International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

Low-Cost Cloud-Integrated Inventory Monitoring System Using ESP32 and Google AppSheet

Dr. J. Praveena[1], A. Sai Samhitha[2],G. Keerthana Sri[3]

[1]Assistant Professor, Department of Mechanical Engineering, Andhra University College of Engineering for Women, Visakhapatnam
[2,3]Final Year Students, Department of Mechanical Engineering, Andhra University College of Engineering for Women, Visakhapatnam

Abstract
This paper presents the design and implementation of a low-cost, cloud-integrated inventory monitoring system developed as a final year engineering project. The system is built around an ESP32 microcontroller connected to an MFRC522 RFID reader module. When an RFID tag is scanned, the ESP32 sends the data over Wi-Fi to a Google Apps Script web application, which updates a Google Sheets database and refreshes an AppSheet mobile dashboard in near real time. The entire prototype was assembled for approximately USD 15-20 in hardware, with no recurring software costs, as all software tools used are freely available. The system is intended primarily for small-scale warehouses and educational prototype applications where affordable, accessible inventory tracking is needed. Testing confirmed that stock records were updated within 2 to 3 seconds of a tag scan, the AppSheet dashboard reflected correct quantities across all test items, and no spurious decrements occurred after implementing a simple debounce routine in the firmware. The paper describes the full hardware setup, cloud architecture, workflow design, and dashboard configuration, and it discusses both the practical strengths and the known limitations of the current prototype.

Keywords: ESP32; RFID; IoT; Inventory Management; Google Sheets; Google Apps Script; AppSheet; Cloud Integration; Small-Scale Industry

1. INTRODUCTION

Managing stock in a small warehouse or a college laboratory can quickly become difficult if done manually. Paper registers, periodic physical counts, and spreadsheets updated at the end of the day all suffer from the same problem: the records lag behind physical reality. A warehouse operator looking at yesterday's spreadsheet does not know whether a particular item was already dispatched this morning. These delays lead to either over-ordering, which ties up money in excess stock, or stock-outs, which interrupt production or distribution.

Barcode-based systems improve the situation somewhat by reducing transcription errors, but they still require someone to scan each item individually and maintain consistent scanning discipline. In small operations with limited staff, that discipline tends to break down during busy periods.

Radio Frequency Identification (RFID) offers a step up in automation. A passive RFID reader can register

a tagged item passing within its detection range without requiring the operator to aim a scanner or hold a steady line of sight. This makes stock movements easier to record consistently. In recent years, low-cost microcontrollers such as the ESP32, combined with free cloud platforms like Google Sheets and AppSheet, have made it realistic for small organisations and student teams to build functional RFID-based systems without enterprise-level hardware or software budgets.

The system presented in this paper was developed as a final-year project by mechanical engineering students at the Andhra University College of Engineering for Women, Visakhapatnam. The project goal was straightforward: build a working inventory monitoring prototype that updates a cloud dashboard in near real time when an RFID tag is scanned, costs no more than a small project budget, and can be understood and operated by non-technical staff. The project does not claim to replace enterprise warehouse management systems, nor does it involve machine learning, AI forecasting, or industrial-scale deployment. It is an educational prototype with practical application potential for small-scale warehouses.

2. LITERATURE REVIEW

A brief review of relevant work helps to position this project within existing research. The references below are grouped into three areas: traditional inventory methods, RFID-based identification systems, and IoT-enabled cloud monitoring.

2.1 Inventory Management Methods

The foundational work in inventory theory dates to Harris [1], who formulated the Economic Order Quantity model in 1913 to determine the optimal replenishment batch size. Silver, Pyke, and Peterson [2] extended this into a comprehensive framework covering safety stock, reorder points, and multi-item control. Nahmias [3] provided accessible textbook treatments of these methods that remain in use in engineering education. The common observation across this literature is that analytical models are only as useful as the data fed into them. Manual stock records rarely provide consistent or timely data.

DeHoratius and Raman [4] studied inventory record accuracy empirically across retail stores and found that a significant fraction of records deviated from actual physical counts by amounts large enough to cause incorrect replenishment decisions. Their findings support the case for automated data capture.

2.2 RFID in Inventory and Supply Chain

Rekik [5] reviewed RFID applications in inventory management and found that the technology reduces stock record errors most significantly in environments where manual tracking was previously error-prone. Wang, Wang, and Wang [6] demonstrated that placing RFID readers at entry and exit points of a warehouse, rather than throughout the floor, offers a practical balance between coverage and cost. This gate-point approach was adopted in the present system. Finkenzeller [7] provides technical background on RFID hardware, including the ISO 14443A protocol used by the MFRC522 module in this project.

Cammarano, Marra, and Michelino [8] examined RFID adoption in small and medium enterprises and identified two dominant barriers: perceived cost and lack of in-house technical capability. The present system was designed with both barriers in mind, using low-cost hardware and free software tools that a non-specialist operator can learn to use.

2.3 IoT and Cloud-Based Monitoring

Ben-Daya, Hassini, and Bahroun [9] surveyed IoT applications across supply chains and noted that real-time connectivity between physical assets and cloud-based records improves responsiveness to demand changes. Rejeb, Rejeb, and Simske [10] reviewed IoT inventory systems specifically and found that cloud integration, mobile access, and no-code platforms were the key enablers for adoption in resource-limited

organisations. Sahoo and Pati [11] built a prototype combining RFID and IoT principles for warehouse monitoring, with design priorities similar to this project: real-time visibility, remote access, and low hardware cost. Sharma and Gupta [12] demonstrated that ESP32-based RFID systems can reliably update cloud records in a small warehouse setting with acceptable latency.

Based on earlier studies on RFID and IoT inventory systems, this project combines RFID scanning, ESP32 communication, Google Sheets as the database, Google Apps Script as middleware, and AppSheet as a no-code dashboard into a single integrated prototype that a small team of students can build, test, and document within an academic project timeline.

3. PROPOSED SYSTEM

3.1 System Overview

The developed system connects an RFID scanning setup to a cloud-based inventory database and a mobile dashboard. At the physical layer, an MFRC522 RFID reader module is connected to an ESP32 development board. When a tagged item is brought within the read range of the antenna, the reader captures the tag's unique identifier (UID). The ESP32 then sends this UID over Wi-Fi to a Google Apps Script web application deployed in the cloud. The script looks up the UID in a Google Sheets spreadsheet, updates the corresponding stock quantity, and records the scan event. AppSheet, which connects directly to Google Sheets, reflects the updated data on a mobile-accessible dashboard within a few seconds.

Fig. 1 illustrates the four functional layers of the system: the physical RFID hardware layer, the ESP32 edge processing layer, the Google Cloud data layer, and the AppSheet dashboard layer.

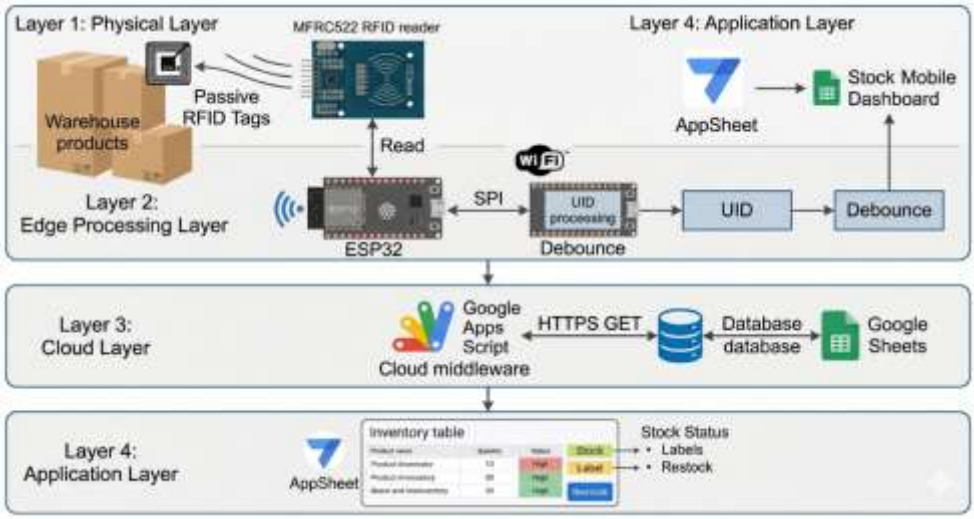

Fig. 1. Four-layer architecture of the proposed cloud-integrated inventory monitoring system.

3.2 Design Objectives

The system was designed with the following practical goals in mind: (1) keep total hardware cost within a small project budget; (2) avoid any software licensing fees; (3) allow a non-technical operator to use the system after minimal training; (4) update stock records automatically whenever an item passes the scanner, without requiring manual entry; and (5) make the inventory dashboard accessible on a smartphone from anywhere with internet access.

4. HARDWARE SETUP

4.1 Components Used

The hardware for this prototype consists of four main components. The ESP32 development board serves as the central processing and communication unit. It is a dual-core microcontroller with built-in Wi-Fi and Bluetooth, runs at up to 240 MHz, and costs under USD 5. The MFRC522 is a passive RFID reader module operating at 13.56 MHz, compatible with MIFARE 1K tags conforming to ISO 14443A. It connects to the ESP32 through a standard 4-wire SPI bus. MIFARE 1K passive tags are used as item identifiers; each tag carries a factory-assigned 4-byte unique identifier (UID). An LED indicator connected to a GPIO pin of the ESP32 provides visual feedback to the operator after each successful scan. The system also uses a standard Wi-Fi router available in the laboratory or warehouse environment.

Table I lists the components and their approximate costs.

Component	Specification	Approx. Cost (INR)
ESP32 Dev Board	Xtensa LX6, Wi-Fi + BT	₹250 - ₹420
MFRC522 RFID Reader	13.56 MHz, SPI interface	₹125 - ₹210
MIFARE 1K RFID Tags (x10)	ISO 14443A, 4-byte UID	₹250 - ₹420
LED + Resistor	Indicator circuit	₹15 - ₹40
Jumper Wires + Breadboard	Prototyping	₹165 - ₹335
Total		~₹1,250 – ₹1,675

Table I. Hardware Components and Cost Estimate

4.2 Wiring Configuration

The MFRC522 module connects to the ESP32 via the SPI bus using the following pin assignments: SDA to GPIO 5, SCK to GPIO 18, MOSI to GPIO 23, MISO to GPIO 19, and RST to GPIO 22. The 3.3V and GND pins of the MFRC522 are connected to the corresponding power pins of the ESP32. The LED anode connects through a 330-ohm resistor to GPIO 2. Fig. 2 shows the wiring arrangement on a breadboard.

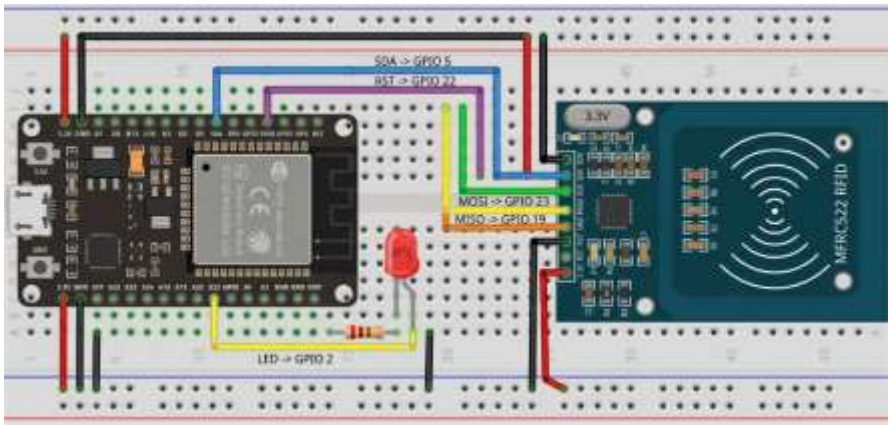

Fig. 2. Hardware wiring of the ESP32 and MFRC522 RFID reader module on a breadboard.

4.3 Read Range and Tag Behaviour

The MFRC522 has an effective read range of approximately 2 to 5 cm for MIFARE 1K tags under typical conditions. This limited range is intentional for controlled scanning station setups where the operator brings each item to the reader. The tags are passive, meaning they draw power from the reader's radio field and require no batteries. Each tag UID is unique and fixed at the factory and is used as the sole identifier in this system. No data is written to the tag memory; all product associations are handled in the Google Sheets database.

5. CLOUD ARCHITECTURE

5.1 Google Sheets as the Database

Google Sheets was selected as the data storage layer because it is free, accessible via a web browser or mobile app, familiar to most users, and supports automatic formula recalculation. The spreadsheet contains three worksheets. The Inventory Master sheet holds one row per product with the following columns: Product ID, Product Name, RFID UID, Current Stock Quantity, Minimum Stock Level, and Stock Status. The Stock Status column uses a formula that automatically labels each item as In-Stock, Low Stock, or Out-of-Stock based on the current quantity relative to the minimum level. The Scan Log sheet records every validated RFID scan event, including the timestamp, UID, matched product name, and the resulting stock quantity. The Restock Log sheet records manual restock entries made through the AppSheet interface.

This structure keeps the database simple and auditable. Because the Scan Log is append-only and never modified after writing, it provides a reliable record of all item movements during operation.

5.2 Google Apps Script as Middleware

Google Apps Script is a JavaScript-based scripting platform built into the Google Workspace ecosystem. It is used here to create a web application that accepts HTTP GET requests from the ESP32, processes them, and writes the results to Google Sheets. The script is deployed as a web app with the URL shared with the ESP32 firmware. No server is required; Google hosts and runs the script automatically.

The script's doGet(e) handler performs the following steps when a request arrives: it reads the UID query parameter from the request URL; it searches the Inventory Master sheet for a row with a matching UID; if a match is found, it decrements the stock quantity by one and updates the Stock Status formula; it appends a new row to the Scan Log with the current timestamp, the UID, the matched product name, and the updated quantity; and it returns a JSON response indicating success or an error if the UID was not recognized. Unrecognised UIDs are silently rejected without modifying any inventory data, which prevents unregistered or stray tags from affecting the database.

5.3 AppSheet Dashboard

AppSheet is a no-code platform from Google that can build mobile and web applications directly on top of Google Sheets. No programming is required to configure it. The dashboard in this project was created by pointing AppSheet at the Inventory Master sheet and defining two views: a table view showing all products with their current quantities and colour-coded status labels, and a detail view that opens when the user taps a product row and shows the full product record along with a Restock button.

The colour coding in the table view is practically useful: green means the item is adequately stocked, yellow means it is running low, and red means it is out of stock. An operator can glance at the dashboard on a phone and immediately identify which items need attention. The Restock button in the detail view

allows the operator to record a restocking action, which updates the quantity and simultaneously writes an entry to the Restock Log.

6. SYSTEM WORKFLOW

Fig. 3 shows the end-to-end workflow of the system from tag scan to dashboard update.

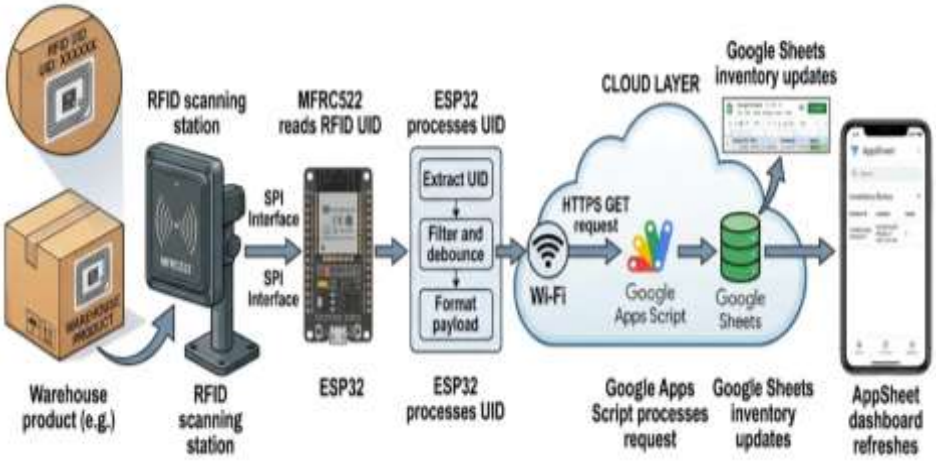

Fig. 3. End-to-end workflow from RFID tag scan to AppSheet dashboard update.

The workflow operates as follows. When a tagged item is brought to the scanning station, the MFRC522 reader detects the tag and reads its UID through the ISO 14443A anticollision protocol. The ESP32 firmware checks whether the same UID was processed within the previous 2,000 milliseconds. If it were, the detection is discarded to prevent duplicate decrements. If not, the UID is packaged as a query parameter in an HTTPS GET request and sent over Wi-Fi to the Apps Script web application URL.

The Apps Script handler receives the request, looks up the UID, performs the stock update, and returns a JSON response. The ESP32 parses the response: if it indicates success, the indicator LED blinks once to confirm the scan was recorded; if it indicates an unrecognised UID, the LED blinks twice to alert the operator. The Google Sheets formulas recalculate immediately when the stock quantity changes, and AppSheet polls the data source and refreshes the dashboard, completing the cycle in 2 to 3 seconds under normal Wi-Fi conditions.

7. ESP32 FIRMWARE

The firmware was written in the Arduino IDE using the ESP32 Arduino core, the MFRC522 library, and the standard ESP32 Wi-Fi and HTTPS Redirect libraries. The main loop runs a polling check every 50 milliseconds to detect whether a new tag is present within the reader's range. When the MFRC522 library signals a new card, the firmware reads the 4-byte UID and converts it to an 8-character hexadecimal string. Before sending the UID to the cloud, the firmware checks a locally stored timestamp. If the same UID was last processed less than 2,000 milliseconds ago, the event is discarded. This debounce window is important because briefly keeping the tag near the antenna may otherwise produce multiple scan events and decrement inventory by the same amount. With the 2-second debounce active, each physical placement of a tag near the reader produces exactly one decrement, regardless of how long the tag remains in range.

The HTTPS GET request is built by appending the UID as a query parameter to the Apps Script web app URL. The ESP32's built-in TLS stack handles the secure connection, so no additional cryptography libraries are needed. After sending the request, the firmware waits for a response, reads the JSON status field, and sets the LED accordingly.
Fig. 4 illustrates the firmware flowchart.

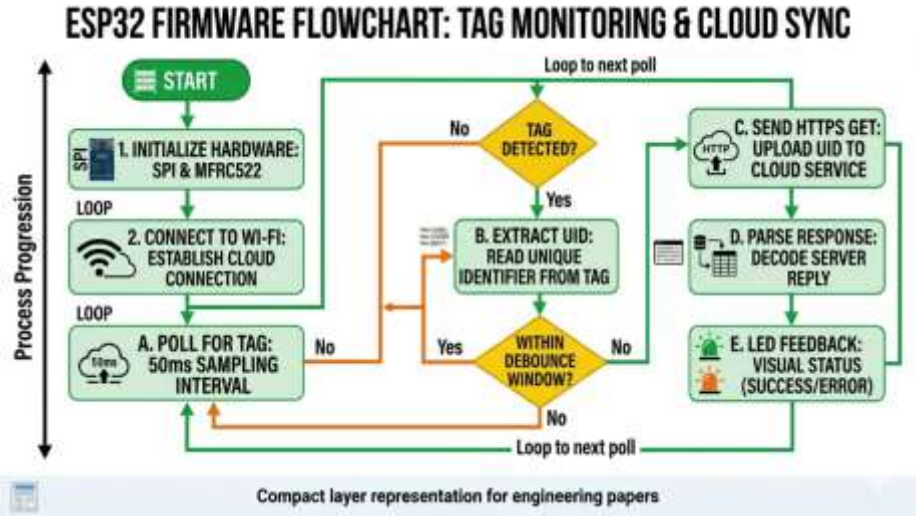

Fig. 4. ESP32 firmware logic flowchart.

8. DASHBOARD DESIGN

The AppSheet dashboard was configured without writing any front-end code. After authorising AppSheet to read the Google Sheets file, the table and detail views were created through the AppSheet editor by selecting columns, assigning display names, and setting conditional formatting rules for the Stock Status column.
The main table view displays Product ID, Product Name, Current Quantity, and Stock Status for every item. The Stock Status column is colour-coded using AppSheet conditional formatting: the cell background shows green when stock is above the minimum level, yellow when stock equals the minimum level, and red when stock falls to zero. This formatting makes the dashboard functional even for operators who are unfamiliar with reading numbers, as the colour state gives an immediate visual signal.
The product detail view shows all columns for a selected item plus a Restock button. Tapping the button opens a simple form where the operator enters the quantity being added. Submitting the form triggers an AppSheet automation that adds the entered quantity to the current stock value and writes a timestamped record to the Restock Log sheet. This closes the loop on the data flow: outgoing stock is decremented automatically by RFID scans, while incoming stock is incremented manually through the dashboard, and both movements are permanently logged.
Fig. 5 shows screenshots of the main table view and the product detail view of the AppSheet dashboard.

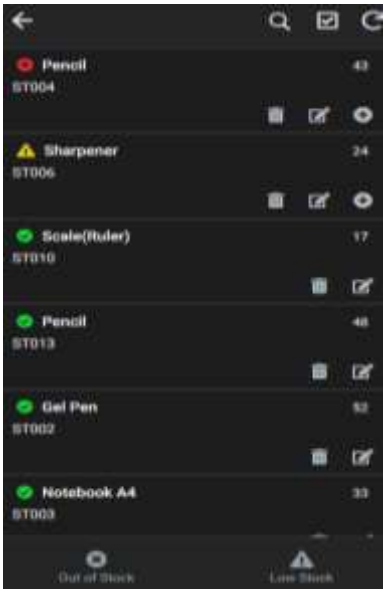

Fig. 5. AppSheet inventory dashboard: main table view (left) and product detail view with Restock button (right).

9. DATA FLOW SUMMARY

Table II summarises the data flow across the four layers of the system, from physical tag detection to dashboard display.

Layer	Component	Action	Output
Physical	RFID Tag + MFRC522	Tag brought within range; UID read via ISO 14443A	4-byte UID
Edge	ESP32	Debounce check; build HTTPS GET request	HTTP request to Apps Script
Cloud	Google Apps Script + Sheets	UID lookup; stock decrement; scan log append	JSON response; updated sheet
Application	AppSheet	Poll Sheets; refresh dashboard views	Updated mobile dashboard

Table II. Data Flow Across System Layers

10. SYSTEM IMPLEMENTATION

10.1 Development Environment

The ESP32 firmware was developed in the Arduino IDE version 2.x with the ESP32 Arduino core installed through the Boards Manager. The MFRC522 library was installed through the Arduino Library Manager. The Google Apps Script was written in the online Apps Script editor accessible from within Google Drive and was deployed as a web application with anonymous execution permissions so that the ESP32 can call it without authentication headers.

The Google Sheets spreadsheet was set up manually before testing, with product data entered for fifteen

items used in the test run. RFID UIDs were registered by holding each tag to the reader and reading the printed output in the Arduino Serial Monitor, then copying the UID string into the corresponding row of the Inventory Master sheet. This one-time registration step takes approximately one minute per item and requires no special tools.

10.2 Testing Procedure

Testing was conducted in two phases. In the first phase, each of the fifteen RFID tags was scanned individually in sequence, and the Serial Monitor output was checked to confirm that the correct UID was read. The Scan Log sheet was checked after each scan to confirm a new row was appended with the correct product name, timestamp, and decremented quantity. In the second phase, a tag was held against the reader for five seconds to verify the debounce behaviour. Without the debounce check, five to seven rows appeared in the Scan Log from a single physical placement. After adding the 2,000-millisecond debounce window to the firmware, repeated tests consistently produced exactly one Scan Log entry per tag placement.

The AppSheet dashboard was monitored on a smartphone during the scanning tests to confirm that the table view updated correctly within the expected time window after each scan.

11. RESULTS AND DISCUSSION

11.1 Latency

End-to-end latency was measured as the time from tag detection at the MFRC522 antenna to the updated stock count appearing on the AppSheet dashboard. This was measured informally using a stopwatch during multiple test scans. Under stable Wi-Fi conditions in a laboratory environment with a reliable internet connection, the update consistently appeared on the dashboard within 2 to 3 seconds of tag detection. For small-scale inventory monitoring, this delay was considered acceptable during testing; no practical decision in a small warehouse requires knowing that an item was dispatched within the last 3 seconds.

Network-related delays were occasionally observed when the Wi-Fi signal was weak, with latency rising to 5 to 6 seconds in such cases. This is a known limitation of Wi-Fi-dependent IoT deployments and is discussed further in the limitations section.

11.2 Identification Accuracy

RFID identification accuracy was 100% across all test scans for tags presented within the effective 2 to 5 cm read range of the MFRC522. No misidentifications occurred during the test session. Tags presented at steep oblique angles occasionally required repositioning before the reader detected them, but this was resolved by orienting the tag more parallel to the reader antenna.

The Apps Script UID validation correctly rejected all test scans of unregistered tags, returning an error response without modifying the Inventory Master sheet.

11.3 Debounce Performance

The 2,000-millisecond debounce window reliably prevented duplicate scan events. In all tests where a tag was held near the reader for more than one second, exactly one Scan Log entry was produced, and exactly one unit was decremented from the stock count. This result confirmed that debouncing is a correctness requirement rather than an optional refinement.

11.4 Comparison with Manual Tracking

The operational advantage of the system over manual tracking is the immediacy of updates. In a manual register, a stock movement is recorded when someone chooses to write it down, which may be minutes or

hours after the physical event. In the RFID system, the Scan Log record is written within 3 seconds of the item passing the scanner. For a small warehouse running this system throughout the working day, the Inventory Master sheet remains representative of actual stock levels in near real time, rather than reflecting the state of affairs as of the last manual update.

Table III summarises the key performance results from the testing phase.

Performance Metric	Observed Result
End-to-end update latency (stable Wi-Fi)	2 to 3 seconds
RFID identification accuracy (within range)	100%
False positives (unregistered tag scans)	0
Debounce effectiveness	Single decrement per tag placement
Dashboard refresh after scan	Visible within 2-3 seconds
Total hardware cost	Approx. ₹1,250-₹1,700
Recurring software cost	₹0

Table III. Summary of System Performance Results

12. ADVANTAGES

The following practical strengths of the proposed system are worth highlighting.

- Cost: The total hardware cost of the prototype was approximately ₹1,250–₹1,700, and all software components are available free of charge. This places the system within reach of small-scale organisations with limited budgets.
- No specialist software skills required: Google Sheets, AppSheet, and Google Apps Script can all be configured by someone with basic spreadsheet literacy. No front-end programming, database administration, or server management is required.
- Real-time visibility: Stock records update within 2 to 3 seconds of a tag scan, keeping the database representative of the current warehouse state throughout the working day.
- Audit trail: The Scan Log is append-only and stores a timestamped record of every RFID-detected dispatch event, which supports post-session review and accountability.
- Remote access: The AppSheet dashboard is accessible from any smartphone or browser with an internet connection, so a supervisor does not need to be physically present at the scanning station to check stock levels.
- Tag reuse: Because product information is stored in Google Sheets rather than in tag memory, the same physical tag can be reassigned to a different product by editing a single spreadsheet cell. This eliminates recurring tag costs.
- Scalability within its scope: Additional products can be registered simply by entering new rows in the Inventory Master sheet and assigning UID values. No firmware changes or hardware additions are needed for inventory expansion.

13. LIMITATIONS

Several limitations of the current prototype are acknowledged below.

- Wi-Fi dependency: The system has no local data buffering. If the Wi-Fi connection is interrupted, scan events during the outage are lost permanently. The ESP32 firmware retries the HTTP request once, but sustained outages will result in missed records.
- Read range: The MFRC522 module has an effective read range of 2 to 5 cm. Each item must be individually presented at the scanning station. The system cannot read multiple tags simultaneously or track items as they move freely around a warehouse floor.
- Single-reader architecture: Google Sheets does not support concurrent write transactions. Running multiple ESP32 readers updating the same sheet simultaneously could cause race conditions in the Apps Script handler, producing incorrect quantity values.
- Manual demand history: There is no automated forecasting or demand analytics in the current system. Reorder decisions rely on the operator reading the AppSheet dashboard and placing orders manually.
- No batch traceability: The system tracks aggregate quantities by product, not individual item serial numbers. It is not suitable for applications requiring batch-level or unit-level traceability.
- Google Sheets performance at scale: Google Sheets performs well for inventories with a few hundred rows, but its performance degrades with very large datasets. It is not a replacement for a dedicated database for high-volume operations.

14. CONCLUSION

This paper has described the design, implementation, and testing of a low-cost, cloud-integrated inventory monitoring system built around an ESP32 microcontroller, an MFRC522 RFID reader, Google Sheets, Google Apps Script, and AppSheet. The system was developed as a final year engineering project and is intended for application in small-scale warehouses and educational prototype contexts.

The prototype demonstrated that RFID-based stock tracking with near-real-time cloud updates is achievable at a total hardware cost of approximately ₹1,250–₹1,700 with no software licensing fees. Testing confirmed that stock records were updated within 2 to 3 seconds of a tag scan under normal Wi-Fi conditions, RFID identification accuracy was 100% within the effective read range, and the firmware debounce routine successfully prevented duplicate inventory decrements.

The project successfully demonstrated that a low-cost IoT-based inventory monitoring system can be developed using affordable hardware and freely available cloud tools. It also demonstrated the practical use of Google Apps Script as middleware between embedded hardware and a cloud spreadsheet database, along with a no-code AppSheet dashboard that can be configured without specialist software skills between embedded hardware and a cloud spreadsheet database, and a no-code AppSheet dashboard that can be configured and operated without specialist software skills. The system addresses the most commonly cited barriers to RFID adoption in small organisations, namely cost and technical complexity, while delivering the core benefit of automated, real-time stock record updates.

15. FUTURE SCOPE

Several improvements could be made to extend the system beyond its current prototype state. Offline event buffering using the ESP32's SPIFFS flash memory would allow scan records to be stored locally during Wi-Fi outages and uploaded to Google Sheets once connectivity is restored, addressing the most critical reliability gap. Replacing the MFRC522 with a UHF RFID reader operating in the 860 to 960 MHz

band would extend the effective read range to several meters, enabling gate-level scanning where tagged items are automatically detected as they pass a warehouse entry or exit point without being individually presented.

On the cloud side, replacing Google Sheets with a lightweight database service such as Firebase Realtime Database would remove the transaction-locking constraint, allowing multiple ESP32 readers to operate simultaneously without race conditions. Automated monthly aggregation of the Scan Log into a demand history table would support simple moving average forecasting without manual effort from the supervisor. From an application standpoint, adding email or SMS alerts when stock falls below the minimum level would improve the responsiveness of the replenishment workflow. Integrating a supplier contact list into the AppSheet dashboard could allow the operator to send a reorder message directly from the dashboard with a single tap. These extensions remain realistic within a student project scope and would meaningfully improve the system's practical utility for small-scale commercial deployment.

REFERENCES

1. F. W. Harris, "How many parts to make at once," Factory: The Magazine of Management, vol. 10, no. 2, pp. 135-136, 1913.
2. E. A. Silver, D. F. Pyke, and R. Peterson, Inventory Management and Production Planning and Scheduling, 3rd ed. New York, NY: Wiley, 1998.
3. S. Nahmias, Production and Operations Analysis, 6th ed. New York, NY: McGraw-Hill/Irwin, 2009.
4. N. DeHoratius and A. Raman, "Inventory record inaccuracy: An empirical analysis," Management Science, vol. 54, no. 4, pp. 627-641, 2008.
5. M. Rekik, "RFID in inventory management: A review," International Journal of Production Economics, vol. 112, no. 2, pp. 657-667, 2008.
6. L. Wang, G. Wang, and Y. Wang, "RFID-based inventory management system," IEEE Transactions on Automation Science and Engineering, vol. 6, no. 3, pp. 544-551, 2009.
7. K. Finkenzeller, RFID Handbook: Fundamentals and Applications in Contactless Smart Cards, Radio Frequency Identification and Near-Field Communication, 3rd ed. Chichester, UK: Wiley, 2010.
8. A. Cammarano, E. Marra, and M. Michelino, "Open innovation and RFID adoption in SMEs," Technovation, vol. 58-59, pp. 1-10, 2017.
9. M. Ben-Daya, E. Hassini, and Z. Bahroun, "Internet of Things and supply chain management: A literature review," International Journal of Production Research, vol. 57, no. 15-16, pp. 4719-4742, 2019.
10. A. Rejeb, K. Rejeb, and S. Simske, "IoT-based inventory management: A systematic review," IEEE Access, vol. 8, pp. 181345-181362, 2020.
11. S. K. Sahoo and B. B. Pati, "Smart inventory management using IoT," IEEE Access, vol. 9, pp. 102376-102385, 2021.
12. A. Sharma and V. Gupta, "RFID-based warehouse automation system for real-time stock tracking," International Journal of Engineering Research and Technology, vol. 10, no. 3, pp. 45-52, 2021.
13. B. Winkelhaus and E. H. Grosse, "Logistics 4.0: A systematic review towards a new logistics system," International Journal of Production Research, vol. 58, no. 1, pp. 18-43, 2020.
14. A. Sarac, N. Absi, and S. Dauzere-Peres, "A literature review on the impact of RFID technologies on supply chain management," International Journal of Production Economics, vol. 128, no. 1, pp. 77-95, 2010.