:42:19.464 -> Login failed
13:42:19.464 -> Connection problem. Reconnecting...
13:42:20.140 -> ...Connected to WiFi
13:42:28.199 -> Login failed
13:42:28.199 -> Connection problem. Reconnecting...
13:42:28.808 -> ...Connected to WiFi
13:42:36.879 -> Login failed
13:42:36.879 -> Connection problem. Reconnecting...
13:42:37.429 -> ...Connected to WiFi
13:42:45.325 -> Login failed
13:42:45.325 -> Connection problem. Reconnecting...
13:42:45.941 -> ...Connected to WiFi
13:42:54.103 -> Login failed
13:42:54.103 -> Connection problem. Reconnecting...
13:42:54.635 -> ...Connected to WiFi
13:43:02.507 -> Login failed
13:43:02.507 -> Connection problem. Reconnecting...
13:43:03.190 -> ...Connected to WiFi
13:43:11.153 -> Login failed
13:43:11.153 -> Connection problem. Reconnecting...
13:43:11.698 -> ...Connected to WiFi
13:43:46.177 -> Login failed
13:43:46.177 -> Connection problem. Reconnecting...
```

Figure 10. ESP32 cannot find the server

On the server side, the system attempts to perform a proper handshake with the ESP32 but is unable to detect the device, as shown in Figure 11.

```

$ npm run dev
> arduinojwt@1.0.0 dev
> nodemon index.js

[nodemon] 3.1.10
[nodemon] to restart at any time, enter 'rs'
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting 'node index.js'
Server running on port 3000
Error: {
  errorCode: 'INVALID_CREDENTIALS',
  message: 'Username atau password salah'
}
Error: {
  errorCode: 'INVALID_CREDENTIALS',
  message: 'Username atau password salah'
}
Error: {
  errorCode: 'INVALID_CREDENTIALS',
  message: 'Username atau password salah'
}
Error: {
  errorCode: 'INVALID_CREDENTIALS',
  message: 'Username atau password salah'
}
Error: {
  errorCode: 'INVALID_CREDENTIALS',
  message: 'Username atau password salah'
}
Error: {
  errorCode: 'INVALID_CREDENTIALS',
  message: 'Username atau password salah'
}

```

Figure 11. Server cannot detect the ESP32

The final security test involves sending a valid token, where the server successfully receives the sensor data and prepares it to be stored in the database, as depicted in Figure 12.

```

Timestamp: 2025-05-17T02:28:50.826Z
}
Received sensor data: {
  temperature: 24.8,
  humidity: 43,
  ph: 5.1,
  turbidity: 52.7,
  deviceId: 'ESP32-1',
  Timestamp: 2025-05-17T02:28:56.045Z
}
Received sensor data: {
  temperature: 30.9,
  humidity: 64.7,
  ph: 8.2,
  turbidity: 47.2,
  deviceId: 'ESP32-1',
  Timestamp: 2025-05-17T02:29:01.615Z
}
Received sensor data: {
  temperature: 34.9,
  humidity: 52.2,
  ph: 11.3,
  turbidity: 46.4,
  deviceId: 'ESP32-1',
  Timestamp: 2025-05-17T02:29:07.414Z
}
Received sensor data: {
  temperature: 24.4,
  humidity: 65.6,
  ph: 10.4,
  turbidity: 77.1,
  deviceId: 'ESP32-1',
  Timestamp: 2025-05-17T02:29:13.117Z
}
Received sensor data: {
  temperature: 20.9,
  humidity: 48.8,
  ph: 8.7,
  turbidity: 64.2,
  deviceId: 'ESP32-1',
  Timestamp: 2025-05-17T02:29:18.468Z
}
Received sensor data: {
  temperature: 21.9,
  humidity: 37.4,
  ph: 6.5,
  turbidity: 47,
  deviceId: 'ESP32-1',
  Timestamp: 2025-05-17T02:29:23.639Z
}

```

Figure 12. Server response displaying sensor data

C. Performance Testing

Performance testing was conducted to measure the efficiency of the server response time in two scenarios: without JWT and with JWT. The first step was to define the test parameters, where the response time was measured from the moment the ESP32 sends a request until it receives a response from the server. The ESP32 was programmed to record time using the built-in `millis()` function, and the time difference was used as the performance indicator.

In the first scenario (without JWT), the ESP32 sends a request to the server without authentication, and the server directly processes the request and returns a response. In the second scenario (with JWT), the ESP32 first sends its credentials to obtain a token from the server and then uses this token to access the service. The server verifies the credentials, generates and signs a JWT token, and sends it back to the ESP32.

The time differences in both scenarios were analyzed to compare the processing load on the server. The results show that the use of JWT increases the response time because of the additional processes required for authentication and token creation. The comparison results are presented in Figure 13.

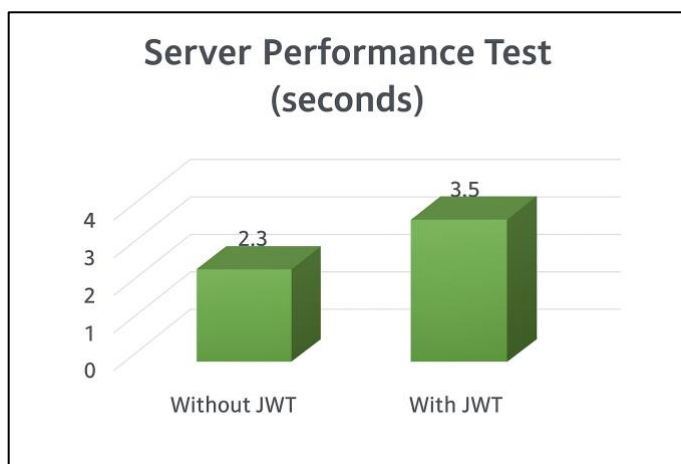


Figure 13. Server performance test graph

The use of JSON Web Token (JWT) causes the server response time to be longer than in the case without JWT because of the additional steps required to ensure data security and validity. In the JWT scenario, the server not only processes the request from the ESP32 but must also authenticate the credentials sent, construct the token payload, sign it using a cryptographic algorithm, and send the token back to the ESP32. The creation and signing of the token require additional computation time, particularly due to the hashing or encryption operations involved.

Although the use of JWT introduces additional processing time due to token generation and validation, the observed overhead remains minimal and acceptable for real-time IoT monitoring, indicating a favorable trade-off between security and performance.

In addition, the server may perform internal verification before sending the token to the client to ensure that the token structure and contents comply with the predefined rules. All of these processes increase the complexity of server-side processing, thereby extending the time needed to provide a response to the ESP32.

Nevertheless, this increase in response time is acceptable because JWT offers significant benefits in terms of security and data integrity in the communication between the server and IoT devices.

4. CONCLUSION

Based on the implementation and testing that have been carried out, the data transmission security system for IoT using JSON Web Token (JWT) on the ESP32 has been proven to enhance the security of data communication between the device and the server. JWT enables device authentication without repeatedly sending credentials and is able to detect invalid or expired tokens. Testing in three aspects—functionality, security, and performance—shows that the system operates correctly, remains responsive, and adds only a small overhead to the response time. This indicates that JWT can be effectively applied in IoT environments, even on devices with limited memory and computing resources such as the ESP32. The system can be further developed for integration with cloud platforms or applied in various IoT scenarios that require lightweight and secure authentication. This study demonstrates that the use of JSON Web Token (JWT) on the ESP32 can improve the security of IoT data transmission without introducing significant performance overhead. For future work, it is recommended to integrate the system with lightweight communication protocols such as MQTT in order to increase efficiency. In addition, end-to-end encryption and more advanced token management mechanisms, such as refresh tokens, can be implemented to further strengthen data protection and provide more flexible authentication. Large-scale testing is also highly recommended to evaluate the stability and scalability of the system under more complex IoT network conditions.

ACKNOWLEDGEMENT

The authors would like to express sincere gratitude to the Ministry of Education, Culture, Research, and Technology of Indonesia for the financial support provided through the Fundamental Research Grant program, which made this research possible.

REFERENCES

- [1] MrFMach, “M2M project between Arduino and ESP32 using Node-RED flows,” 2019. [Online]. Available: <https://github.com/MrFMach/M2M-Arduino-ESP32-NodeRED>
- [2] S. R. Department, “Number of IoT connected devices worldwide 2019–2033,” 2024. [Online]. Available: <https://www.statista.com>
- [3] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, “Internet of things: Vision, applications and research challenges,” *Ad Hoc Networks*, vol. 10, no. 7, pp. 1497–1516, 2012.
- [4] K. Ashton, “That ‘Internet of Things’ Thing,” 2009.
- [5] S. Sicari, A. Rizzardi, L. Grieco, and A. Coen-Porisini, “Security, privacy and trust in Internet of Things: The road ahead,” *Comput. Networks*, vol. 76, pp. 146–164, 2015.
- [6] H. Ning and H. Liu, “Cyber-physical-social based security architecture for future IoT,” *IEEE Internet Things J.*, vol. 1, no. 5, pp. 430–444, 2014.
- [7] A. Banks and R. Gupta, “MQTT Version 3.1.1,” 2014.
- [8] A. Banks, E. Briggs, K. Borgendale, and R. Gupta, “MQTT Version 5.0,” 2019.
<https://csridjournal.potensi-utama.org/index.php/CSRIDjournal/index> | redaksijurnalupu@gmail.com

- [9] ESP32io, “ESP32 - MQTT Tutorial,” 2025. [Online]. Available: <https://esp32io.com/tutorials/esp32-mqtt>
- [10] W. Stallings, *Cryptography and Network Security: Principles and Practice*. Pearson, 2017.
- [11] W. G. Halfond, J. Viegas, and A. Orso, “A classification of SQL-injection attacks and countermeasures,” in *IEEE International Symposium on Secure Software Engineering*, 2006.
- [12] M. Jones, J. Bradley, and N. Sakimura, “JSON Web Token (JWT),” 2015.
- [13] Auth0, “Introduction to JSON Web Tokens,” 2024. [Online]. Available: <https://auth0.com/docs>
- [14] M. Jones and D. Hardt, “The OAuth 2.0 Authorization Framework: Bearer Token Usage,” 2012.
- [15] J. F. R. Herrera, “Secure IoT communications using JWT authentication,” *IEEE Access*, vol. 9, pp. 12345–12356, 2021.
- [16] S. Author, “IoT Based Real-Time Prototype Design for Smart Aquaculture Ecosystem,” in *IEEE Conference*, 2025. [Online]. Available: <https://ieeexplore.ieee.org/document/10696610>
- [17] M. Dwiyaniti, R. Novita, and Tohazen, “Development of Water Quality Monitoring Systems in Super Intensive Aquaculture System using ESP32 and Blynk,” in *SCITEPRESS*, 2019. [Online]. Available: <https://www.scitepress.org/Papers/2019/98942/98942.pdf>
- [18] C. P. S. Kumar, “IoT-based shrimp pond monitoring using ESP32,” *Int. J. Eng. Res.*, 2024.
- [19] A. Kurniawan, “IoT-based environmental monitoring for oyster mushroom cultivation using ESP32,” *J. Agric. Technol.*, 2023.
- [20] R. Novita, “Hydroponic plant monitoring system using ESP32 and IoT sensors,” *Indones. J. Electr. Eng. Comput. Sci.*, 2022.
- [21] M. O. Pradeep, “IoT-based fish farming monitoring system using NodeMCU ESP32,” *GIJET*, 2024.
- [22] D. S. Rani, “IoT-based mangrove crab farming monitoring system,” *Int. J. Aquac. Technol.*, 2023.
- [23] M. Flores-Iwasaki, G. A. Guadalupe, M. Pachas-Caycho, S. Chapa-Gonza, R. C. Mori-Zabarburú, and J. C. Guerrero-Abad, “Internet of Things (IoT) Sensors for Water Quality Monitoring in Aquaculture Systems: A Systematic Review and Bibliometric Analysis,” Mar. 01, 2025, *Multidisciplinary Digital Publishing Institute (MDPI)*. doi: 10.3390/agriengineering7030078.
- [24] J. Li, “Smart aquaculture system based on IoT and cloud computing,” *IEEE Access*, vol. 8, pp. 123456–123467, 2020.
- [25] T. Miller *et al.*, “Integrating Artificial Intelligence Agents with the Internet of Things for Enhanced Environmental Monitoring: Applications in Water Quality and Climate Data,” Feb. 01, 2025, *Multidisciplinary Digital Publishing Institute (MDPI)*. doi:

10.3390/electronics14040696.

- [26] S. Tandel, B. Tandel, H. V Parmar, Y. Solanki, and T. Tandel, "Smart Aquaculture: IOT AND AI Application for Sustainable Fisheries," vol. 5, no. 2, pp. 6319–6327, 2025, doi: 10.5281/ScienceWorld.14946381.