Website-based Battery Capacity and Position Monitoring System for Automated Guided Vehicle (AGVs)

Adriand Pratama[1], Rifqi Amalya Fatekha[2], Eko Rudiawan Jamzuri[3], and Anugerah Wibisana[4*]

[1] Department of Electrical Engineering, Robotics Engineering Technology Study Program, Batam State Polytechnic, Batam, Indonesia

[*]Email: adriandpratama678@gmail.com

Received on dd-mm-yyyy | Revised on dd-mm-yyyy | Accepted on dd-mm-yyyy

***Abstract*— This article discusses research related to a website-based Automated Guided Vehicle (AGV) monitoring system using the Laravel framework, focusing on monitoring the battery and position of AGVs represented using MFRC522 RFID sensors in real time. Data is obtained from the ESP32 microcontroller, which reads the MFRC522 RFID sensor and PZEM-017 battery sensor, then sent to the Laravel server via the HTTP POST method in JSON format. The data received by the system is stored and processed through the Laravel API, then stored in a MySQL database before being displayed on the dashboard page. The system uses an IP and RFID mapping mechanism to identify the AGV's identity and determine its position on the track. The system was tested under various conditions, such as sending data in the wrong format and using unregistered IPs. The results show that the system can handle these conditions and display the data on the website dashboard. The performance analysis system shows that the recommended monitoring system operates safely under standard operating conditions, achieving a 100% data transmission success rate with consistent server response, as indicated by HTTP status code 200. An asynchronous communication mechanism allows the dashboard interface to update monitoring data at 10 second intervals. Furthermore, the system uses a mechanism of IP and RFID mapping, which enables precise identification of AGV identity and position. It is also able to handle incomplete or invalid data inputs without interfering with system operation. A structured and scalable web-based AGV monitoring approach that integrates real-time data logging and flexible data mapping using the Laravel 10 framework is presented in this study.**

Keywords: Monitoring, Laravel, API, RFID, Battery, Web Application, Database, IP, MySQL, PHP

I. INTRODUCTION

Robotics and automation engineering in the current era has penetrated the industrial world with technology that is advancing day by day. Automated Guided Vehicles (AGVs) are one example of industrial robots that are unique from others. AGV robots are mobile robots that can be programmed and have integrated sensors that can automatically sense and move along a planned route[1][6]. In the industrial world, AGVs are used for more efficient transportation of goods and materials and to reduce human error[2]. Behind the usefulness of AGVs lie several challenges, one of which is the condition of the battery and its position in real time[1]. If this challenge is not addressed, it will cause errors in AGV operation and become unconducive.

The monitoring of Automated Guided Vehicles (AGVs) utilizing a variety of methods and technologies has been the subject of several research. Using encoder sensors and XBee-based wireless communication, Satriyo et al. created an AGV position monitoring system with MATLAB as the monitoring interface [3]. The system relied on desktop software and did not incorporate a web-based monitoring platform with centralized data storage, despite its ability to display AGV movements in real time. Furthermore, identity mapping and battery monitoring for many AGVs running concurrently were not addressed by the system.

The MQTT protocol is used to connect the microcontroller and web server to the Internet of Things (IoT)-based building security monitoring system. The system has login, database storage, and security notification features [4]. The performance analysis of REST APIs in real-time IoT-based vehicle monitoring systems was the subject of other research, such as the study by Dwiyanto et al. [5]. The integration of AGV position identification mechanisms, such as RFID-based location mapping, and how the system manages incomplete, invalid, or unmapped data in real-world deployment scenarios were not specifically addressed in this study, despite the fact that it offered insightful information about API performance and data transmission efficiency.

In contrast to the study by Satriyo et al. [3], which focused solely on AGV position monitoring using encoder sensors and desktop-based software, this research proposes a web-based monitoring system with centralized data storage and real-time accessibility. Moreover, this study integrates battery monitoring and identity mapping mechanisms that were not

addressed in [3].

Compared to the work by Dwiyanto et al. [5], which primarily analyzed REST API performance in IoT-based monitoring systems, this research emphasizes the practical implementation of RFID-based position identification and IP address mapping for AGVs. In addition, this study explicitly evaluates system behavior when handling incomplete, invalid, or unmapped data, which was not discussed in [5].

The lack of a web-based AGV monitoring system that combines real-time battery monitoring, RFID position identification, and adaptable data mapping techniques within a single framework can be highlighted as a research need based on the review of previous works. Specifically, resolving data inconsistencies such as missing sensor data or unregistered IP addresses which are prevalent in actual industrial settings has received little attention. In order to offer dependable and scalable AGV monitoring, this research proposes a Laravel-based system that integrates real-time data collecting, IP and RFID mapping, and structured database logging.

This study focuses on the use of a monitoring website that allows operators to view the status of AGVs, including battery status and AGV position, in real time. This study centers on the creation of a website-based monitoring system using the Laravel framework[7]. AGV data is sent via a communication protocol and stored in a database, after which the data is displayed on the website's main page. At this implementation stage, the system was tested using a battery and RFID MFRC522 to simulate the location and battery of the AGV. Therefore, the purpose of this study was to design a Laravel-based website monitoring system that could display data from the AGV's battery and location in real time, thereby facilitating the operator's monitoring of the AGV in an industrial environment.

Consequently, the following research questions are the focus of this study:

(1) How can real-time AGV battery position and status be displayed in a Laravel-based web monitoring system?

(2) How can RFID and IP address mapping techniques be used to precisely determine the location and identification of AGVs?

(3) In a real-time AGV monitoring situation, how can the system manage incomplete, erroneous, or unmapped data?

The goal of this study is to develop and deploy a Laravel-based AGV monitoring system that can show real-time battery and position data via a web interface based on these research topics. The development of an IP and RFID mapping mechanism and an assessment of the system's capacity to manage data discrepancies in real-world deployment scenarios are further goals of this research.

The contributions of this research include the development of a Laravel-based web monitoring system for real-time AGV battery and position visualization, the implementation of IP and RFID-based identity and location mapping mechanisms, and an evaluation of system robustness in handling data inconsistencies commonly encountered in real-world monitoring environments.

II. METHOD

A. System Design

In this study, the system being developed aims to monitor the battery status and position of AGVs in real time using a website and the Laravel framework. At this stage, RFID MFRC522 and lead-acid batteries are used as simulations for this study. The system consists of an ESP32 device as a data transmitter, a Laravel web server and API for data communication, and a user interface (website dashboard).

The data transmission flow is as follows: the first ESP32 reads the RFID tag and sends it to the second ESP32 using Wi-Fi. The second ESP32 acts as the RFID tag data receiver and reads the battery data, then sends the RFID tag data and battery data to the server using the HTTP POST protocol. The Laravel web server then receives the data through the API and stores it in the MYSQL database.

In addition, the system is equipped with a user login feature consisting of two roles, namely admin and user. This feature aims to restrict user access according to their respective responsibilities.

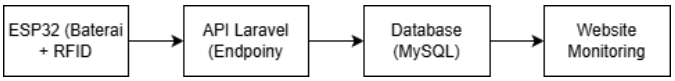

Figure 1. Block diagram of a Laravel-based website monitoring system.

The overall system architecture is illustrated in Figure 2, which shows the interaction between hardware components, communication protocols, and the web-based monitoring system. The architecture consists of two ESP32 devices, RFID and battery sensors, a RESTful API, a Laravel-based web server, and a MySQL database.

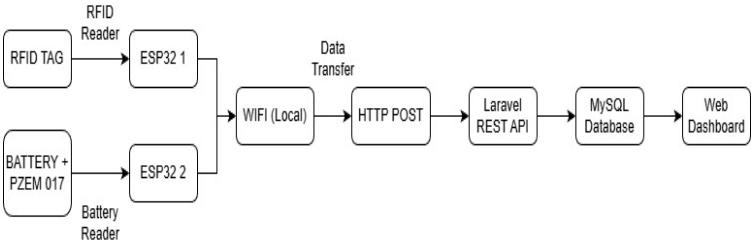

Figure 2. architecture diagram

The use of two ESP32 devices is intended to separate data acquisition tasks and reduce processing load on a single microcontroller. The first ESP32 is dedicated to reading RFID tag data for AGV position identification, while the second ESP32 functions as a data aggregator that receives RFID data and simultaneously reads battery parameters before transmitting the combined data to the server. This separation improves system modularity and reliability.

To further clarify the data transmission process, a sequence diagram is provided in Figure 3. The diagram illustrates the sequential flow of data starting from RFID and battery sensors, transmission between ESP32 devices, data delivery to the Laravel API, database storage, and visualization on the dashboard interface.

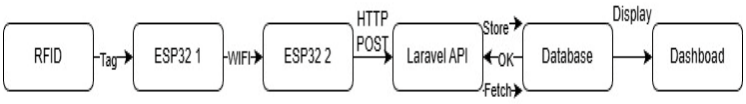

Figure 3. Sequence Diagram

B. Hardware

The system is tested using hardware such as:

1) The ESP32 functions as a microcontroller that can read sensor data from battery capacity and data from the MFRC522 RFID tag[8][9].
2) RFID MFRC522 is used to simulate the location of AGVs, with each RFID tag representing the position of an AGV.
3) Lead-acid batteries used as batteries for AGVs.
4) The PZEM 017 sensor reads voltage, current, and power data from the battery [10]
5) RS485 as a communication tool between the sensor and the ESP32 microcontroller[11].

The hardware connections between the ESP32, RFID MFRC522, PZEM-017 sensor, RS485 module, and battery are illustrated in a wiring diagram shown in Figure 4. This diagram provides a clear overview of the electrical connections used during system implementation.

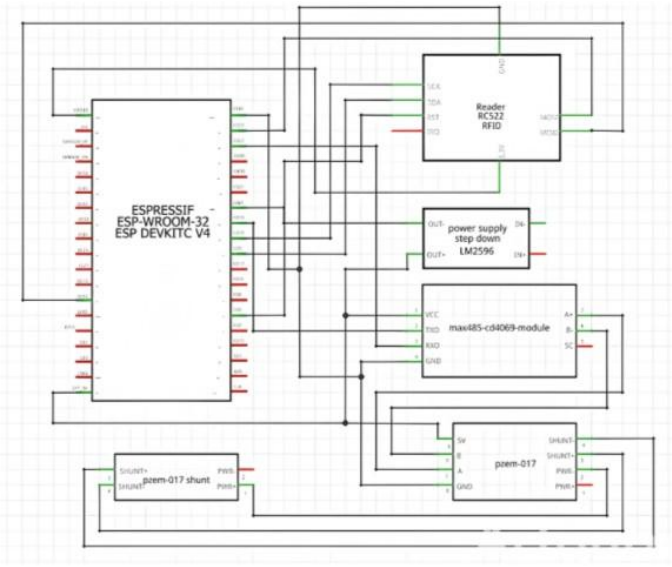

Figure 4. Wiring diagram

C. Software

The website monitoring stands using the Laravel framework that manages the API, database, and user interface. The related software components are:

1) Laravel API that functions as a communication protocol between components and the server to receive all data from the ESP32 microcontroller sent using the HTTP POST method[12].
2) MYSQL is a database server program that receives and sends data and then stores all related data[13].
3) Dashboard interface that displays battery data and AGV position.
4) Arduino IDE is software used to program the ESP32 to send data[14].

The software environment used in this study consists of Laravel framework version 9.52.16, PHP version 8.2.12, and MySQL database version 10.4.32. The ESP32 microcontroller is programmed using Arduino IDE version 2.3.7. These software versions were selected to ensure compatibility, stability, and ease of integration between the hardware and web-based monitoring system.

D. Desain Database

TABLE I
DATABASE STRUCTURE

Table Name	Field	Description
battery	id, name, ip, battery, created_at, updated_at	Storing AGV battery capacity data
data_logs	id, name, ip, location, tag, created_at, updated_at	Storing overall AGV data
ip_mappings	id, ip, robot_name, created_at, updated_at	Storing and setting specific IP data for specific AGVs
rfid_mappings	id, tag, location_label, created_at, updated_at	Storing and setting specific RFID data for specific points of AGV positions
robots	id, name, ip, location, created_at, updated_at, tag	Storing RFID tag data for AGV positions
users	id, name, email, profile_picture, email_verified_at, password, remember_token, created_at, updated_at, role	Storing logged-in user data

One of the important components in a system is a database. A database functions to store and manage all data. In this system, the database functions to store all related data, such as battery data and RFID data, namely AGV position data sent from ESP32. The database structure was designed using MYSQL with six main tables, namely battery, data_logs, ip_mappings, rfid_mappings, robots, and users. The database implementation is carried out through PhpMyAdmin, a web-based tool designed to manage MySQL administration through a graphical interface[15].

The database structure is further described using an Entity Relationship Diagram (ERD), as shown in Figure 5. The ERD

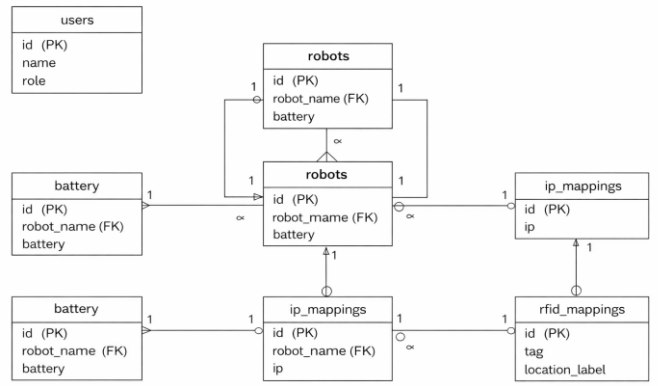

Figure 5. Entity Relationship Diagram

illustrates the relationships between the battery, robots, RFID mappings, IP mappings, data_logs, and users tables, ensuring data consistency and structured logging of AGV monitoring information.

III. RESULTS AND DISCUSSION

A. *System Testing*

1) *RFID Tag Does Not Match Hexadecimal Format*

RFID tag data in numerical format (numbers) can be stored in the rfid_mappings table via the website and can be assigned to a specific location point, namely G. After the mapping process, the AGV location data will appear on the dashboard page.

These results indicate that the system does not perform strict validation of RFID tags, but rather treats RFID tags as general strings. This condition allows for freedom in managing RFID tags, but will cause inconsistencies in the system if the input data does not comply with RFID standards. Therefore, future development requires UID format validation to ensure the uniformity of stored RFID tag data. RFID tag does not match hexadecimal format as shown in Figure 6.

This occurs because the Laravel system we built does not enforce strict rules regarding RFID UID format validation, so the system continues to accept any data in string form. As a result, RFID UID data that does not match the format can be displayed.

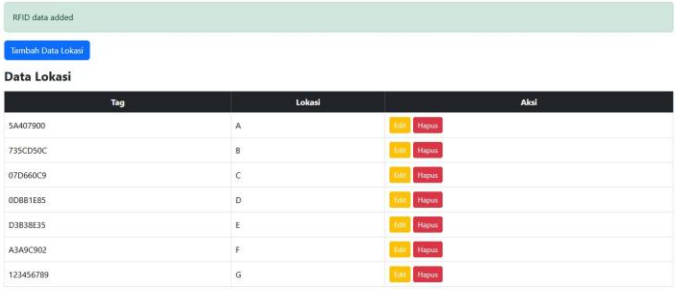

Figure 6. Incorrect RFID format

2) *RFID Data Transmission Without Battery Power*

In this test, data transmission was performed only on RFID data without battery data. The results showed that RFID data was still sent to the database and the AGV positions could still be displayed on the dashboard, but the battery data was displayed as a strip (-). Figure 7 shows RFID data transmission without battery power. This happens because the system we built prioritizes RFID location data as the primary identity of the AGV.

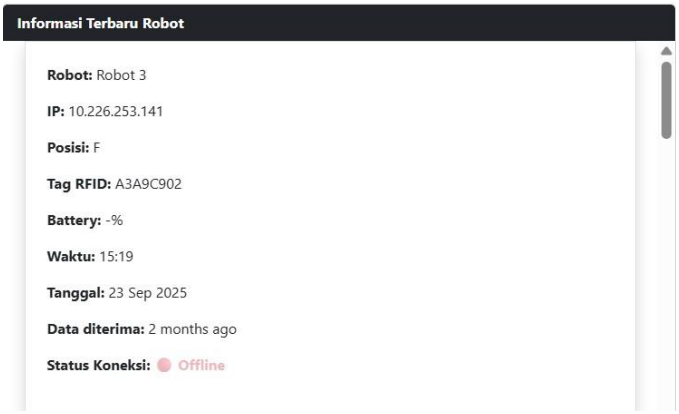

Figure 7. RFID data transmission without battery data

3) *Battery Data Transmission Without RFID Data*

This test was conducted by sending battery data without any RFID data. The result was that the battery data was still entered into the database, but the battery data did not appear on the dashboard page because the system requires RFID data as the AGV position identifier to be displayed on the dashboard page. This shows that only complete AGV data can be displayed on the dashboard page. Figure 8 shows battery data transmission without RFID data. We built this system because we set the battery data to be sent from the sensor to the system every 10 seconds, so if the battery data without RFID is still displayed on the dashboard, the battery data will fill up the dashboard.

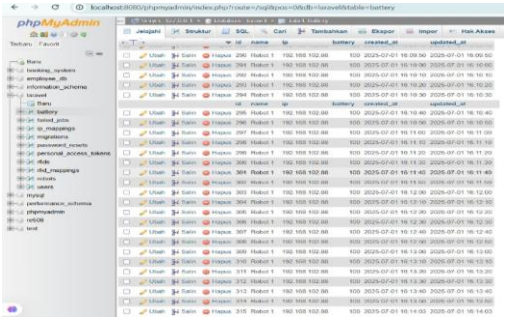

Figure 8. Battery database

4) *Difference between the recipient's and sender's IP addresses*

The IP mapped on the settings page was not the same as the ESP32 sender IP used for the test. This discrepancy is depicted in Figure 9. The robot name was shown as "unknown," yet the

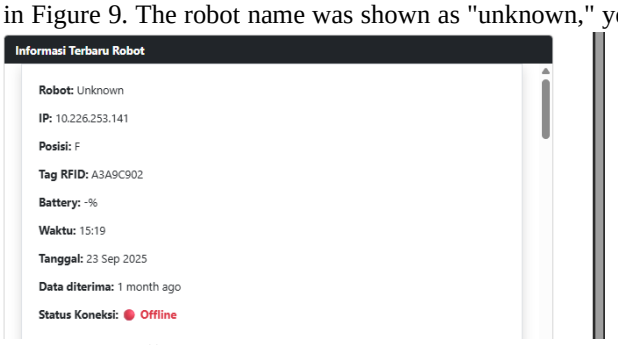

Figure 9. The difference between the recipient's and sender's IP addresses

data was still saved and shown on the dashboard. In order to reduce data loss and signal mapping inconsistencies during AGV monitoring operations in real deployment scenarios to preserve continuous system functionality overall, the system allows unmapped IP data and assigns a default identity rather than blocking it.

5) Manual Data Addition to Datalog with Incorrect IP Format

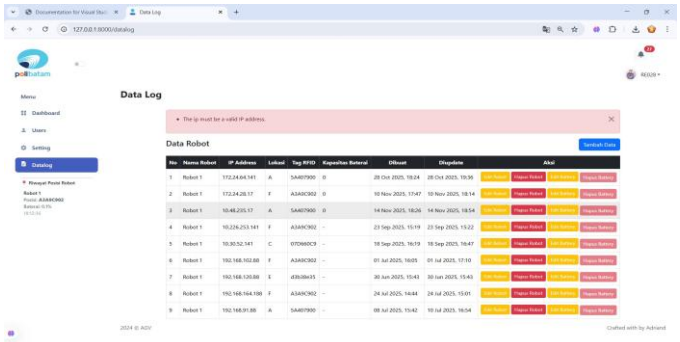

Figure 10. Incorrect IP format

This test was conducted by adding manual data to the data log page using an IP format that did not comply with the IPv4 standard. As a result, the data could not be added to the database or the website, and the website displayed an error message. Figure 10 shows manual data addition to datalog with incorrect IP format. This occurs because the system implements strict IP validation according to the format to prevent invalid IP formats from being stored in the database.

6) RFID Data Does Not Match Format and IP Not Yet Mapped

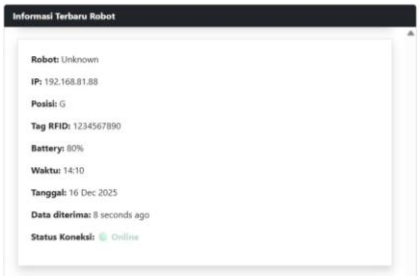

Figure 11. RFID does not match the format and IP is not registered

Data was manually added to the data log page in order to perform this test. The dashboard exhibited unmapped IPs and RFID tags that did not match UID or hexadecimal format, but the robot identification was shown as "Unknown," according to the results. This happens because the mapping method needs proper RFID and IP references to determine robot identity, yet the system allows data logging with incomplete identification. This condition is seen in Figure 11.

7) Normal operation system

Figure 12 shows normal operation system. The ESP32 successfully sent legitimate RFID and battery data to the server during regular operation testing. The Laravel API processed the data correctly, stored it in the database, and showed it on the dashboard without any delays or data loss.

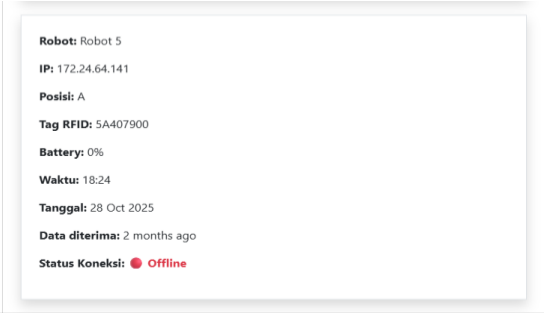

Figure 12. Normal operating system

8) Load and Stress Testing

Under typical working settings, load testing was carried out by continually transmitting data from the ESP32 to the server at regular intervals. The dashboard showed real-time changes without any data loss, and the system successfully processed and stored incoming data.

By raising the data transmission frequency over typical bounds, load testing was carried out. The database server started to exhibit changes at increasing demand levels, such as longer data loading times, which indicated server-side processing limits. Nevertheless, there was no data damage or system failure during testing, and the system kept running.

B. Software Implementation

This website-based monitoring system is built on two things, namely ESP32 microcontroller programming using Arduino IDE and Laravel-based website monitoring development. These two components are interconnected via a WIFI network to perform real-time data transmission and reception.

1) ESP32 Microcontroller Integration

The ESP32 microcontroller functions as a component for sending data from sensors to the Laravel server. The basic program of the ESP32 uses several libraries such as WiFi.h, HTTPClient.h, MFRC522.h, and the Modbus library for communication with the PZEM-017 sensor. The ESP32 reads data from the MFRC522 RFID sensor to detect RFID tags that represent the position of the AGV and reads data from the PZEM-017 sensor to measure the battery voltage of the AGV. The sensor reading data is created in JSON format and sent to the Laravel server using the HTTP POST method via the Laravel REST API.

2) Laravel Website Monitoring

This website monitoring is a key component of this system. The website was developed using the Laravel 10 framework. It functions as a monitoring center for AGVs, storing data in a database and displaying it on the dashboard page in real time.

The structure of this website monitoring consists of:

- API Endpoint

  Laravel provides an API Endpoint that receives data from the microcontroller using the HTTP POST method. The data is then stored in the battery table for battery data and the robots table for AGV position RFID tag data.
- MYSQL Database

  The database is used to store all sensor data that enters

the server. The relationship between tables uses foreign keys so that each RFID and battery is connected to a specific AGV.

- Monitoring Dashboard

The monitoring dashboard is a page that can be accessed by users who log in as admins or users. This dashboard page displays information related to AGVs, such as robot name, battery capacity, position, robot IP, data entry time, data entry date, RFID tag, etc., in a box-by-box format. Data is automatically updated using the AJAX method and auto-refreshes every 10 second

- Login Authentication

This system is equipped with an authentication feature that distinguishes between admin and user access rights. Admins have full access rights in the system to manage robot data, other pages in the system, access features to add, edit, and delete data (CRUD), and configure system settings. Users can only view monitoring data on the dashboard page without being able to change system settings.

3) Website Monitoring Display

The website interface is designed with a simple and responsive display using Bootstrap 5 and Laravel's Blade Template Engine to facilitate user operation.

4) Database Structure

The database structure in this website monitoring project uses MySQL and functions as a center
storage for all data sent by the ESP32 microcontroller. The database structure is designed to be flexible, efficient, and support real-time data storage from the ESP32 microcontroller. There are 5 main tables in this project:

a) Battery Table

TABLE II
BATTERY TABLE DATABASE STRUCTURE

Field	Type	Description
id	bigint(20)	Robot identity primary key
name	varchar(255)	Connected robot name.
ip	varchar(255)	Robot IP address.
battery	int(11)	Indicator of stored battery capacity data.
created_at	timestamp	Data transmission time.
updated_at	timestamp	Latest data time.

This table stores all battery monitoring data sent by the microcontroller. Battery data is linked to the robots table via the robot_id foreign key.

b) Robots Table

TABLE III
ROBOTS TABLE DATABASE STRUCTURE

Field	Type	Description
id	bigint(20)	Primary key robot identity
name	varchar(255)	Name of connected robot
ip	varchar(255)	Robot IP address
location	varchar(255)	Contains AGV locations that have not been mapped (still raw tags)
created_at	timestamp	Data transmission time
updated_at	timestamp	Latest data time
tag	varchar(50)	RFID tags sent

This table stores data on all RFID tags sent by the microcontroller to the Laravel server.

c) ip_mappings Table

TABLE IV
IP_MAPPINGS TABLE DATABASE STRUCTURE

Field	Type	Description
id	bigint(20)	Robot identity primary key
ip	varchar(255)	Robot address entered into the database
robot name	varchar(255)	Robot identity after mapping
created_at	timestamp	Data delivery time
updated_at	timestamp	Latest data time

This table is used to store the results of robot identity mapping based on IP. Each incoming IP will be stored in the database, then mapped to the

respective robot identity (IP 192.168.88.88 for Robot 1). The results of the mapping will be stored in the ip_mappings table.

d) users Table

TABLE V
USERS TABLE DATABASE STRUCTURE

Field	Type	Description
id	bigint(20)	Unique user identity
name	varchar(255)	User name
email	varchar(255)	User email
email_verified_at	timestamp	Email registration time
password	varchar(255)	Encrypted password
profile_picture	varchar(255), nullable	Saved profile photo
remember_token	varchar(255), nullable	Token for "remember me" when logging in
created_at	timestamp	Data submission time
updated_at	timestamp	Latest data time
role	varchar(255)	User role status

This table is used to store all user data that has successfully registered an account on the monitoring website. The data stored in this table includes names, email addresses, and other authentication information such as encrypted passwords and authentication tokens. This table also stores user profile photos to make user profiles on the website look more professional. In addition, this table also stores email verification status to support system security.

e) Rfid_mappings Table

This table is used to store data from RFID tag locations that have been mapped. RFID tags that enter the database are then mapped based on specific locations, for example, tag 5A407900 for location A. Location data that has been mapped is then stored in the rfid_mapping table.

TABLE VI
RFID_MAPPINGS TABLE DATABASE STRUCTURE

Field	Type	Description
id	bigint(20)	Robot identity primary key
tag	varchar(255)	RFID tags entered into the database
location_tabel	varchar(255)	Mapped RFID tag locations
created_at	timestamp	Data transmission time
updated_at	timestamp	Latest data time

C. System Test Results

This section displays the test results from monitoring the position and battery capacity of the AGV. The tests were conducted to ensure that every component in the system, from data reading by ESP32, transmission via the WIFI network, data reception via the Laravel API, data storage in the MYSQL database, to data display on the Laravel website page, ran well and in accordance with the research objectives.

The testing was conducted using a lead-acid battery and a PZEM-017 sensor as components for reading battery data, and an MFRC522 RFID sensor as a representation of the AGV's position. The microcontroller used was an ESP32, and the Laravel web application was hosted locally via XAMPP.

a) RFID Tag Reading Test on the First ESP32

TABLE VII
RESULTS OF TAG READING AND TRANSMISSION TO THE SECOND ESP32 TESTING

UID Read	Status Connection to ESP32 B	Delivery status UID
5A407900	Failed	Not sent
5A407900	Successful	Sent
735CD50C	Successful	Sent
07D660C9	Successful	Sent
A3A9C902	Successful	Sent
0DBB1E85	Successful	Sent
D3B38E35	Successful	Sent

The first test was conducted on the first ESP32 used to read RFID tag data using RFID MFRC522. Each tag that was read and stored was represented as a location point of the AGV.

The test results showed that the RFID MFRC522 was able to read the tag and send it to the second ESP32. The RFID tag was read in hexadecimal format and sent via the WIFI network. Figure 13 shows the RFID tag reading test on the first ESP32.

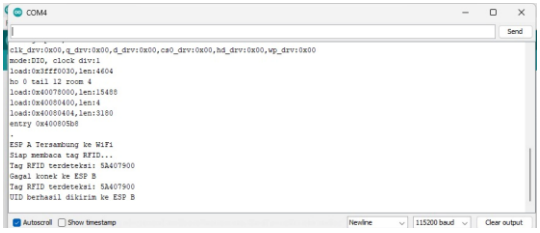

Figure 13. Reading RFID tags on the first ESP32

A brief connectivity problem between the ESP32 devices during data transmission prevented the UID 5A407900 from being processed. The RFID UID data was not sent to the Laravel server because the Wi-Fi connection between the ESP32 in charge of reading the RFID tag and the ESP32 in charge of sending data to the server was not successfully established. Consequently, the associated AGV position data could not be processed and displayed by the system.

Resetting the ESP32 devices and reestablishing their Wi-Fi connection fixed the problem. The RFID UID data could be properly sent to the server and processed by the system once the communication channel had been effectively reestablished. This suggests that dependable AGV position tracking depends on steady inter-device communication.

b) Testing Data Reception, Reading, and Transmission on the Second ESP32

After the RFID tag is successfully read on the first ESP32, the tag data will be sent to the second ESP32, which combines the RFID tag data and battery data before sending it to the Laravel server.

The second ESP32 is connected to a PZEM-017 sensor that uses RS485 serial communication to read battery data. When voltage data is received, the second ESP32 calculates the battery capacity percentage according to a predetermined formula. Figure 13 shows testing data reception, reading, and transmission on the second ESP32.

The data to be sent includes the RFID tag UID, robot ID, robot IP (ESP IP), and battery capacity. The data is sent to the Laravel server in JSON format using the HTTP POST method. Figure 14 shows Testing Data Reception, Reading, and Transmission on the Second ESP32.

Figure 14. Reading battery data, receiving RFID tag data, and sending data to the Laravel server on the second ESP32

TABLE VIII
RESULTS OF READING, RECEIVING, AND SENDING SECOND ESP32 DATA

UID Sent	Data Sent	Server Response
5A407900	RFID UID tag Robot ID Robot IP (ESP IP) Battery Capacity	200
735CD50C	RFID UID tag Robot ID Robot IP (ESP IP) Battery Capacity	200
07D660C9	RFID UID tag Robot ID Robot IP (ESP IP) Battery Capacity	200
A3A9C902	RFID UID tag Robot ID Robot IP (ESP IP) Battery Capacity	-1
0DBB1E85	RFID UID tag Robot ID Robot IP (ESP IP) Battery Capacity	-11

The server response results show that most data transmissions were processed successfully using HTTP response code 200, which indicates successful data storage in the database. However, there are response codes -1 and -11, where -1 means that the ESP32 failed to connect to the server and -11 means that the WiFi connection was lost while sending data to the server. Connection failures to the server can be resolved by ensuring that the API server URL used on the ESP32 matches the server, and WiFi connection failures can be resolved by resetting the ESP32 and the connected WiFi.

c) Data Storage Results in the robots Table

TABLE IX
ROBOTS TABLE DATA

name	ip	tag
Robot 1	192.168.102.88	A3A9C902
Robot 1	192.168.91.88	5A407900
Robot 1	192.168.164.188	A3A9C902
Robot 1	10.30.52.141	07D660C9
Robot 1	10.226.253.141	A3A9C902

The robots table is used to store robot identity data such as the robot ID and name, IP address, and RFID tag successfully sent by the second ESP32. The robots table is one of the main references for the system in mapping RFID tag data to specific robot locations (e.g., tag 5A407900 for location A).

Test results show that RFID tag data can be automatically added directly to the database table after being received by the Laravel server via API. Each RFID tag data uses a robot_id foreign key so that all received data is connected to the correct robot.

The location and tag columns in the robots table in the database will automatically change every time new data enters the database, and the created_at and updated_at timestamps will show that data changes occur in real time. Figure 15 displays data storage results in the robots table

Figure 15. Robots database table in phpMyAdmin

d) Data Storage Results in the battery Table

This table is used to store all battery capacity data sent by the second ESP32 to the Laravel server. Storage in this table uses a real-time and append-only method, meaning that any new data entered into the database will be added to a new row without overwriting the data that has been entered previously. This can store a complete history of battery capacity, facilitating long-term monitoring, battery performance analysis, and detection of capacity decline. Figure 16 displays data storage results in the battery table

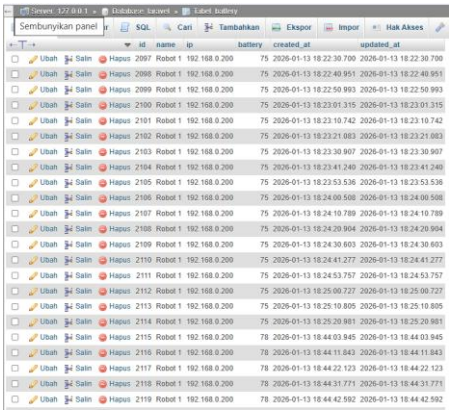

Figure 16. Battery database table in phpMyAdmin

TABLE X
BATTERY TABLE DATA

ip	battery	created_at
192.168.0.200	75	2026-01-13 18:44:11
192.168.0.200	75	2026-01-13 18:44:11
192.168.0.200	78	2026-01-13 18:44:11
192.168.0.200	78	2026-01-13 18:44:11

These results show that the system is able to consistently read the battery every second ESP32 sends data and displays the battery history in a database table based on the time sent automatically. The system is also able to maintain data accuracy because each incoming battery data will create a new row in the database without overwriting the old data that has been entered previously.

e) RFID and IP Mapping Testing

Before data can be displayed in the dashboard menu, the system must perform a mapping process on the RFID and IP to link the robot's identity based on the IP and the robot's location based on the RFID tag. If both data are not mapped, the AGV robot data will not be displayed on the dashboard page because the raw data sent by the microcontroller is only in the form of an IP and RFID tag without any name identity and location point.

TABLE XI
IP_MAPPINGS TABLE DATA

id	ip	robot_name
1	172.24.28.17	Robot 1
3	10.48.235.17	Robot 2

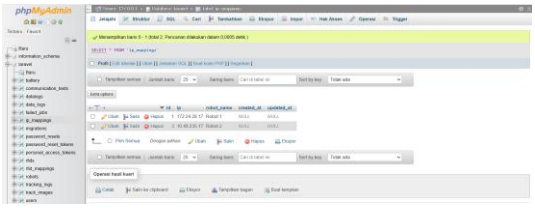

Figure 17. The ip_mappings database table in phpMyAdmin

Then, after each RFID tag's location point is determined, the location data will be stored in the rfid_mappings table in the database. Test results show that the website monitoring system successfully stores RFID mapping data in the rfid_mappings table in the database.

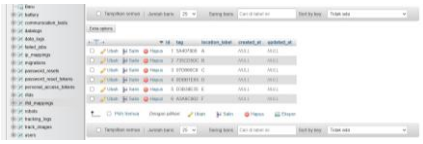

Figure 18. The rfid_mappings database table in phpMyAdmin

TABLE XII
RFID_MAPPINGS TABLE DATA

Id	tag	location_label
1	5A407900	A
2	735CD50C	B
3	07D660C9	C
4	0DBB1E85	D
5	D3B38E35	E
6	A3A9C902	F

When the ESP32 sends data, the system will immediately recognize the identity of the robot connected to the relevant IP, and the location and battery data will automatically be linked to the correct robot. When the ESP32 reads the RFID tag and sends it to the server, the system immediately matches the tag with the specified location so that the position data displayed on the dashboard is in the form of a location point, not a regular RFID tag. Figure 17, 18, and 19 shows RFID and IP mapping testing.

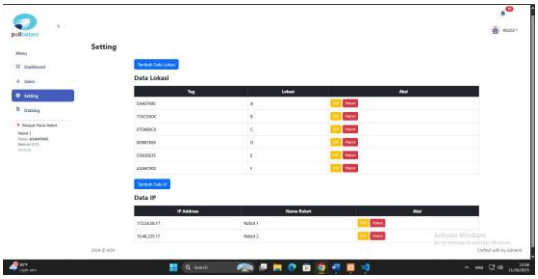

Figure 19. The settings page display on the website

f) Testing on the Dashboard Page

The dashboard displays all incoming robot data using an information card model. Each piece of incoming data will be displayed in the form of a card on the page. The information displayed includes the robot name, IP address, last location, RFID tag, battery capacity, date of entry, and last entry time. Information cards are displayed one by one according to the IP address sent, and the location on the card will be updated automatically if the robot with that IP sends its latest location.

The system uses AJAX to update the page without having to manually refresh the page. AJAX works by periodically sending requests to the server to retrieve data from the robots, battery, rfid_mappings, and ip_mappings tables. Figures 20 and 21 show testing on the dashboard page.

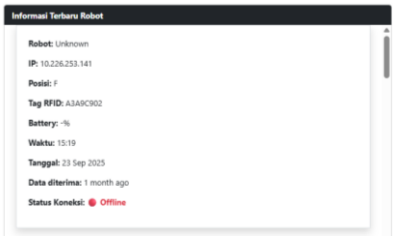

Figure 20. Data displayed on the information card

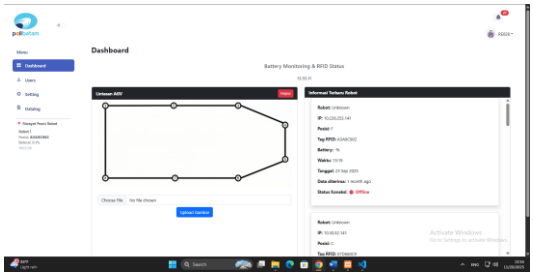

Figure 21. The dashboard page display on the website

g) Login, Registration, and User Storage Testing

TABLE XIII
USERS TABLE DATA

name	email	role
Diran	adriandpratama678@gmail.com	Admin
Sign_up	wird69644@gmail.com	User
RE028	deerun70@gmail.com	Admin

Testing is carried out on the authentication module, which consists of registration, login, session management, and user data storage in the database. This test aims to ensure that only registered users with access rights can log into the website.

The registration form is tested to see whether users can enter a username, email address, and password. The system will then perform validation, such as ensuring that the email is

unique, the password is filled in, and the email format is correct. If the data validation is successful, the user data will be stored in the users table in the database. The password stored in the table is not the original password, but uses automatic hashing from Laravel (bcrypt), so that the password is encrypted and cannot be read directly in the database. Figure 22 shows login, registration, and user storage testing.

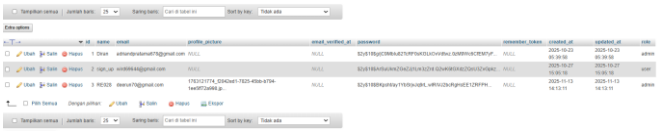

Figure 22. Database *users* in PhpMyAdmin

h) System Malfunction

Malfunctions in the AGV monitoring system can occur due to technical or operational factors. This system works in real time and uses a WIFI network, so even minor disruptions can hinder the process of sending and receiving data. Solutions to this problem involve checking the network used, maintaining devices, validating API endpoints, and so on. There are several possible malfunctions that can occur. The following are some of the malfunctions and solutions found during this study:

- Data not entering the database

This could be caused by an inactive Laravel server or an error in the URL when accessing the server and an incorrect API endpoint. The solution is to ensure that the URL and API port are correct.

- Data does not appear on the dashboard

This is caused by the query not finding the appropriate robot, mapping errors, and browser cache. The solution is to ensure that all robot data has been mapped correctly and to refresh the browser using ctrl-f5.

- User data is not saved

The cause could be an email address that does not match the format, an incomplete registration form, using an email address that is already registered, or a password that does not match the format. The solution is to complete the registration form, provide clear validation for users, and check the users table to ensure the data is saved correctly.

IV. CONCLUSION

Based on the research conducted, the study entitled "Website-Based Battery Capacity and Automated Guided Vehicle (AGV) Position Monitoring System" concluded that this research produced a system that, through software and hardware, can monitor the condition of AGV robots in real time. Using the HTTP POST method, the ESP32 microcontroller can send battery data using the PZEM-017 sensor and AGV position data represented using the MFRC522 RFID sensor to the Laravel server. This data is then stored in a MySQL database and displayed on the website monitoring dashboard page. In addition, the mapping process is also very important so that raw data from batteries and RFID tags sent can be identified as the robot's identity and corresponding location. The database structure consisting of six main tables has been successful in storing data and supporting the needs during the research process.

Furthermore, the combination of the ESP32 microcontroller with the PZEM-017 sensor, MFRC522 RFID, and RS485 proved capable of consistently reading and sending data to the server. The website monitoring dashboard can also display robot information data efficiently, and the role-based admin and user authentication features support data security and make the system more structured. Overall, the Laravel and ESP32-based AGV monitoring system can be an effective solution for monitoring the condition of AGVs in an industrial environment. This system can also be further developed in terms of complementary features related to AGV monitoring in the industrial world.

It can be inferred from the research questions posed in this study that the suggested system effectively addresses each of the stated goals. In addition to implementing IP and RFID mapping techniques for precise AGV identification and handling incomplete, invalid, or unmapped data through validation and structured server responses, the system may show real-time AGV battery capacity and position data via a web-based interface.

Quantitatively speaking, the system showed an average response time of hundreds of milliseconds during regular operation, with a success rate of more than 90% for data storage and transfer. These findings show that the system is appropriate for small- to medium-scale AGV monitoring implementations and operates dependably for real-time monitoring applications.

However, the research method revealed a number of shortcomings. The system has not yet been tested in high-load scenarios with several AGVs sending data at a high frequency at the same time. Furthermore, sensor calibration is necessary for accurate battery monitoring, and the existing system lacks predictive analysis for battery degradation or remaining operating time.

In order to assess system scalability, load and stress testing with several AGVs is advised for future development. To further increase system robustness in actual industrial settings, other enhancements could include including predictive battery health algorithms, displaying AGV movement trajectories on a dynamic map, and strengthening data validation procedures.

Future developments will also concentrate on strengthening RFID data validation logic to guarantee that the system only processes registered and appropriately formatted UID values. In order to increase system resilience in real-world deployment and minimize identification discrepancies, invalid or unregistered RFID data will be discarded and logged for additional examination.

REFERENCES

[1] A. J. Moshayedi, J. Li, and L. Liao, "AGV (automated guided vehicle) robot: Mission and obstacles in design and performance," *Journal of Simulation & Analysis of Novel Technologies in Mechanical Engineering*, vol. 12, no. 4, pp. 0005-0018, 2019.

[2] TechThink Hub Indonesia, "Penerapan Automated Guided Vehicles (AGV) untuk Meningkatkan Efisiensi Pergudangan," TechThink Hub Indonesia, 25 Oct. 2024. [Online]. Available:

https://techthinkhub.co.id/penerapan-automated-guided-vehicles-agv-untuk-meningkatkan-efisiensi-pergudangan/. [Accessed: Oct. 6, 2025].

[3] Y. D. Satriyo, A. Rusdinar, and P. D. Wibawa, "The Position Monitoring System of Automated Guided Vehicle (AGV) In The Industrial Production Process," in Proc. 2018 8th Int. Workshop on Computer Science and Engineering (WCSE), Bangkok, Thailand, 2018, pp. 98-103, doi: 10.18178/wcse.2018.06.017.

[4] M. K. Faahmi, I. Nawawi, and R. Y. Yuliantari, "Sistem Monitoring Security Building Berbasis E-KTP dan NodeMCU ESP8266*," Journal of Applied Electrical Engineering*, vol. 8, no. 2, pp. 117–124, Dec. 2024.

[5] R. A. Dwiyanto, G. A. Mutiara, and M. I. Sari, "Performance analysis of REST API in a real-time IoT-based vehicle monitoring system," *International Journal of Reconfigurable and Embedded Systems (IJRES)*, vol. 14, no. 3, pp. 766–784, Nov. 2025.

[6] L. Wang, S. Jia, X. Li, and S. Wang, "RFID and Vision Based Person Tracking of a Mobile Robot Using Improved Compressive Tracking," *International Conference on Computer Application and System Modeling (ICCASM)*, 2013.

[7] A. Herdiansah, R. I. Borman, and S. Maylinda, "Sistem Informasi Monitoring dan Reporting Quality Control Proses Laminating Berbasis Web Framework Laravel," *Jurnal TEKNO KOMPAK*, vol. 15, no.2, pp. 13-24, n.d

[8] C. Winata, P. R. Rumahorbo, and S. M. Naibaho, "RANCANG BANGUN SISTEM MONITORING PENGISIAN DAYA BATERAI 48 VOLT 12 AH TERINTEGRASI INTERNET OF THINGS," *Jurnal Teknologi Rekayasa Instalasi Listrik*, 2024. [Online]. Available: https://ojs.polmed.ac.id/index.php/JTRIL/article/download/2161/1121

[9] M. Syahputra and A. I. Santoso, "Rancang Bangun Sistem Absensi Otomatis Berbasis RFID Dan ESP32 Di Kampus AMIK Polibisnis Perdagangan," *Jurnal Minfo Polgan*, vol. 14, no.1, May. 2025.

[10] P. Gunoto, A. Rahmadi, and E. Susanti, "PERANCANGAN ALAT SISTEM MONITORING DAYA PANEL SURYA BERBASIS INTERNET OF THINGS," *Sigma Teknika*, vol. 5, no. 2, pp. 285-294, Nov. 2022.

[11] S. Rachman, Z. Ahyadi, and Syarifudin, "KOMUNIKASI ANTARMUKA PROGRAMABLE LOGIC CONTROLLER PADA MODBUS RTU SENSOR SUHU DAN KELEMBABAN UDARA DENGAN DATA LOGGER," *Jurnal Media Elektro*, vol. 11, no. 2, Oct. 2022.

[12] M. H. Tinambunan, A. H. Siregar, and S. Ginting, "FULLSTACK PROGRAMMING: MEMBANGUN APPLICATION PROGRAMMING INTERFACE (API) DENGAN LARAVEL," *Penerbit Tahta Media*, Oct. 2024.

[13] E. Hartati, "SISTEM INFORMASI TRANSAKSI GUDANG BERBASIS WEBSITE PADA CV. ASYURA," *Jurnal Ilmu Komputer*, vol. 3, no. 1, 2022.

[14] S. Manurung, I. Parlina, F. Anggraini, D. Hartama, Jalaluddin, "Penggunaan Sistem Arduino Menggunakan RFID untuk Keamanan Kendaraan Bermotor," *Jurnal Penelitian Inovatif*, vol. 1, no. 2, pp. 139-148, Dec. 2021.

[15] C. Niesa, M. Furqan, Nuzliah, N. Yunanda, L. Fadhliah, Juanda, "Meningkatkan Literasi Digital Mahasiswa melalui Pengelolaan Database UMKM dengan PhpMyAdmin," *Golden Edu: Jurnal Ilmiah Pendidikan Usia Dini dan Dasar*, vol. 1, no. 1, pp. 22-28, Apr. 2025.