Research in Education, Technology, and Multiculture

ISSN: 3025-6763 | Journal's DOI: 10.61436/rietm

Homepage: https://imrecsjournal.com/journals/index.php/rietm

ESP32-Based Inventory Tracker for Goods Recording with Barcode Identification

Sitti Wetenriajeng Sidehabi[1,*], Wahidah[2], St. Nurhayati Jabir[3], Muh. Rizal Wahyudin[4], and Muhammad Fauzan Suharman[5]

Department of Automated Machinery System, Politeknik ATI Makassar, Indonesia

ARTICLE INFO

Article History:

Received: 28 May 2026

Accepted: 21 July 2026

Published: 27 July 2026

Keywords:

Inventory management;
ESP32;
Barcode identification;
Internet of Things (IoT);
Google spreadsheets.

Corresponding Author:

tenri@atim.ac.id

ABSTRACT

Manual inventory recording remains vulnerable to delayed updates, transcription errors, and repeated reconciliation, particularly in warehouse units that still rely on spreadsheet-based data entry. This study aims to design and evaluate a Smart Inventory Tracker for recording goods at the PT Berkah Industri Mesin Angkat Site Kendari by integrating barcode scanning, ESP32-based edge processing, and cloud-based storage. The prototype used an ESP32 development board, a GM66 barcode scanner communicating through a Universal Asynchronous Receiver-Transmitter (UART) interface, a thin-film transistor (TFT) display driven through the Serial Peripheral Interface (SPI) protocol, a 12 V DC power supply with step-down regulation, a JSON reference database hosted on GitHub, and a Google Apps Script endpoint for writing validated transactions to Google Spreadsheets. The evaluation followed an engineering research approach covering hardware assembly, software integration, database matching, transaction logging, and functional testing. System performance was assessed through barcode readability distance testing, registered and unregistered barcode validation, end-to-end response-time measurement, and paired comparison with manual Google Sheets recording. Barcode reading was reliable at 3.7-16.4 cm, while scans at 3.6 cm and 16.5 cm failed outside the empirically verified optical working range. Ten registered barcode samples were successfully displayed and recorded, with an average end-to-end cloud logging time of 4.58 s per item. Manual recording required 45.57 s per item on average; therefore, the automated method reduced the observed recording duration by 40.99 s per item, equivalent to an 89.94% time reduction. Unregistered barcodes were rejected and were not appended to the spreadsheet. These findings indicate that the architecture supports faster and more consistent recording for small- to medium-sized warehouse operations. However, component-level latency logging, offline buffering, and broader network-condition testing remain necessary before large-scale commercial deployment.

To cite this article:

Sidehabi, S. W., Wahidah, Jabir, S. N., Wahyudin, M. R., & Suharman, M. F. (2026). Esp32-based inventory tracker for goods recording with barcode identification. *Research in Education, Technology, and Multiculture*, *5*(3), 208-221. doi: https://doi.org/10.61436/rietm/v5i3.pp208-221

■ INTRODUCTION

Inventory management is a core operational activity because warehouse records determine purchasing decisions, production continuity, maintenance readiness, and a company's ability to trace the movement of goods across the supply chain. In conventional warehouse practice, records are often updated through handwritten logs or spreadsheet forms after goods are received, issued, or transferred. Although spreadsheets are flexible and inexpensive, manual entry introduces a gap between the physical movement of goods and the digital record. This gap can generate stock discrepancies, delayed replenishment decisions, duplicate entries, and additional reconciliation work. Previous studies on digital supply chains emphasize that data timeliness and item-level traceability are essential requirements for Industry 4.0 logistics (Lasi et al., 2014), particularly when firms attempt to connect physical warehouse activities with cloud-based decision support systems (Atzori et al., 2010; Ben-Daya et al., 2019; Gubbi et al., 2013; Hofmann & Rüsch, 2017; Winkelhaus & Grosse, 2020).

The practical problem observed at PT Berkah Industri Mesin Angkat Site Kendari is consistent with this pattern. Before the proposed intervention, warehouse operators recorded item movement directly in Google Sheets after visually checking item labels. A

preliminary baseline observation using ten item-recording samples showed that manual recording required an average of 45.57 s per item. This metric was selected as the quantitative baseline because systematic pre-intervention error percentages were not available in the warehouse log. The delay is operationally significant because every additional item increases queueing time during receipt or issue activities. Manual spreadsheet entry also requires the operator to shift attention from the physical item to the keyboard and screen, increasing the risk of entering an incorrect material code, selecting the wrong category, or postponing the update until several items have accumulated. The baseline therefore indicates that the problem is not only narrative but also measurable as an update-latency burden in routine inventory work.

Digital identification technologies such as barcodes and RFID have been widely used to reduce data entry errors (Unhelkar et al., 2022) and improve the synchronization between physical objects and database records. RFID is advantageous for non-line-of-sight reading (Sarac et al., 2010). However, barcode systems remain attractive for low-cost warehouse environments because labels are inexpensive, the technology is mature, and item codes can be decoded using compact optical modules. Recent warehouse and logistics studies show that automated identification becomes more useful when combined with IoT connectivity, edge processing, and cloud data storage, because the device can validate an item locally and update a shared database immediately after scanning (Lee & Lee, 2015; Patil & Kale, 2016; Rejeb et al., 2020). In this context, the Internet of Things is not limited to sensing (Nižetić et al., 2020); it also provides a mechanism for linking an embedded device, a reference database, and an online reporting platform into a single recording workflow.

The ESP32 microcontroller is ideal for this application due to its integration of Wi-Fi connectivity, programmable GPIO, and peripheral interfaces such as UART, SPI, and I2C on an economical embedded platform. According to the ESP32 datasheet (Espressif Systems, 2024), the chip provides multiple serial interfaces. It operates as a Wi-Fi and Bluetooth system-on-chip (Ray, 2018), making it well-suited for compact IoT data loggers and industrial automation prototypes (Espressif Systems, n.d.-a; Hercog et al., 2023). In inventory applications, this capability allows the microcontroller to receive barcode data via UART, display validated information on an SPI-connected TFT screen, and communicate with cloud services via HTTP requests. Google Apps Script can also be deployed as a web application containing doGet(e) or doPost(e) functions (Li et al., 2023), while Google Sheets provides a REST-oriented spreadsheet environment for storing and updating tabular records (*Google*, n.d.). These characteristics allow a low-cost embedded device to interact with cloud-based spreadsheets without requiring a dedicated local server (Melo et al., 2021).

Nevertheless, most previously reported low-cost IoT inventory systems focused primarily on barcode-based identification or cloud connectivity as separate functionalities rather than on an integrated transaction-processing architecture. Pratama et al. (2025) and Santosa et al. (2024) developed an ESP32-based IoT barcode-scanning system for warehouse stock opname, emphasizing wireless communication and real-time synchronization with a Firebase cloud database. However, scanned barcode data were transmitted directly to the cloud without an intermediate validation step against an external reference database prior to transaction recording. Likewise, Nico (2024) developed a barcode-assisted inventory application, but the architecture relied primarily on local database processing and did not separate barcode verification from cloud transaction logging. Similarly, Widoretno et al. (2025) implemented an ESP32-based QR code logistics system integrated with Google Spreadsheets for automatic cloud recording; however, the system focused on direct data transmission and did not employ an external JSON-based master database for pre-transaction validation. Consequently, previous studies have generally emphasized either automatic identification or cloud connectivity (Szymańska et al., 2017; Tubis & Rohman, 2023), whereas the proposed architecture integrates remote JSON validation, edge processing, operator confirmation, and serverless cloud recording into a unified inventory transaction workflow.

In contrast, the architecture proposed in this study introduces a sequential validation workflow that separates four functional stages: (1) barcode acquisition through the GM66 scanner via the UART interface; (2) verification of the scanned barcode against a remotely hosted JSON master database stored on GitHub; (3) immediate visualization of validated item information on the TFT display to provide operator confirmation before data submission; and (4) automated transaction recording to Google Sheets through a Google Apps Script web service. Unlike conventional implementations in which barcode scanning

directly inserts records into a local database, the proposed architecture introduces an intermediate validation layer that ensures only registered inventory items are accepted before cloud synchronization.

This layered validation system offers two significant practical benefits. First, maintaining the master inventory database as an external JSON repository enables administrators to update inventory references without modifying or reflashing the ESP32 firmware, thereby simplifying long-term maintenance and reducing deployment costs. Second, the use of Google Apps Script as a lightweight middleware eliminates the need for dedicated web servers or commercial cloud databases while still supporting centralized inventory logging. Consequently, the novelty of this study lies not merely in the use of ESP32 or barcode technology, both of which have been widely adopted in previous research, but in the proposed integration architecture that combines remote JSON validation, edge processing, operator confirmation, and serverless cloud recording into a single low-cost inventory transaction workflow suitable for small- and medium-scale warehouse environments.

This architecture also clarifies the limitation of the phrase 'without manual intervention.' In the revised design description, automation refers to the step that records the transaction after an item is scanned. It does not mean that warehouse master data never requires human governance. Upon the introduction of a new material category, an authorized administrator is required to revise the master item list and either regenerate or finalize the JSON reference file utilized by the device. This distinction is crucial because master-data maintenance is a controlled administrative task, while transaction recording is a repetitive process aimed at by the proposed automation. Clarifying this separation helps prevent overestimating the system's autonomy and ensures the claim matches the prototype's real workflow.

Based on the identified problem and supporting literature, this study investigates the design and performance of an ESP32-based Smart Inventory Tracker for barcode-based warehouse recording. The study is guided by four research questions: RQ1: How can an ESP32-based barcode identification device be architected to match scanned item codes with an online JSON reference database and record validated transactions in Google Sheets? RQ2: What scanning-distance range enables the GM66 barcode scanner to read warehouse barcodes reliably under the tested conditions? RQ3: How does the end-to-end recording time of the proposed system compare with manual spreadsheet entry? RQ4: How does the system process registered and unregistered barcode inputs during transaction recording? The working hypothesis is that the proposed ESP32-based Smart Inventory Tracker reduces the average recording time compared with manual spreadsheet entry while correctly rejecting unregistered barcode samples.

■ METHOD

Research Design, Site, and Baseline Observation

This research was conducted at the warehouse facility of PT Berkah Industri Mesin Angkat Site Kendari using an engineering research approach. The study consisted of requirements identification, prototype design, hardware assembly, firmware development, cloud service integration, and functional testing. Engineering research was selected because the objective was to design and evaluate an artifact rather than to test a behavioral or social intervention. The baseline manual process was documented before the automated system was evaluated. Ten representative item samples were recorded manually by the warehouse operator using Google Sheets on the available office computer. The elapsed time was measured using a digital stopwatch, starting when the operator began identifying the item label and ending when the corresponding record was completely entered into Google Sheets. The mean manual update duration was 45.57 s per item.

Hardware Architecture

Figure 1 illustrates the comprehensive architecture of the Smart Inventory Tracker system, which is strategically divided into two primary operational environments: local hardware and cloud infrastructure. The local environment comprises several interconnected components: a GM66 barcode-scanning module, an ESP32 microcontroller, and a TFT LCD screen. The barcode scanner optically captures item codes and transmits the resulting digital data to the ESP32 over a stable UART connection. Upon receiving this input, the ESP32 immediately processes it and displays the corresponding item information on the TFT screen, enabling the warehouse operator to verify directly.

The local hardware consisted of an ESP32 development board, a GM66 barcode scanner module, a TFT LCD, an expansion board, and a 12 V DC adapter. The GM66 module was allocated to UART2 on the ESP32, with scanner TX connected to ESP32 GPIO16 (RX2) and scanner RX connected to ESP32

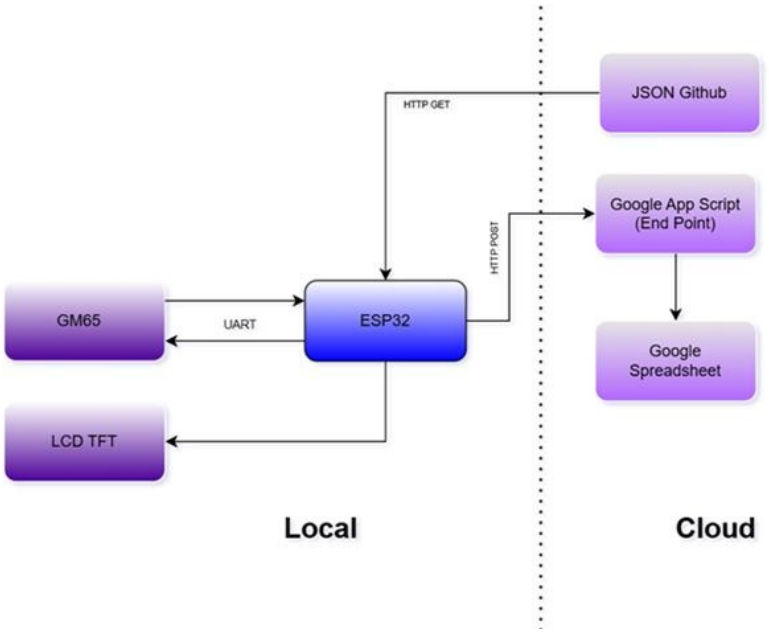

Figure 1. Block diagram of the smart inventory tracker architecture.

GPIO17 (TX2). The TFT display was allocated to the SPI bus, with GPIO18 as SCK, GPIO23 as MOSI, GPIO19 as MISO, GPIO5 as chip select, GPIO2 as data/command, and GPIO4 as reset. The GND lines of all modules were connected to a common ground. This pin allocation follows the ESP32 capability to support multiple UART and SPI interfaces (*Espressif Systems*, n.d.-a).

The system used a 12 V DC adapter as the external supply input. Because the ESP32 chip operates in a low-voltage domain and the datasheet specifies 3.3 V conditions for digital operation, the 12 V input was not connected directly to the microcontroller. Instead, power was routed through the expansion-board regulation stage: a DC-DC step-down stage provides the board-level 5 V rail, and the onboard regulator supplies the 3.3 V rail used by the ESP32 and logic-level peripherals. This explanation was added to prevent the misleading impression that the ESP32 was powered directly from 12 V. A stable, regulated supply is required to avoid brownout resets during Wi-Fi transmission and scanning.

Software Architecture and Data-Update Workflow

The firmware initialized the TFT interface, UART scanner interface, Wi-Fi connection, Network Time Protocol synchronization, HTTP client, and barcode-processing routine. After receiving a barcode string from the GM66 scanner, the ESP32 validated its format and sent an HTTP GET request to retrieve the JSON reference file hosted on a GitHub repository. The JSON file contained the material code, material description, and item category used for matching. If the scanned code was present in the reference file, the ESP32 displayed the matching information on the TFT screen and submitted the transaction to a Google Apps Script web endpoint via an HTTP POST request. Apps Script web applications can process GET and POST requests via the doGet(e) and doPost(e) functions, while Google Sheets can store appended transaction records in a shared cloud spreadsheet (*Google*, n.d.).

The revised workflow distinguishes transaction automation from master-data maintenance. When a new item category or material code is introduced, an authorized administrator must update the master list and regenerate the JSON reference file through a controlled commit or upload to the GitHub repository. The ESP32 then retrieves the newest reference file during subsequent synchronization or scanning. Therefore, the system does not eliminate manual governance of warehouse master data; it eliminates repetitive manual typing during transaction recording. This clarification resolves the contradiction between the use of a static JSON file and the previous claim that the whole system operated without manual intervention.

Connection-Failure Handling

The tested prototype displayed a connection failure notification when Wi-Fi initialization failed, and an upload failure notification when the POST request to Google

Apps Script failed. The original prototype did not include a verified persistent offline queue using SPIFFS (Espressif Systems, n.d.-b) or an SD card. Consequently, the end-to-end tests were conducted under available Wi-Fi connectivity, and failed uploads required the operator to repeat the scan after the connection recovered. The revised design recommendation is to implement SPIFFS- or SD-card-based transaction buffering so that barcode, timestamp, material code, and upload status can be stored locally before being flushed to Google Sheets after reconnection. ESP-IDF provides SPIFFS support for storing files in flash memory, which is well-suited for a small pending-transaction queue (*GitHub*, n.d.).

Testing Procedure

System testing covered four aspects: scanning distance, handling of registered barcodes, handling of unregistered barcodes, and recording-time comparison. The scanning-distance test used the original measured boundary points of 3.6, 3.7, 3.8, 3.9, 4, 8, 12, 16, 16.1, 16.2, 16.3, 16.4, and 16.5 cm. Because the present dataset was collected using boundary-range distances, the results are interpreted as an empirical working-range verification rather than as a complete optical response curve. Future studies should apply a uniform 2 cm interval to model gradual changes in barcode readability across scanning distances.

Response time was measured as end-to-end recording duration from barcode scan completion to confirmation of data entry in Google Spreadsheets. This value includes local parsing, online JSON retrieval, cloud endpoint response, and spreadsheet writing. Because the original firmware did not log timestamps separately for UART receipt, GitHub GET resolution, and Google Sheets POST execution, the reported 4.58 s average must be interpreted as total cloud logging latency rather than the pure scanner or microcontroller processing speed. For future replication, firmware instrumentation should record millis() timestamps at four points: scanner data arrival, JSON GET completion, POST request completion, and spreadsheet confirmation.

The operational sequence commences when the system is powered on, marked by an initialization phase and the presentation of the initial interface on the TFT display seen in Figure 2. Following this step, the microcontroller attempts to establish a wireless network connection and to synchronize its internal clock using the Network Time Protocol (Wen & Zhu, 2022). If the network connection attempt is unsuccessful, the system shall present a notification indicating a connection failure and subsequently suspend all further operations.

Conversely, upon establishment of the Wi-Fi connection, the system displays a 'Ready for Scan' message, indicating that the barcode scanner is prepared for operation. The operator then scans an item using the GM66 optical scanner. The firmware first determines whether barcode data have been successfully received from the scanner. If no barcode is detected, the system displays a Scan Failure (No barcode detected) message and returns to the scanning state. When barcode data are successfully captured, the firmware performs barcode format validation. If the decoded data do not satisfy the expected barcode format, the system displays an Invalid Barcode Format notification and terminates the current transaction. Only barcode data with a valid format proceed to the next stage, where the ESP32 retrieves the JSON reference file from the GitHub repository for database verification.

Subsequently, the ESP32 compares the validated barcode with the downloaded JSON reference database. If no matching record is found, the system displays a Data Not Found (Unregistered barcode) message and returns to the ready state without transmitting any transaction. If a matching record exists, the corresponding item information is displayed on the TFT screen for operator confirmation before being transmitted to the Google Apps Script web service. After successful transmission, the validated transaction is automatically recorded in Google Spreadsheets; otherwise, the system displays a Failed to Upload Data notification and waits for the next scanning cycle.

Provided the transmission is successful, the item information is automatically and permanently recorded into Google Spreadsheets, which serves as the primary system database. This final step concludes the operational workflow, signifying the successful completion of one entire inventory recording cycle.

■ RESULTS AND DISCUSSION

The findings of this research provide a comprehensive evaluation of the designed system. This evaluation includes determining the operational scanning distance, measuring cloud logging response time, testing barcode handling for registered and unregistered barcodes, and comparing the automated recording process with the manual spreadsheet method. Figure 3 depicts the wiring configuration used in the Inventory Tracker system. The primary device receives power

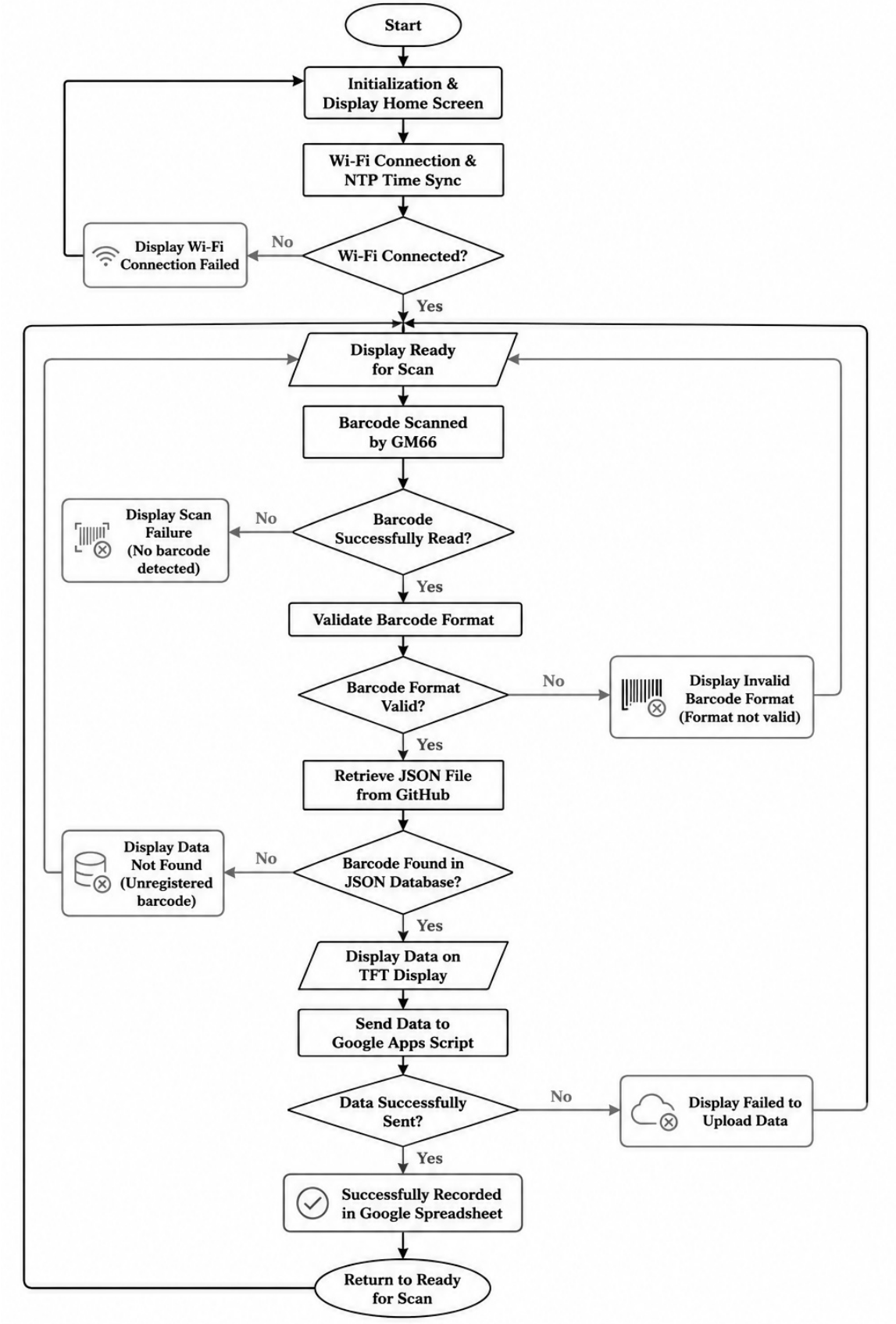

Figure 2. Operational flowchart of barcode validation and cloud recording.

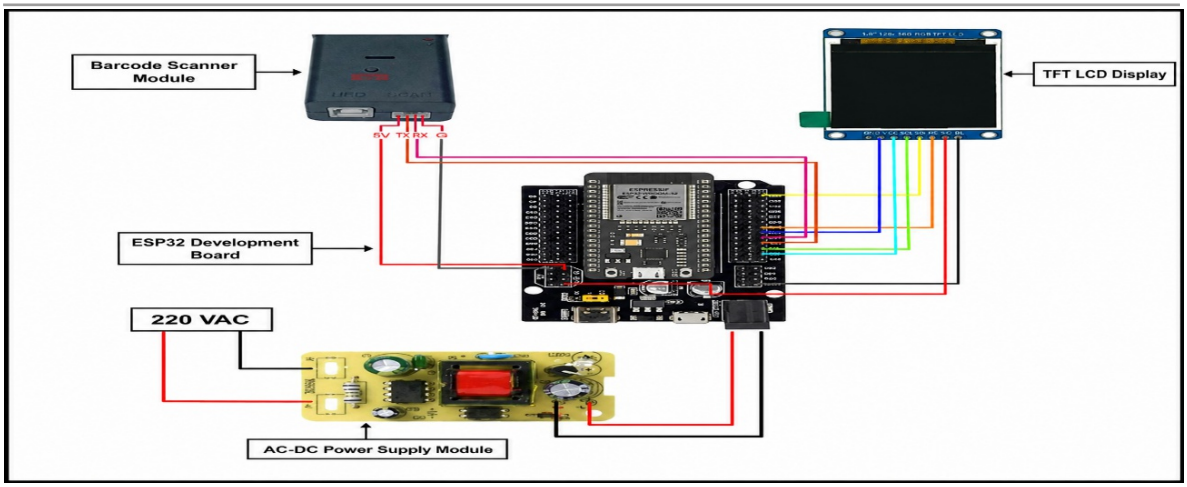

Figure 3. Wiring diagram showing the esp32, scanner, tft display, and regulated power module

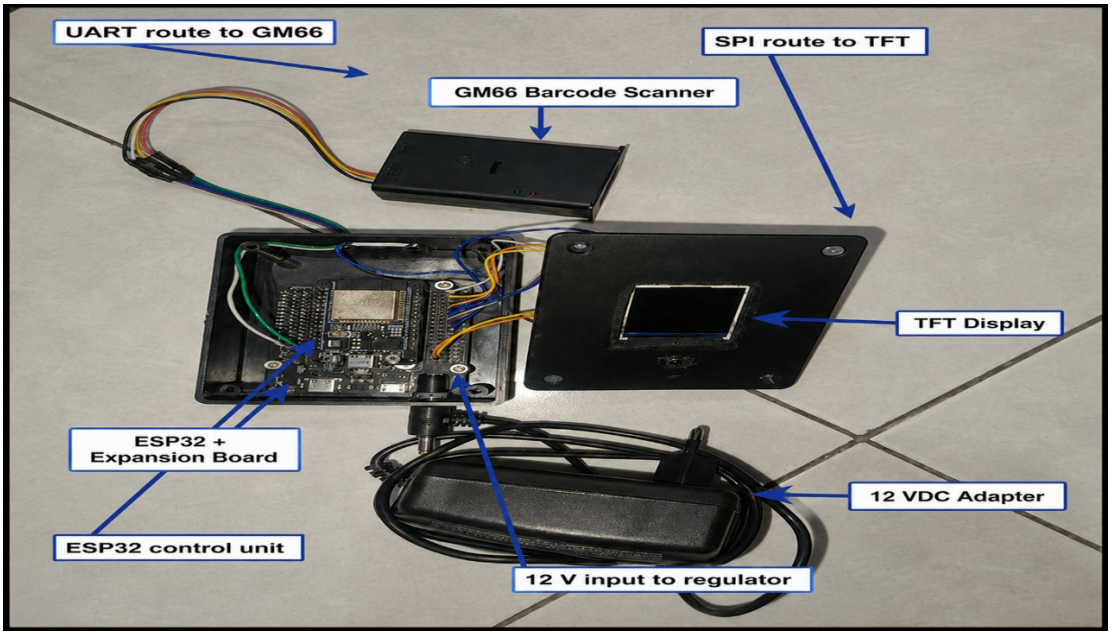

Figure 4. Annotated prototype of the smart inventory tracker with visible physical routes.

from a standard AC source through a 12 V DC adapter and regulation stage. After power-on, the ESP32 executes its startup sequence, initializes the scanner and display, connects to Wi-Fi, and synchronizes time through Network Time Protocol. When an operator scans an item, barcode data are transmitted to the ESP32 via UART, then matched against the JSON reference database, and finally displayed and submitted to Google Sheets.

Figure 4 presents the annotated prototype. The annotation clarifies the physical cable route between the GM66 scanner, the ESP32 development board, the TFT display, and the 12 V DC adapter/regulation path. This visual labeling was added to make the installation photograph consistent with the wiring diagram and to improve reproducibility for readers who intend to build a similar prototype.

Based on the testing results in Table 1, the distance evaluation indicates that the prototype successfully read barcode samples within the observed range of 3.7-16.4 cm. Reading failed at 3.6 cm and 16.5 cm. The near-distance failure should be interpreted as an empirical limitation in the scanner module's focus range under the tested label and lighting conditions. The exact focal length of the GM66 lens could not be confirmed from the available manufacturer documentation; therefore, the revised discussion avoids claiming a precise focal length and instead reports the experimentally observed working range. The far-distance failure is consistent with reduced barcode resolution when the printed code is positioned beyond the optical module's usable

Table 1. Boundary-range scanning-distance test results

Scanning distance (cm)	Detection outcome	Interpretation
3.6	Unreadable	Below is the observed minimum readable range
3.7	Readable	Lower boundary of observed working range
3.8	Readable	Within the observed working range
3.9	Readable	Within the observed working range
4	Readable	Within the observed working range
8	Readable	Within the observed working range
12	Readable	Within the observed working range
16	Readable	Within the observed working range
16.1	Readable	Within the observed working range
16.2	Readable	Within the observed working range
16.3	Readable	Within the observed working range
16.4	Readable	Upper boundary of observed working range
16.5	Unreadable	Above observed maximum readable range

reading range.

The original distance points were not evenly distributed; therefore, Figure 5 uses a scatter plot rather than a line chart. This format avoids implying continuous interpolation between sparse distance points and makes the two failed boundary observations visible. The measured result is sufficient to guide practical operation, as operators can place the barcode within the 3.7–16.4 cm range. However, it is insufficient for detailed optical performance modeling. Future testing with uniform 2 cm intervals is required before gradual degradation of barcode readability can be discussed as a function of scanning distance. Therefore, the present study limits its claim to empirical verification within the working range rather than to continuous modeling of optical accuracy.

The scanning-distance evaluation was conducted at 13 distances to identify the readable and unreadable ranges of the GM66 barcode scanner. The results showed that 11 out of 13 tested distances produced successful readings, while two distances resulted in reading failures. This indicates a scanning success rate of 84.62% and a failure rate of 15.38% under the tested conditions. Successful barcode readings occurred within the range of 3.7-16.4 cm, suggesting that this interval represents the empirically verified operational range of the prototype. In contrast, the barcode could not be detected at 3.6 cm and 16.5 cm. The reading failure at 3.6 cm indicates that the barcode was positioned closer than the scanner's effective optical working range, which may reduce image clarity and decoding reliability. Meanwhile, the failure at 16.5 cm suggests that the barcode exceeded the scanner's maximum readable distance, resulting in insufficient optical resolution for accurate recognition. These findings indicate that

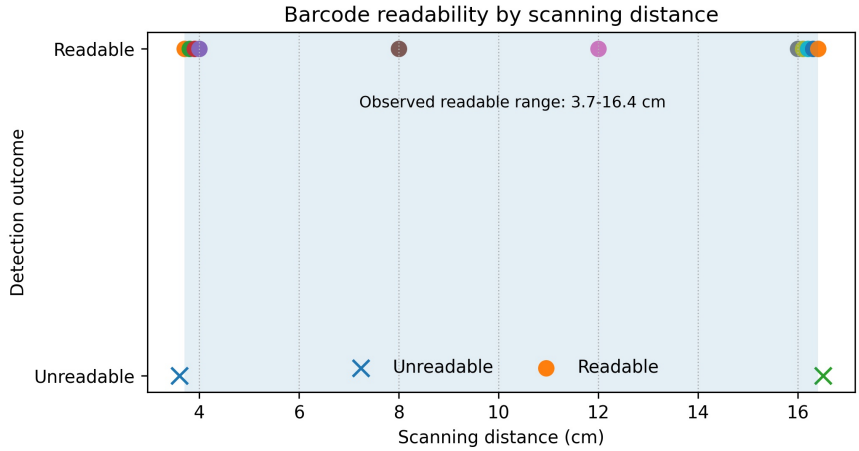

Figure 5. Scatter plot of barcode detection outcomes at tested scanning distances

maintaining the scanning distance within the verified range is essential to ensure stable barcode detection during warehouse inventory recording.

The transmission test in Table 2 shows that the end-to-end cloud logging time ranged from 4.1 to 4.9 s, with an average of 4.58 s per item. The observed response time includes local execution, network communication, cloud-service response, and spreadsheet writing; however, the contribution of each component could not be quantified because detailed network metrics and firmware-level timestamps were not recorded.

The present study did not document network-quality parameters such as ICMP ping latency, available bandwidth, or packet loss during the experimental sessions. Therefore, the reported average response time of 4.58 s should be interpreted as the observed end-to-end transaction duration under the available network conditions, rather than as a benchmark obtained under controlled network performance settings. Consequently, variations in wireless network quality may have influenced the measured response time. Future studies should record network latency, bandwidth, and packet-loss metrics simultaneously with transaction logging to isolate the effect of communication quality on overall system performance.

As timestamp logging was not implemented during the development of the firmware, the internal contributions of UART reception, HTTP GET request execution, JSON

Table 2. Data transmission response time test results to google sheets

No	Barcode Sample	Response Time (Seconds)	Status
1	CC-005-000357	4.4	Read and recorded successfully
2	AB-015-000470	4.8	Read and recorded successfully
3	BA-002-001023	4.5	Read and recorded successfully
4	BA-002-001035	4.8	Read and recorded successfully
5	CA-002-000006	4.2	Read and recorded successfully
6	BA-002-001026	4.7	Read and recorded successfully
7	BA-002-001041	4.6	Read and recorded successfully
8	BD-001-000787	4.8	Read and recorded successfully
9	AA-002-000203	4.9	Read and recorded successfully
10	BA-002-001068	4.1	Read and recorded successfully
	Average	**4.58**	

Table 3. Testing results of barcodes

No.	Barcode Sample	Initial Barcode Status	TFT status string	Google Sheets output	Remarks
1		Registered	Registered item displayed: AIR ACCU	Code, material description, and category appended	Matched and transmitted
2		Registered	Registered item displayed: ACCU N120	Code, material description, and category appended	Matched and transmitted
3		Registered	Registered item displayed: BAN DALAM 10.00-20	Code, material description, and category appended	Matched and transmitted
4		Registered	Registered item displayed: BAN DALAM 8.25-15	Code, material description, and category appended	Matched and transmitted
5		Registered	Registered item displayed: MARSET 10.00-20.00	Code, material description, and category appended	Matched and transmitted
6		Unregistered	Data Not Found	Not appended	Unregistered barcode
7		Unregistered	Data Not Found	Not appended	Unregistered barcode
8		Unregistered	Data Not Found	Not appended	Unregistered barcode
9		Unregistered	Data Not Found	Not appended	Unregistered barcode
10		Unregistered	Data Not Found	Not appended	Unregistered barcode

parsing, and HTTP POST submission could not be isolated retrospectively. Consequently, only the aggregate end-to-end transaction time is reported in this study. Future implementations should incorporate firmware-level timestamp logging (e.g., using the millis() function) at each processing stage to quantify each subsystem's contribution to the total response time. Although the 4.58 s response time is faster than the manual recording baseline in this study, it should be interpreted as a prototype-level performance indicator rather than as a commercial throughput benchmark. High-throughput cashier and warehouse systems generally prioritize near-real-time transaction confirmation; therefore, further benchmarking

Table 4. Testing results for the comparison of manual recording time and the automated system

No	Item Sample	Manual Recording Time (Seconds)	System Recording Time (Seconds)	Time Difference (Seconds)
1	Item 1	45.5	4.5	41.0
2	Item 2	46.7	4.7	42.0
3	Item 3	45.2	4.8	40.4
4	Item 4	44.4	4.6	39.8
5	Item 5	45.5	4.9	40.6
6	Item 6	46.1	4.8	41.3
7	Item 7	45.3	4.8	40.5
8	Item 8	45.3	4.1	41.2
9	Item 9	46.1	4.2	41.9
10	Item 10	45.6	4.4	41.2
Average		**45.57**	**4.58**	**40.99**

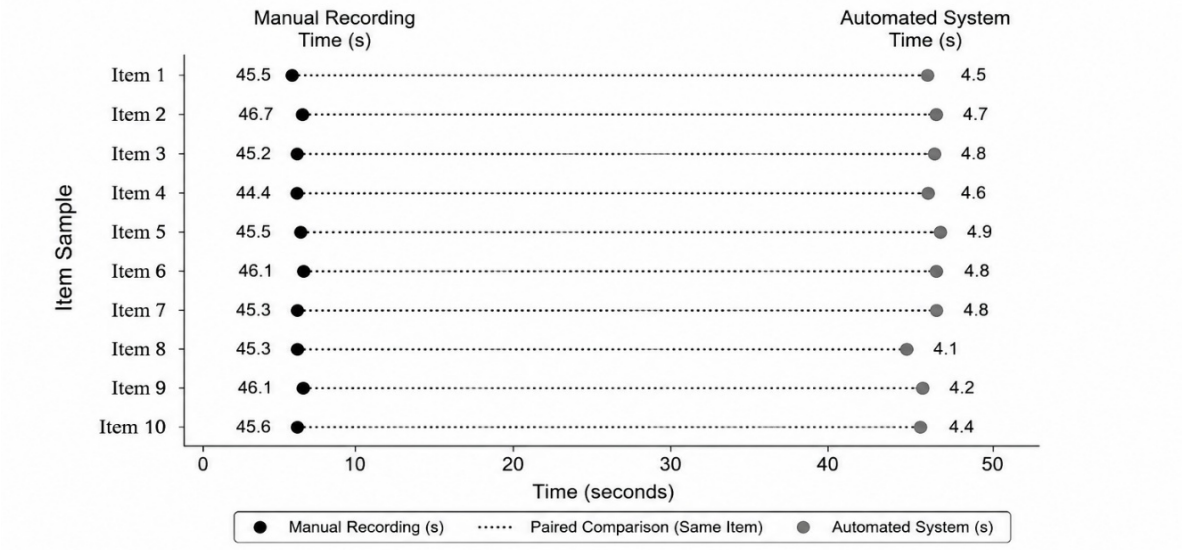

Figure 6. Dumbbell plot comparing manual recording time and automated system recording time across 10 item samples. Each line connects paired observations from the same item.

Manual Recording Time (s)	Automated System Time (s)
Mean = 45.57 Median = 45.55	Mean = 4.58 Median = 4.65
Q1 = 45.20 Q3 = 46.10 Min = 44.40 Max = 46.70	Q1 = 4.35 Q3 = 4.80 Min = 4.10 Max = 4.90

Note: Lower time indicates better performance.

Figure 6. Dumbbell plot comparing manual recording time and automated system recording time across ten paired item samples.

against commercial inventory systems is required before claiming suitability for large-scale logistics operations.

Despite the encouraging response-time results, one limitation of the present study is the absence of component-level latency measurements. Because the reported response time represents the cumulative duration of UART communication, HTTP GET retrieval from the GitHub JSON repository, local JSON parsing, HTTP POST transmission to Google Apps Script, and spreadsheet updating, it is not possible to determine which subsystem contributes most to the total delay. Under unstable wireless network conditions, increased latency during HTTP communication may substantially prolong the overall transaction time, even when local barcode acquisition and processing remain fast. This condition may reduce operator efficiency in high-throughput warehouse environments where multiple items are scanned consecutively. Furthermore, because the prototype does not implement a persistent offline transaction queue, temporary network failures require operators to rescan barcodes after connectivity is restored, potentially increasing workload and introducing delays in inventory recording. Future implementations should therefore incorporate firmware-level timestamp instrumentation, together with a persistent offline buffering mechanism (e.g., SPIFFS or SD card-based

storage), to facilitate bottleneck identification and maintain transaction continuity during network interruptions. Therefore, the reported average response time of 4.58 s should be interpreted as an overall system-level performance indicator rather than as evidence of the efficiency of individual processing stages or network communication components.

The combined barcode validation results presented in Table 3 demonstrate that the proposed system correctly distinguished between registered and unregistered barcode samples. All registered barcodes stored in the JSON reference database were successfully matched, displayed on the TFT screen with the corresponding item information, and automatically appended to Google Spreadsheets. In contrast, all unregistered barcode samples were correctly identified as invalid, displayed the message "Data not found/ invalid barcode" on the TFT screen, and were rejected without being appended to the spreadsheet. These results confirm that the proposed validation mechanism effectively prevents unauthorized or unknown barcode data from being recorded, thereby improving the integrity and reliability of inventory transaction records

Based on the testing results presented in Table 4, comparing manual recording time with the automated system across 10 sample items, the average manual recording time is 45.57 seconds. In contrast, the automated system requires only 4.58 seconds. Consequently, the automated system saves an average of 40.99 seconds per item, representing an operational acceleration of approximately 89.94 percent compared to the manual method. The automated method reduced recording time mainly because the barcode scanner captured the item code directly, the ESP32 performed deterministic matching against the reference file, and the Google Apps Script endpoint appended the validated record without requiring the operator to retype the material code, description, and category. The time-saving, therefore, came from removing repeated visual checks and keyboard entry, not from computational complexity.

Figure 6 presents a dumbbell plot comparing manual recording time and automated system recording time for each of the ten item samples. Each horizontal line connects paired observations for the same item, allowing direct comparison of recording durations before and after the implementation of the proposed Smart Inventory Tracker. In every paired observation, the automated recording time is substantially lower than the corresponding manual recording time, demonstrating a consistent reduction in transaction duration across all tested samples. The average manual recording time was 45.57 s per item, whereas the proposed system required only 4.58 s per item, resulting in an average time reduction of 40.99 s (89.94%). The relatively uniform separation between paired observations indicates that the improvement was consistently achieved across item samples, suggesting that the proposed workflow provides stable performance for routine warehouse inventory recording (Jarašūnienė et al., 2023; Khan et al., 2022). These findings support the effectiveness of integrating barcode identification, ESP32 edge processing, JSON-based reference validation, and automated cloud logging to reduce repetitive manual data-entry activities while improving recording efficiency.

■ CONCLUSION

The ESP32-based Smart Inventory Tracker successfully demonstrates a low-cost barcode identification architecture for warehouse recording. The system architecture addressed the research objective by integrating the GM66 scanner, ESP32 edge processor, JSON reference database, TFT display, and Google Spreadsheets logging endpoint into a cohesive transactional workflow. Under the tested conditions, the prototype read barcode samples within the empirically verified range of 3.7-16.4 cm, recorded ten registered barcode samples with an average end-to-end cloud logging time of 4.58 s per item, and rejected unregistered barcode samples so that they were not appended to the spreadsheet. Compared with manual Google Sheets recording, the automated workflow reduced the observed average recording duration by 89.94%. These findings indicate that the proposed system can improve the speed and consistency of inventory recording in small- to medium-sized warehouse operations (Melo et al., 2021; Tubis & Rohman, 2023). However, the study did not evaluate cybersecurity protection, API-key hardening, encrypted payload transmission, persistent offline buffering, or component-level latency decomposition. Future work should include SPIFFS- or SD-card-based local queuing, timestamp logging for UART/GET/ POST processes, network-condition testing, and a uniform scanning-distance retest to support broader deployment.

■ DECLARATION OF GENERATIVE AI USAGE IN THE WRITING PROCESS

During the drafting of this manuscript, the author(s) utilized ChatGPT from OpenAI and

Claude to improve readability and correct English grammar. Following the use of this tool, the author(s)reviewed and revised the content as necessary and accept full responsibility for the final content of the article.

■ REFERENCES

Atzori, L., Iera, A., & Morabito, G. (2010). The Internet of Things: A survey. *Computer Networks*, *54*(15), 2787–2805. https://doi.org/10.1016/j.comnet.2010.05.010

Ben-Daya, M., Hassini, E., & Bahroun, Z. (2019). Internet of things and supply chain management: A literature review. *International Journal of Production Research*, *57*(15–16), 4719–4742. https://doi.org/10.1080/00207543.2017.1402140

Espressif Systems. (2024). *ESP32 technical reference manual* (Version 5.2). https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf

Espressif Systems. (n.d.-a). SPI master driver —ESP32—ESP-IDF programming guide v6.0.2 documentation. Retrieved July 14, 2026, from https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/peripherals/spi_master.html

Espressif Systems. (n.d.-b). SPIFFS filesystem —ESP32—ESP-IDF programming guide v6.0.2 documentation. Retrieved July 14, 2026, from https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/storage/spiffs.html

GitHub. (n.d.). GitHub REST API documentation. Retrieved July 14, 2026, from https://docs.github.com/en/rest

Google. (n.d.). Google Sheets API overview. Google for Developers. Retrieved July 14, 2026, from https://developers.google.com/workspace/sheets/api/guides/concepts

Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, *29*(7), 1645–1660. https://doi.org/10.1016/j.future.2013.01.010

Hercog, D., Lerher, T., Truntič, M., & Težak, O. (2023). Design and implementation of ESP32-based IoT devices. *Sensors*, *23*(15), Article 6739. https://doi.org/10.3390/s23156739

Hofmann, E., & Rüsch, M. (2017). Industry 4.0 and the current status as well as future prospects on logistics. *Computers in Industry*, *89*, 23–34. https://doi.org/10.1016/j.compind.2017.04.002

Jarašūnienė, A., Čižiūnienė, K., & Čereška, A. (2023). Research on impact of IoT on warehouse management. *Sensors*, *23*(4), Article 2213. https://doi.org/10.3390/s23042213

Khan, M. G., Huda, N. U., & Zaman, U. K. U. (2022). Smart warehouse management system: Architecture, real-time implementation and prototype design. *Machines*, *10*(2), Article 150. https://doi.org/10.3390/machines10020150

Lasi, H., Fettke, P., Kemper, H. G., Feld, T., & Hoffmann, M. (2014). Industry 4.0. *Business & Information Systems Engineering*, *6*(4), 239–242. https://doi.org/10.1007/s12599-014-0334-4

Lee, I., & Lee, K. (2015). The Internet of Things (IoT): Applications, investments, and challenges for enterprises. *Business Horizons*, *58*(4), 431–440. https://doi.org/10.1016/j.bushor.2015.03.008

Li, Z., Guo, H., Xu, W., & Chen, B. (2023). Serverless computing: State-of-the-art, challenges and opportunities. *IEEE Transactions on Services Computing*, *16*(2), 1522–1539. https://doi.org/10.1109/TSC.2022.3166553

Melo, G. C. G., Torres, I. C., de Araújo, I. B. Q., Brito, D. B., & Barboza, E. A. (2021). A low-cost IoT system for real-time monitoring of climatic variables and photovoltaic generation for smart grid application. *Sensors*, *21*(9), Article 3293. https://doi.org/10.3390/s21093293

Nico, J. A. N. (2024). Rancang bangun prototipe inventaris barang berbasis IoT menggunakan mikrokontroler ESP32 dan sensor radio frequency identification [*Undergraduate thesis*, Universitas Islam Balitar]. Universitas Islam Balitar Repository. https://repository.unisbablitar.ac.id

Nižetić, S., Šolić, P., López-de-Ipiña González-de-Artaza, D., & Patrono, L. (2020). Internet of Things (IoT): Opportunities, issues and challenges towards a smart and sustainable future. *Journal of Cleaner Production*, *274*, Article 122877. https://doi.org/10.1016/j.jclepro.2020.122877

Patil, K. A., & Kale, N. R. (2016). A model for smart agriculture using IoT. *2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC)*, 543–545. https://doi.org/10.1109/ICGTSPICC.2016.7955360

Pratama, S. A. P., Hidayat, H., Rahim, W. G. E., & Nugraha, M. A. (2025). IoT-based barcode scanning system implementation

using ESP32 for enhancing warehouse stock opname efficiency. *International Journal of Research and Applied Technology (INJURATECH)*, *5*(1), 268–276. https://ojs.unikom.ac.id/index.php/injuratech/article/view/19200

Ray, P. P. (2018). A survey on Internet of Things architectures. *Journal of King Saud University – Computer and Information Sciences*, *30*(3), 291–319. https://doi.org/10.1016/j.jksuci.2016.10.003

Rejeb, A., Simske, S., Rejeb, K., Treiblmaier, H., & Zailani, S. (2020). Internet of Things research in supply chain management and logistics: A bibliometric analysis. *Internet of Things*, *12*, Article 100318. https://doi.org/10.1016/j.iot.2020.100318

Santosa, S. H., Hidayat, A. P., Siskandar, R., & Husyairi, K. A. (2024). IoT-based barcode scanning system for production and warehouse management. *Batrisya: Journal of Electrical Engineering and Computer Science*, *2*(1), 45–58. https://batrisyaedu.com/journal/index.php/batrisya/article/view/18

Sarac, A., Absi, N., & Dauzère-Pérès, S. (2010). A literature review on the impact of RFID technologies on supply chain management. *International Journal of Production Economics*, *128*(1), 77–95. https://doi.org/10.1016/j.ijpe.2010.07.039

Szymańska, O., Adamczak, M., & Cyplik, P. (2017). Logistics 4.0—A new paradigm or set of known solutions? *Research in Logistics & Production*, *7*(4), 299–310. https://doi.org/10.21008/j.2083-4950.2017.7.4.2

Tubis, A. A., & Rohman, J. (2023). Intelligent warehouse in Industry 4.0—Systematic literature review. *Sensors*, *23*(8), Article 4105. https://doi.org/10.3390/s23084105

Unhelkar, B., Joshi, S., Sharma, M., Prakash, S., Mani, A. K., & Prasad, M. (2022). Enhancing supply chain performance using RFID technology and decision support systems in the Industry 4.0—A systematic literature review. *International Journal of Information Management Data Insights*, *2*(2), Article 100084. https://doi.org/10.1016/j.jjimei.2022.100084

Wen, Y., & Zhu, Q. (2022). An improved network time protocol for industrial Internet of Things. *Sensors*, *22*(13), Article 5021. https://doi.org/10.3390/s22135021

Widoretno, S., Hadi, A., & Zuhair, A. (2025). *Implementasi pemilah barang paket ekspedisi menggunakan QR Code dengan ESP32 yang terintegrasi dengan Google Spreadsheet untuk manajemen data* [Implementation of expedition package sorting using QR Code with ESP32 integrated with Google Spreadsheet for data management]. *Jurnal Qua Teknika*, *15*(2), 119–124. https://doi.org/10.35457/quateknika.v15i02.5083

Winkelhaus, S., & Grosse, E. H. (2020). Logistics 4.0: A systematic review towards a new logistics system. *International Journal of Production Research*, *58*(1), 18–43. https://doi.org/10.1080/00207543.2019.1612964