

Design of IoT-Based Monitoring System on Miniature Automated Storage and Retrieval System (ASRS) with Barcode-Based Item Identification Using ESP32

Amalia Khairun Nadia
Dept. Of Electrical Engineering
Universitas Negeri Semarang
Semarang, Indonesia
amalianadia@students.unnes.ac.id

Tatyantoro Andrasto, S.T., M.T.
Dept. of Electrical Engineering
Universitas Negeri Semarang
Semarang, Indonesia
tatyantoro@mail.unnes.ac.id

Mario Norman Syah, S.Pd., M.Eng.
Dept. of Electrical Engineering
Universitas Negeri Semarang
Semarang, Indonesia
marionormansyah@mail.unnes.ac.id

Abstract—This study aims to design and develop a monitoring system for a miniature Automated Storage and Retrieval System (ASRS) based on Internet of Things (IoT) with barcode-based item identification using the ESP32 microcontroller. The system integrates the GM65-S barcode scanner, two ESP32 units as input and robot arm controllers, a MySQL database with 11 relational tables, and a web monitoring dashboard (SmartASRS) accessible at <https://smartasrs.my.id>. The research adopted the Research and Development (R&D) method with the Waterfall model. Testing was performed using Black Box Testing on all implemented features. Results showed a 100% success rate across 42 web monitoring test scenarios spanning 6 modules, and 100% across 81 database test scenarios spanning 11 tables. The system supports three item retrieval modes: FIFO (First In First Out), LIFO (Last In First Out), and FEFO (First Expired First Out). All features including real-time storage grid monitoring, barcode identification, robot progress tracking, and activity history functioned as designed. The SmartASRS system demonstrates that an integrated IoT-based ASRS monitoring solution can be effectively realized on a miniature scale as both a functional warehouse management system and an industrial automation learning medium.

Keywords—ASRS, IoT, ESP32, Barcode, MySQL, Monitoring, Web Dashboard

I. INTRODUCTION

The development of automation technology and information systems in the industrial sector, particularly in logistics and warehousing, has been advancing rapidly in recent years [1]. The increasing volume of goods along with demands for faster distribution drive the need for storage systems that can operate efficiently, accurately, and with comprehensive real-time monitoring capabilities [2]. In practice, many warehouse systems still rely on manual recording and direct handling by operators, which can lead to recording errors, process delays, and limitations in comprehensively monitoring the condition of goods.

One approach widely developed to improve warehouse efficiency is the implementation of Automated Storage and Retrieval Systems (ASRS) [3]. This system enables the storage and retrieval of goods to be performed automatically and in a structured manner [9]. However, the success of ASRS implementation depends not only on the mechanical system and robot movements, but equally on the monitoring system's ability to provide accurate and continuous information about item movements and system conditions.

As the Internet of Things (IoT) concept continues to evolve, integration between physical devices and digital systems has become increasingly feasible [4]. Through IoT, devices such as sensors, barcode readers, and microcontrollers can interconnect over networks so that data can be transmitted and displayed in real-time [5]. This capability opens opportunities to build monitoring systems that allow users to observe system conditions without being physically present at the location.

In automated warehouse systems, item identification is a critical aspect for ensuring data traceability. Barcode technology is widely used because it is relatively simple, highly accurate, and easily integrated with digital systems [6]. By reading barcodes, each item can be uniquely identified, helping to reduce recording errors and accelerate data management processes [8]. The ESP32 microcontroller is a popular choice due to its integrated Wi-Fi connectivity and adequate processing power [4]. In monitoring systems, ESP32 acts as the bridge that receives data from barcode readers and transmits it to the server for real-time updates [10].

Several prior studies have addressed related problems. Khan et al. [2] developed a Smart Warehouse Management System with IoT-based real-time monitoring, but focused on large-scale infrastructure without addressing miniature-scale integration for educational purposes. Sahara and Aamer [5] investigated real-time data integration in IoT-based smart warehouses, yet did not incorporate physical barcode identification with robot command feedback. Pratama et al. [7] implemented an ESP32-based barcode scanning system for warehouse stock opname, but without integration into a full ASRS framework including slot management, multi-method retrieval, and robot progress monitoring. Anisa and Juhri [6] built a web-based barcode-integrated warehouse management system; however, their approach lacked physical ASRS hardware and real-time robot control. Existing ASRS research typically operates at industrial scale with high investment costs [1][3], limiting accessibility for research and educational deployment.

This research addresses these gaps by designing SmartASRS a miniature ASRS monitoring system that uniquely integrates physical barcode scanning via GM65-S and ESP32, a relational MySQL database with 11 purpose-built tables, three retrieval methods (FIFO/LIFO/FEFO), real-time robot progress tracking across 13 operation stages, and a fully functional web monitoring dashboard into a single, cohesive IoT-based

platform. The novelty of this work lies in three aspects: (1) the end-to-end integration of barcode identification, ESP32 communication, centralized database management, and web monitoring into one unified miniature ASRS system; (2) the quantitative performance analysis of robot mission timing and network delay across 113 real missions; and (3) the identification of Wi-Fi-induced communication anomalies and their practical impact on monitoring accuracy, which has not been reported in prior miniature ASRS studies [7][8].

II. METHODS

This research adopts the Research and Development (R&D) approach with the Waterfall software development model. The Waterfall model was selected because the system requirements are well-defined from the outset and each development phase can be completed sequentially: requirements analysis, system design, implementation, testing, and evaluation. This structured approach ensures that each component hardware, software, and database is developed and verified systematically before system integration.

A. System Block Diagram

Fig. 1. System Block Diagram of SmartASRS Input-Process-Output (IPO) architecture showing the integration of barcode scanner, ESP32, MySQL database, and web dashboard.

The input variables consist of six main elements: (1) Item data including name, type, weight, and expiry date entered by the user through the web monitoring; (2) Barcode/QR Code read by the GM65-S scanner when items pass the scanning area on the conveyor; (3) Storage slot data covering slot identity (A1 to C3), robot coordinates (X, Y, Z), and slot status (filled/empty); (4) Operation mode (FIFO/LIFO/FEFO) set by the user to determine item retrieval priority; (5) Store/retrieve commands sent by the user; and (6) User commands via the web dashboard.

The processing layer consists of four main components: ESP32-ESP1 as the input controller that reads barcodes via GM65-S and sends data to the server via Wi-Fi/HTTP; ESP32-ESP2 as the robot arm controller that receives commands from ESP1 via ESP-NOW protocol and controls robot motor movements; the MySQL server (<https://smartasrs.my.id>) as the backend that processes and stores all system data; and the SmartASRS web dashboard as the visualization and control interface.

The output variables include six types of information delivered to the user: ASRS status information (ESP32 connectivity, slot counts), item in/out records, activity history logs, real-time robot status (idle/processing/completed with step-by-step timeline), slot condition updates, and auto-generated QR code/barcode for each registered item. Each output is sourced from specific database tables and updated automatically with every system event.

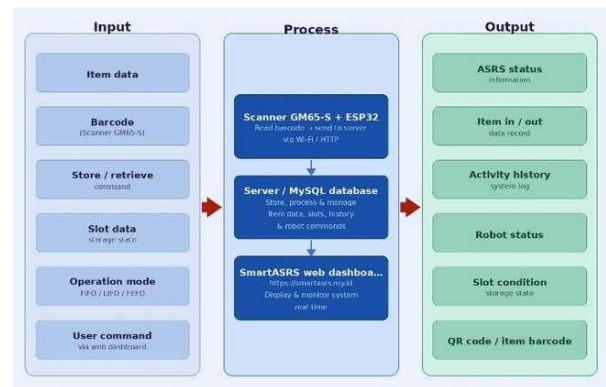


Fig. 1. System Block Diagram of SmartASRS

B. Hardware Design

The hardware system consists of a miniature ASRS as shown in Fig. 2, built as an integrated unit with several main subsystems working together to support automatic item storage and retrieval.

The storage rack is divided into nine slots arranged in a 3x3 grid (rows A, B, C and columns 1, 2, 3), giving slots A1 through C3. Each slot has specific robot navigation coordinates (X, Y, Z) stored in the database slot table. A robot manipulator placed at the front of the system performs item transfers between the conveyor and storage slots through horizontal and vertical movement along a rail mechanism, enabling precise placement controlled by the system.

The conveyor serves as the item entry and exit medium. During storage, it transports items to the robot work area; during retrieval, it serves as the item exit path. A proximity sensor detects item presence on the conveyor and acts as the trigger for the barcode scanning process. The GM65-S barcode scanner module reads the barcode automatically when an item passes the scanning area, communicating with ESP32 via UART serial (GPIO 16 RX2, GPIO 17 TX2). The trigger pin is connected to GPIO 4 for direct scan initiation.

Two ESP32 units serve as the main controllers. ESP32-ESP1 functions as the input controller: it reads barcodes from GM65-S, sends scan data to the server via HTTP over Wi-Fi, and periodically polls the server for pending robot commands. ESP32-ESP2 functions as the robot arm controller: it receives commands from ESP1 via ESP-NOW (a direct ESP-to-ESP wireless protocol without a Wi-Fi router) and drives the robot motors. A 12V supply is stepped down to 5V by an LM2596 buck converter to power the ESP32 units, scanner, and sensors.



Fig. 2. Physical Miniature ASRS Hardware

C. Flowchart System

Fig. 3. System Operational Flowchart complete workflow for item INPUT (barcode scan, slot assignment, robot PUT) and item OUTPUT (FIFO/LIFO/FEFO retrieval, robot TAKE).

For item input: (1) the user enters item data (name, type, weight, expiry date) through the web form; (2) the system validates the form empty required fields are rejected; (3) upon validation, a unique barcode is generated in PROD-[date]-[random] format using the phpqr code library and saved in the database; (4) the user prints the barcode and attaches it to the physical item; (5) the item is placed on the conveyor, where the proximity sensor detects its presence and triggers the conveyor; (6) the GM65-S scanner reads the barcode and sends the data to ESP32; (7) ESP32 transmits the scan data to the server via HTTP; (8) the server identifies the item from the database and determines an available slot; (9) a robot command is issued and ESP32-ESP2 executes the PUT operation; (10) the slot table, item table, history table, and log_aktivitas table are all updated automatically.

For item output: (1) the user selects a retrieval method FIFO (by earliest entry time), LIFO (by latest entry time), FEFO (by nearest expiry date), or Manual (by specific slot); (2) the system determines item retrieval priority accordingly; (3) the server sends a TAKE command to ESP32 via robot_commands table; (4) ESP32-ESP2 executes the retrieval operation; (5) the item exits via the conveyor; (6) slot status, item status, history, and log_aktivitas are all updated. Robot progress is reported back in stages from 5% to 100% and displayed in real-time on the Robot Progress monitoring page.

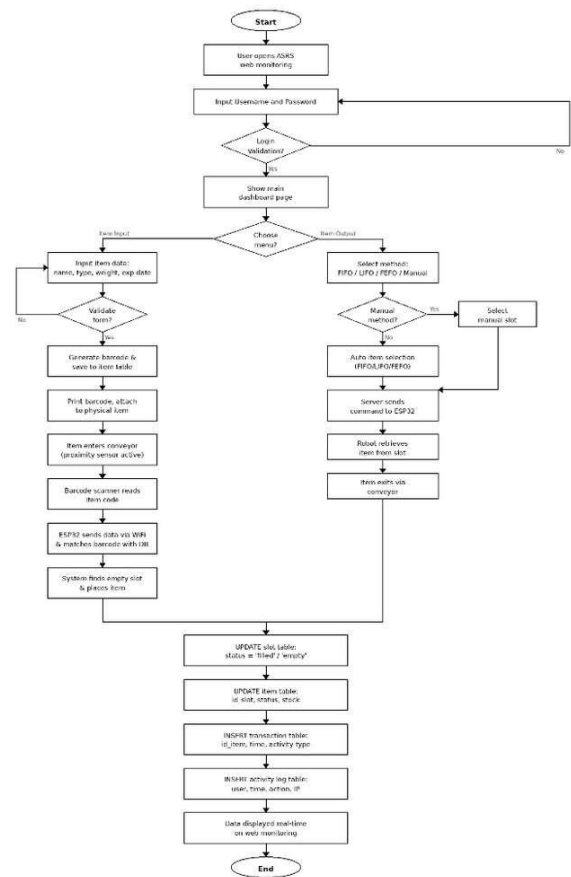


Fig. 3. System Operational Flowchart

D. Database Design

Fig. 4. Database Entity Relationship Diagram 11-table MySQL schema with barang as the central entity, linked to slot, history, users, robot_commands, output_queue, and 5 supporting tables.

The barang table is the main entity, storing all item information including name, type, weight, barcode, QR code path, expiry date, status, and storage position. It has direct relationships with: (1) slot stores the 9 rack locations A1-C3 with their X/Y/Z coordinates and status (filled/empty); (2) history records every I/O transaction with item identity, slot, action type (INPUT/OUTPUT), user, and timestamp; (3) users stores user accounts with bcrypt-hashed passwords; (4) robot_commands stores robot commands with their execution status; and (5) output_queue manages the retrieval queue with pending/processing/done status flow.

Five secondary tables serve supporting roles: log_aktivitas records all system activities as an audit trail (scanner, robot, and user actions); robot_progress stores step-by-step robot operation progress (5% to 100%); robot_session maintains active session data including current slot and barcode being processed; esp_status tracks ESP32 device connectivity via heartbeat timestamps; and settings stores system configuration including the active storage mode (FIFO/LIFO/FEFO). This relational structure ensures data integrity, supports real-time updates, and enables comprehensive activity traceability.

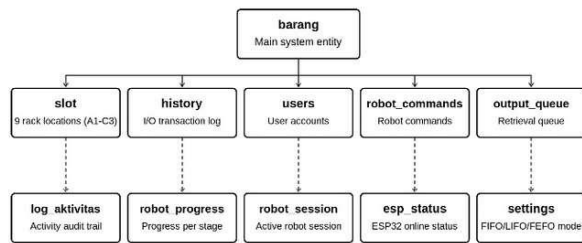


Fig. 4. Database Entity Relationship Diagram

E. Web Dashboard Design

Fig. 5. SmartASRS Web Dashboard Site Map navigation structure from Login page to five main pages: Dashboard, Input Barang, Output Barang, Robot Progress, and History.

The Login page is the entry point for user authentication. After successful login, users can navigate to five main pages: (1) Dashboard Awal the system home page displaying the 3x3 storage grid with real-time slot status, ESP32 device status, operation mode panel, and I/O control buttons; (2) Halaman Input Barang the item registration page for entering new items and generating barcodes; (3) Halaman Output Barang the item retrieval page with FIFO/LIFO/FEFO method selection and manual slot picking; (4) Halaman Robot Progress real-time robot operation monitoring with progress bar, stage timeline, and live activity feed; (5) Halaman History the complete transaction log with filtering by activity type and statistical summary cards. All pages share a unified navigation bar and communicate asynchronously with the backend API.

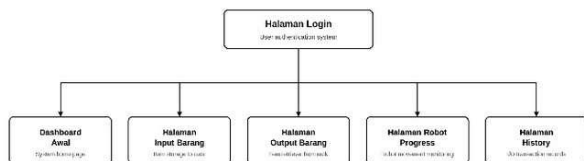


Fig. 5. SmartASRS Web Dashboard Site Map

F. Testing Method

System testing was conducted using the Black Box Testing method, which evaluates system behavior from the user perspective by focusing on input-output relationships without examining internal code logic [8]. This method was selected over White Box Testing for three reasons: (1) it validates end-user-facing functionality directly against the system requirements specification; (2) it is applicable regardless of the underlying implementation language or framework, making it appropriate for a multi-technology system spanning PHP, JavaScript, and MySQL; and (3) it covers all functional paths accessible to operators and administrators without requiring knowledge of internal code structure. Black Box Testing is widely adopted for validating web-based information systems in embedded IoT deployments [7]. Testing was applied across two domains: web monitoring functional testing on all six pages (42 scenarios), and database structural testing on all 11 tables (81 scenarios). Each test scenario specifies the

test action, expected output, actual output observed, and pass/fail status.

III. RESULT AND DISCUSSION

The SmartASRS system was successfully implemented and deployed at <https://smartasrs.my.id>. The web monitoring application integrates six main pages, a MySQL database with 11 tables, and two ESP32 devices operating as the scanner and robot arm controllers. This section presents the implementation results for each system component followed by detailed Black Box Testing results.

A. Web Monitoring Implementation

The SmartASRS web monitoring was successfully implemented at <https://smartasrs.my.id> using PHP, JavaScript, and MySQL. All six pages share a unified authentication system and real-time database access, ensuring data consistency across all views.

1) Login Page

Login Page (Fig. 6): Users authenticate via username and password validated against the users table using bcrypt hashing. Successful login creates a session and redirects to the dashboard; failed attempts display an error message.

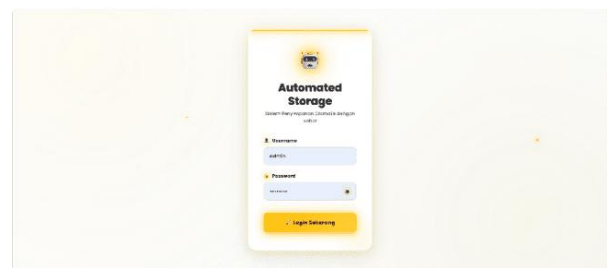


Fig. 6. Login Page - user authentication form with bcrypt-validated credentials and session management.

2) Dashboard Page

Dashboard Page (Fig. 7): The central monitoring hub displays a real-time 3x3 storage grid with slot status (filled/empty), item names, and stock counts. The right panel shows the operation mode selector (FIFO/LIFO/FEFO) and I/O control buttons. ESP32 device connectivity is updated every 300 seconds from the esp_status table.

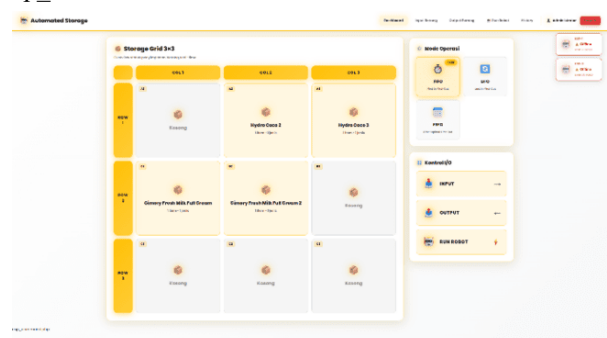


Fig. 7. Main Dashboard - real-time 3x3 storage grid visualization with operation mode selector (FIFO/LIFO/FEFO) and ESP32 connectivity status

3) Item Input Page

Item Input Page (Fig. 8): Users register new items by entering name, type, weight, and expiry date. After form validation, item data is saved to the barang table and a unique barcode (PROD-[date]-[random]) is auto-generated using the phpqrcode library for printing and physical attachment.

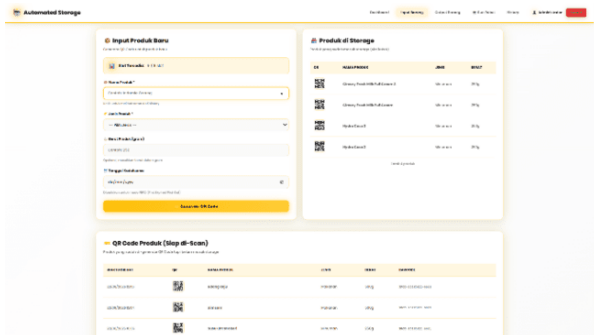


Fig. 8. Item Input Page - item registration form with automatic QR code/barcode generation (PROD-[date]-[random] format).

4) Item Output Page

Item Output Page (Fig. 9): Manages item retrieval using the active mode (FIFO, LIFO, or FEFO). The system automatically selects the appropriate item and generates a robot command in robot_commands table. Manual slot selection is also available. Slot status updates from filled to empty upon completion.

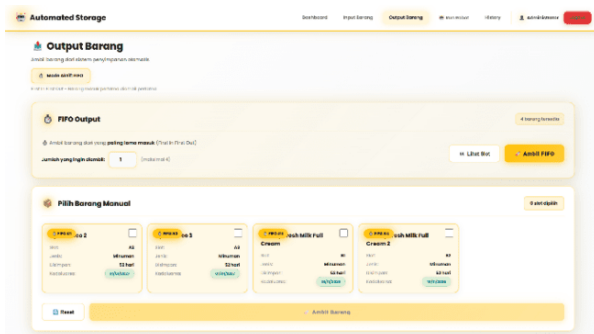


Fig. 9. Item Output Page - retrieval mode panel (FIFO/LIFO/FEFO/Manual) with automatic item prioritization and robot command generation.

5) Robot Progress Page

Robot Progress Page (Fig. 10): Provides real-time robot monitoring with a progress bar (5%-100%), stage-by-step operation timeline, and a live activity feed of the 10 most recent transactions. Data is sourced simultaneously from robot_commands, robot_progress, robot_session, and history tables.

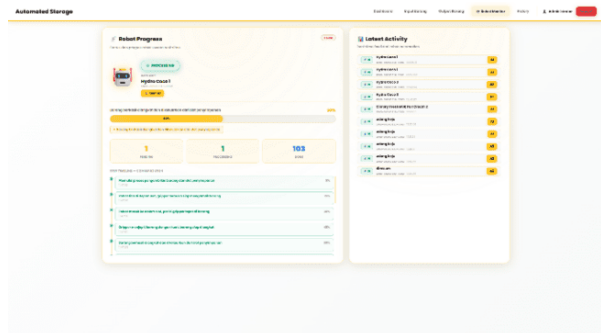


Fig. 10. Robot Progress Page - real-time progress bar (5%–100%), 13-stage operation timeline, and live I/O activity feed.

6) Activity History Page

Activity History Page (Fig. 11): Displays a complete transaction log with summary statistics (total: 100; items in: 44; items out: 56) and a filterable table (All/In/Out) showing item name, slot, action type, user, and timestamp for full I/O traceability.

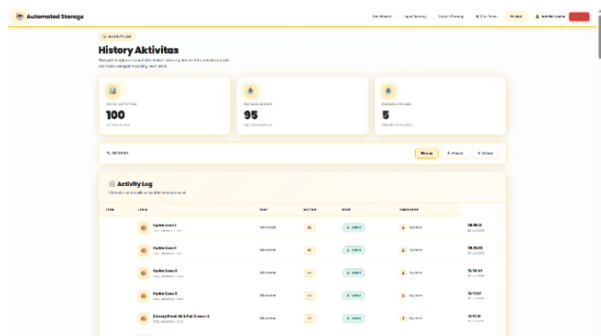


Fig. 11. Activity History Page - complete I/O transaction log with statistical summary cards and All/In/Out filter functionality.

B. Database Implementation

The MySQL database (smartasrs_automated_storage) was deployed on a shared hosting server. As shown in Fig. 12, all 11 tables were successfully created and populated, storing 2,894 total rows during the testing period. Table sizes range from 8.0 KiB (users) to 144.0 KiB (robot_progress), confirming sustained system activity throughout testing.

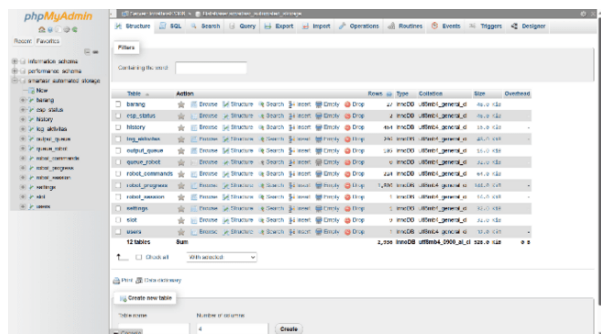


Fig. 12. MySQL Database Implementation (phpMyAdmin) view of 11 implemented tables

C. Web Monitoring Testing Results

Black Box Testing was conducted on all six web monitoring modules. Testing was performed by providing various inputs to each feature and observing the system output against expected behavior. The results for each module are presented below, followed by an overall recapitulation.

1) Login Page Testing

The Login page was tested with 5 scenarios covering valid login, invalid credentials, empty field submissions, and logout functionality. The testing verified that the authentication mechanism correctly accepts valid credentials, rejects invalid or incomplete inputs with appropriate error messages, and safely terminates user sessions upon logout.

TABLE I. TESTING RESULTS — LOGIN PAGE

No	Input / Test Action	Expected Output	Actual Output	Status
1	Enter valid username and password	System redirects to dashboard	Dashboard successfully displayed	Pass
2	Enter correct username, wrong password	System rejects login with error	Error message displayed correctly	Pass
3	Leave username field empty	System rejects login process	Login not processed, warning shown	Pass
4	Leave password field empty	System rejects login process	Login not processed, warning shown	Pass
5	Logout from system	System returns to login page	Login page successfully displayed	Pass

Based on the results in Table I, all 5 login scenarios passed successfully. The system correctly authenticates users, blocks unauthorized access, and properly ends sessions on logout. The bcrypt hashing implementation effectively protects credentials, as the system correctly validates hashed passwords without exposing raw data.

2) Dashboard Page Testing

The Dashboard page was tested with 7 scenarios verifying real-time data display accuracy, ESP32 device status monitoring, slot counting consistency, and dashboard refresh functionality. The testing confirmed that all statistics shown on the dashboard are dynamically sourced from the database and accurately reflect the actual system state.

TABLE II. TESTING RESULTS — DASHBOARD PAGE

No	Input / Test Action	Expected Output	Actual Output	Status
1	Open dashboard page	System statistics displayed	Statistics successfully displayed	Pass
2	Check item count displayed	Item count matches database	Data consistent with database records	Pass

No	Input / Test Action	Expected Output	Actual Output	Status
3	Check filled slot count	Filled slot data matches storage state	Data matches actual slot conditions	Pass
4	Check empty slot count	Empty slot information displayed	Empty slot count displayed correctly	Pass
5	Check ESP Scanner device status	Device online/offline status shown	Online status successfully displayed	Pass
6	Check ESP Robot device status	Device online/offline status shown	Online status successfully displayed	Pass
7	Refresh dashboard	Latest data fetched and displayed	Dashboard successfully updated with new data	Pass

All 7 dashboard test scenarios passed. The dashboard successfully synchronizes with the database in real-time. Device status detection via the esp_status heartbeat mechanism functions correctly, providing accurate connectivity information for both ESP32 units. The storage grid visualization always reflects the current slot conditions from the slot and barang tables.

3) Item Input Page Testing

The Item Input page was tested with 7 scenarios covering complete data submission, barcode uniqueness, form validation, sequential item additions, and history recording. This testing ensured that the entire item registration workflow from data entry to barcode generation and database recording functions reliably.

TABLE III. TESTING RESULTS — ITEM INPUT PAGE

No	Input / Test Action	Expected Output	Actual Output	Status
1	Fill and submit all item data fields	Item data saved to database	Data saved to barang table correctly	Pass
2	Enter name, type, weight, and expiry date	All fields stored matching input	Data stored correctly per input	Pass
3	Submit form after all data entered	Barcode and QR Code auto-generated	Barcode and QR Code created successfully	Pass
4	Add multiple items sequentially	All items stored in database	All items stored without error	Pass
5	Leave item name field empty and submit	System rejects the submission	Data not saved, validation warning shown	Pass
6	Check barcode of registered item	Barcode is unique per item	No duplicate barcodes found in database	Pass
7	Save new item and check history	Item activity recorded in history	Record successfully added to history table	Pass

All 7 item input test scenarios passed. The system reliably stores complete item data, generates unique barcodes in the PROD-[date]-[random] format, and automatically creates corresponding history and log_aktivitas records. The form validation mechanism correctly rejects incomplete submissions, and no barcode duplication was detected across all test items.

4) Item Output Page Testing

The Item Output page was tested with 8 scenarios covering all three automatic retrieval methods (FIFO, LIFO, FEFO), the robot command generation process, sequential queue creation, and automatic slot/history updates upon completion. This testing validated the core business logic of the SmartASRS retrieval system.

TABLE IV. TESTING RESULTS — ITEM OUTPUT PAGE

No	Input / Test Action	Expected Output	Actual Output	Status
1	Select available item and trigger output	Item added to retrieval queue	Data inserted into output_queue table	Pass
2	Activate FIFO mode and retrieve	Earliest-entered item selected first	Item selected correctly per FIFO logic	Pass
3	Activate LIFO mode and retrieve	Latest-entered item selected first	Item selected correctly per LIFO logic	Pass
4	Activate FEFO mode and retrieve	Nearest expiry item selected first	Item selected correctly per FEFO logic	Pass
5	Press Retrieve Item button	Robot command created in database	Data inserted into robot_commands table	Pass
6	Retrieve multiple items sequentially	System creates ordered retrieval queue	All queue entries created successfully	Pass
7	Complete output process	Slot status changes to empty	Slot table updated to empty correctly	Pass
8	Complete output process	Activity recorded in history	History table updated with OUTPUT record	Pass

All 8 item output scenarios passed. The system correctly implements all three retrieval methods: FIFO selects by earliest created_at, LIFO by latest created_at, and FEFO by nearest exp_date in the barang table. Robot commands are generated in robot_commands table and the output_queue status transitions correctly through pending, processing, and done states. Slot and history tables are automatically updated upon operation completion.

5) Robot Monitor Page Testing

The Robot Progress monitoring page was tested with 7 scenarios verifying real-time status updates, incremental progress display, stage-by-stage timeline, session tracking, and command completion detection. This testing confirmed that the robot monitoring system provides accurate, live feedback throughout the entire robot operation lifecycle.

TABLE V. TESTING RESULTS — ROBOT MONITOR PAGE

No	Input / Test Action	Expected Output	Actual Output	Status
1	Trigger robot process and observe status	Robot status changes to running/processing	Status successfully changed to PROCESSING	Pass
2	Monitor retrieval operation in progress	Progress bar updates incrementally	Progress displayed correctly per stage	Pass
3	Observe progress percentage values	Percentage increases up to 100%	Progress values: 5%,15%,25%,...,100%	Pass
4	Check operation stage timeline	Each robot stage listed with timestamp	Timeline displayed correctly with steps	Pass
5	Issue multiple robot commands sequentially	All commands processed in order	Commands processed sequentially without error	Pass
6	Wait for operation to complete	Status changes to completed	Completed status displayed correctly	Pass
7	Check robot session information	Session info displayed with item/slot	Session data displayed from robot_session	Pass

All 7 robot monitoring scenarios passed. The progress monitoring system accurately reflects robot operation stages in real-time, with percentage values recorded incrementally in the robot_progress table and displayed on the web page. The session tracking in robot_session correctly records item barcode, target slot, processing status, start and completion timestamps, and processed item counts throughout each operation.

6) Activity History Page Testing

The Activity History page was tested with 8 scenarios verifying complete activity recording, INPUT/OUTPUT differentiation, timestamp accuracy, slot information display, user attribution, sequential recording, and data integrity between the page display and underlying database records.

TABLE VI. TESTING RESULTS — ACTIVITY HISTORY PAGE

No	Input / Test Action	Expected Output	Actual Output	Status
1	Open history page	All system activities displayed	Activity data successfully displayed	Pass
2	Perform item INPUT and check history	INPUT activity recorded in history	INPUT transaction recorded correctly	Pass
3	Perform item OUTPUT	OUTPUT activity recorded in history	OUTPUT transaction recorded correctly	Pass

No	Input / Test Action	Expected Output	Actual Output	Status
	and check history			
4	Check transaction timestamps	Timestamps match actual event times	Timestamps accurate to the second	Pass
5	Check slot column in history	Slot location per transaction shown	Slot data (A1-C3) displayed correctly	Pass
6	Check user column in history	User performing transaction recorded	User name/ID displayed correctly	Pass
7	Perform multiple transactions sequentially	All activities recorded in order	Sequential records stored correctly	Pass
8	Compare page data with database records	Page display matches database	No data discrepancy found	Pass

All 8 history testing scenarios passed. The history page accurately displays all system transactions with complete information including item name, slot, action type (INPUT/OUTPUT), performing user, and timestamp accurate to the second. No discrepancy was found between the data displayed on the history page and the records stored in the history database table, confirming full data integrity.

TABLE VII. WEB MONITORING TESTING RECAPITULATION

No	Module	Scenarios	Passed	Failed
1	Login	5	5	0
2	Dashboard	7	7	0
3	Item Input	7	7	0
4	Item Output	8	8	0
5	Robot Monitor	7	7	0
6	History	8	8	0
	Total	42	42	0

Based on the recapitulation in Table VII, the web monitoring test success rate = $(42/42) \times 100\% = 100\%$. All 42 test scenarios across the 6 modules passed without any failures. This result falls in the Very Good category according to the established success criteria. The testing demonstrates that all web monitoring features authentication, real-time grid display, barcode generation, FIFO/LIFO/FEFO retrieval logic, robot command generation, and complete activity logging function correctly and reliably as designed.

D. Database Testing Results

Database testing was conducted on all 11 MySQL tables to verify that each column correctly stores, updates, and maintains data according to its designed function. Testing was performed by executing system operations through the web monitoring and then observing the resulting database records via phpMyAdmin. The results for each table are presented below.

1) barang Table Testing

The barang table is the central entity of the SmartASRS database, storing complete information for all registered items. Testing covered all 13 columns to verify data correctness from automatic ID generation through to timestamped tracking of creation and modification events.

TABLE VIII. TESTING RESULTS — BARANG TABLE

No	DB Column	Expected Data	Observed Result	Status
1	barang_id	Auto-generated unique ID for each new record	IDs generated automatically: 90, 91, 92, 93...	Pass
2	nama_barang	Item name stored per user input	Names stored correctly: Susu Vanilla, Freshmilk, Indomilk, etc.	Pass
3	jenis_barang	Item type field available and functional	Column functional for item type classification	Pass
4	jenis	Item category stored per user selection	Stored correctly: Minuman, Makanan	Pass
5	berat	Item weight stored as entered	Weights stored: 0 g to 1000 g (flexible input)	Pass
6	barcode	Unique barcode auto-generated per item	Format PROD-YYYYMMDD-XXXX, no duplicates found	Pass
7	qr_code_path	QR Code file path stored in database	Path saved in storage/qrcodes/ matching barcode	Pass
8	exp_date	Expiry date stored per user input	Dates stored correctly: 2026-04-25, 2027-01-01, etc.	Pass
9	keterangan	Additional item notes field available	Column available and accepts custom text	Pass
10	status	Initial item status set after input	All new items correctly assigned status: standby	Pass
11	position	Item position field available	Column available and linked to slot coordinates	Pass
12	created_at	Record creation timestamp auto-recorded	Timestamp matches actual item input time	Pass
13	updated_at	Record update timestamp auto-recorded	Timestamp correctly updated on data changes	Pass

All 13 columns of the barang table functioned correctly. The system generates unique item IDs and barcodes automatically, stores all item attributes accurately, and maintains creation/modification timestamps automatically. No empty or incorrectly stored data was found. The barang table also serves as the

primary data source for the dashboard storage grid, output retrieval logic, and history recording, confirming its critical role in system integration.

2) history Table Testing

The history table records every item storage and retrieval transaction, serving as the primary audit trail for all I/O activities in SmartASRS. Testing covered all 11 columns to verify that each transaction is completely and accurately documented.

TABLE IX. TESTING RESULTS — HISTORY TABLE

No	DB Column	Expected Data	Observed Result	Status
1	history_id	Auto-generated sequential transaction ID	IDs generated automatically for every activity	Pass
2	barang_id	Processed item ID recorded in transaction	Item ID correctly linked to barang table	Pass
3	nama_barang	Item name recorded at transaction time	Names recorded: Susu Vanila, Indomie, Teh Botol, etc.	Pass
4	jenis	Item category recorded in transaction	Categories recorded: Makanan and Minuman	Pass
5	berat	Item weight recorded per item data	Weight recorded matching barang table data	Pass
6	barcode	Item barcode recorded as transaction identifier	Barcode stored matching processed item's barcode	Pass
7	slot_id	Slot location of storage/retrieval recorded	Slots recorded: A1, A2, A3, B1, B2, B3, C1, C2, C3	Pass
8	exp_date	Item expiry date recorded in transaction	exp_date correctly recorded matching item data	Pass
9	action_type	System differentiates INPUT and OUTPUT	Values stored as INPUT or OUTPUT per activity type	Pass
10	user_id	User performing the transaction recorded	All transactions linked to the active user session	Pass
11	created_at	Transaction timestamp auto-recorded	Timestamps accurate to the second per event	Pass

All 11 history table columns functioned as designed. Every I/O transaction generates a new record capturing complete item information, storage location, activity type, responsible user, and precise timestamp. The action_type column correctly distinguishes between INPUT and

OUTPUT events, enabling the history page filter functionality to work accurately. Slot locations from A1 to C3 were all correctly recorded across the test transactions.

3) log_aktivitas Table Testing

The log_aktivitas table provides a more granular activity audit trail compared to the history table, recording operational activities such as scanner events, robot actions, and user interactions at a finer level of detail. Testing covered all 7 columns.

TABLE X. TESTING RESULTS — LOG_AKTIVITAS TABLE

No	DB Column	Expected Data	Observed Result	Status
1	log_id	Auto-generated sequential log ID	Log IDs generated automatically and incrementally	Pass
2	user_id	User performing activity recorded	Activities correctly linked to active user account	Pass
3	aktivitas	Activity type and description recorded	Activities recorded: Scan INPUT, Robot PUT, Robot TAKE	Pass
4	barang_id	Item involved in activity recorded	Item ID stored matching the activity performed	Pass
5	slot	Slot location involved in activity recorded	Slots A1, A2, B1, B2, B3 stored correctly	Pass
6	keterangan	Additional activity description stored	Activity notes recorded per operational step	Pass
7	created_at	Activity timestamp auto-recorded	Timestamp stored matching event time	Pass

All 7 log_aktivitas columns functioned correctly. The log records all operational activities including barcode scan events, robot PUT operations, and robot TAKE operations with their associated items, slots, and timestamps. This table enables more detailed troubleshooting and operational analysis compared to the higher-level history table, providing a complete operational audit trail for the SmartASRS system.

4) output_queue Table Testing

The output_queue table manages the retrieval request pipeline between the user's web command and the physical robot execution. It tracks each retrieval request from initial creation through to final completion. Testing covered all 7 columns.

TABLE XI. TESTING RESULTS — OUTPUT_QUEUE TABLE

No	DB Column	Expected Data	Observed Result	Status
1	id	Auto-generated queue entry ID	Queue IDs created for each output request	Pass

No	DB Column	Expected Data	Observed Result	Status
2	slot_id	Location of item to retrieve recorded	Slots recorded: A1, B2, B3, C1, C2, C3	Pass
3	barang_id	ID of item to be retrieved recorded	Selected item correctly stored in queue	Pass
4	nama_barang	Name of item to retrieve recorded	Item name displayed from barang table reference	Pass
5	status	Queue status transitions per robot execution	Status correctly changes: pending→processing→done	Pass
6	created_at	Queue entry creation timestamp recorded	Timestamp stored automatically on queue creation	Pass
7	updated_at	Queue status change timestamp recorded	Timestamp updated correctly on each status change	Pass

All 7 output queue columns functioned correctly. Every item selected for retrieval via the web monitoring generates a queue entry with the correct slot, item ID, and initial status of pending. As the robot processes the command, the status transitions to processing and finally to done. The timestamp fields accurately capture both the creation time and the time of each status update, enabling full traceability of the retrieval pipeline.

5) robot_commands Table Testing

The robot_commands table stores all commands sent to the ESP32 robot controller and tracks their execution status. It serves as the communication bridge between the server and the physical robot. Testing covered all 7 columns.

TABLE XII. TESTING RESULTS — ROBOT_COMMANDS TABLE

No	DB Column	Expected Data	Observed Result	Status
1	id	Auto-generated sequential command ID	Command IDs generated automatically per command	Pass
2	command	Robot command code stored	Commands stored: A, E, F, 1, 2, 3 and other codes	Pass
3	barcode	Barcode of item being processed recorded	Barcode stored matching the item being handled	Pass
4	status	Command status updates per robot execution	Status changes correctly to completed after execution	Pass

No	DB Column	Expected Data	Observed Result	Status
5	esp_id	Robot device identifier recorded	Data stored as ESP1 identifying the controller	Pass
6	created_at	Command creation timestamp recorded	Timestamp stored automatically on command creation	Pass
7	updated_at	Command completion timestamp recorded	Timestamp updated correctly when process completes	Pass

All 7 robot_commands columns functioned correctly. Each robot operation generates a command record that ESP32-ESP1 periodically polls via the get_robot_command.php endpoint. Command codes (A,B,C,D,E,F,G,H,I,1,2,3,4,5,6,7,8,9) represent specific robot movements, and the status field transitions from pending to completed upon successful execution. The esp_id field correctly identifies the device responsible for command execution.

6) robot_progress Table Testing

The robot_progress table stores incremental stage-by-stage progress data for each robot operation, enabling the real-time progress bar and stage timeline displayed on the Robot Monitor page. Testing covered all 5 columns.

TABLE XIII. TESTING RESULTS — ROBOT_PROGRESS TABLE

No	DB Column	Expected Data	Observed Result	Status
1	id	Auto-generated progress entry ID	Progress IDs generated automatically per step	Pass
2	command_id	Progress record linked to correct command	All progress records correctly linked to command_id	Pass
3	step	Robot operation stage description recorded	Each robot step recorded sequentially in the table	Pass
4	percent	Progress percentage increases through operation	Percentages recorded: 5%,15%,25%,35%,45%,50%,60%,70%,78%,85%,90%,95%,100%	Pass
5	timestamp	Timestamp of each step recorded	Timestamps stored on each progress update	Pass

All 5 robot_progress columns functioned correctly. The table records 13 distinct progress stages from 5% to 100% for each robot operation. These incremental records are read by the robot_monitor.php endpoint and displayed on the web monitoring page in real-time, giving users accurate visual feedback of the robot's current operation stage. The command_id foreign key correctly links each progress record to its parent robot command.

7) robot_session Table Testing

The robot_session table maintains comprehensive information about active robot operation sessions, including the item currently being processed, target slot, session status, and timing information. Testing covered all 9 columns.

TABLE XIV. TESTING RESULTS — ROBOT_SESSION TABLE

No	DB Column	Expected Data	Observed Result	Status
1	session_id	Auto-generated session ID	Session IDs generated automatically per operation	Pass
2	status	Session status updated per robot condition	Status changes correctly per operation phase	Pass
3	started_at	Session start timestamp recorded	Timestamp stored automatically on session creation	Pass
4	completed_at	Session completion timestamp recorded	Timestamp stored correctly after process completion	Pass
5	total_items	Total items in session recorded	Item count stored correctly for the session	Pass
6	processed_items	Number of processed items updated	Data updated incrementally per robot progress	Pass
7	current_barcode	Barcode of item being processed recorded	Correct barcode displayed per active operation	Pass
8	current_slot	Slot being processed recorded	Correct slot displayed per active operation	Pass
9	last_update	Session last update timestamp recorded	Timestamp updated automatically per event	Pass

All 9 robot_session columns functioned correctly. The session table provides a comprehensive real-time snapshot of the robot's current operation state. Users can see exactly which item (by barcode) and which slot the robot is currently working on, along with session timing and progress counts. This data feeds directly into the Robot Progress monitoring page's header information panel.

8) slot Table Testing

The slot table represents the physical storage locations in the miniature ASRS, storing each slot's identity, robot navigation coordinates, current occupant, and status. Testing covered all 8 columns.

TABLE XV. TESTING RESULTS — SLOT TABLE

No	DB Column	Expected Data	Observed Result	Status
1	slot_id	All slot identities stored in database	Slots stored: A1, A2, A3, B1, B2, B3, C1, C2, C3	Pass

No	DB Column	Expected Data	Observed Result	Status
2	barang_id	Item occupying each slot recorded	Item correctly linked to the slot it occupies	Pass
3	koordinat_x	Robot X-axis coordinate stored per slot	X coordinates stored matching system design	Pass
4	koordinat_y	Robot Y-axis coordinate stored per slot	Y coordinates stored matching system design	Pass
5	koordinat_z	Robot Z-axis coordinate stored per slot	Z coordinates stored matching system design	Pass
6	status	Slot status updated per storage activity	Status correctly changes between empty and filled	Pass
7	created_at	Slot record creation timestamp stored	Timestamp stored automatically	Pass
8	updated_at	Slot update timestamp recorded	Timestamp correctly updated on each status change	Pass

All 8 slot table columns functioned correctly. All nine physical storage locations (A1-C3) are represented in the database with their unique coordinates used for robot navigation. The status field transitions between empty and filled states automatically with each INPUT and OUTPUT operation, ensuring the database always reflects the actual physical state of the storage rack. This table is the primary data source for the dashboard's storage grid visualization.

9) esp_status Table Testing

The esp_status table monitors the connectivity status of ESP32 devices by tracking their last heartbeat timestamp. The system marks a device as offline if no heartbeat has been received within 300 seconds. Testing covered all 3 columns.

TABLE XVI. TESTING RESULTS — ESP_STATUS TABLE

No	DB Column	Expected Data	Observed Result	Status
1	id	Auto-generated device record ID	Device IDs created in database	Pass
2	esp_id	ESP device identity stored	ESP1 and ESP2 stored as device identifiers	Pass
3	last_update	Last heartbeat timestamp recorded	Timestamp updated each time device sends data	Pass

All 3 esp_status columns functioned correctly. Both ESP32 devices (ESP1 and ESP2) are registered in the table and their last_update timestamps are refreshed each time the devices communicate with the server. The dashboard's device status indicator reads this table to determine online/offline status based on the time elapsed since the last update, providing reliable real-time device connectivity monitoring.

10) settings Table Testing

The settings table stores system configuration parameters, particularly the active item retrieval mode (FIFO, LIFO, or FEFO). This table allows the storage method to be changed dynamically through the web interface without modifying the system code. Testing covered all 4 columns.

TABLE XVII. TESTING RESULTS — SETTINGS TABLE

No	DB Column	Expected Data	Observed Result	Status
1	setting_id	Auto-generated configuration ID	Config IDs stored correctly in database	Pass
2	setting_key	System configuration key name stored	Key stored correctly as storage_mode	Pass
3	setting_value	Configuration value stored per user selection	Values stored correctly: FIFO, LIFO, and FEFO	Pass
4	updated_at	Configuration change timestamp recorded	Timestamp updated each time configuration changes	Pass

All 4 settings table columns functioned correctly. When the user changes the retrieval mode through the web dashboard, the setting_value field is updated to FIFO, LIFO, or FEFO accordingly, and the updated_at timestamp is refreshed. This mechanism enables dynamic system configuration without code changes, and the output retrieval logic reads this value to determine item priority for each retrieval operation.

11) users Table Testing

The users table stores user account data for authentication and activity attribution. Security is maintained by storing passwords as bcrypt hashes rather than plain text. Testing covered all 7 columns.

TABLE XVIII. TESTING RESULTS — USERS TABLE

No	DB Column	Expected Data	Observed Result	Status
1	user_id	Auto-generated user ID	User IDs created automatically in database	Pass
2	username	Username stored per user account	Username stored and correctly used for login	Pass
3	password	Password stored as encrypted hash	Password stored as bcrypt hash (\$2y\$10\$...)	Pass
4	nama_lengkap	Full name stored per account identity	Full name stored correctly in database	Pass
5	role	User access role stored	Role stored correctly as Administrator	Pass
6	created_at	Account creation timestamp recorded	Timestamp stored automatically	Pass

No	DB Column	Expected Data	Observed Result	Status
			on account creation	
7	updated_at	Account modification timestamp recorded	Timestamp updated correctly on account changes	Pass

All 7 users table columns functioned correctly. User data is stored securely with bcrypt-hashed passwords that cannot be reversed to plain text. The role field enables future role-based access control expansion. The users table is queried by the login system for authentication, and its user_id is referenced by the history and log_aktivitas tables to attribute all system activities to specific users.

TABLE XIX. DATABASE TESTING RECAPITULATION

No	Table	Scenarios	Passed	Failed
1	barang	13	13	0
2	history	11	11	0
3	log_aktivitas	7	7	0
4	output_queue	7	7	0
5	robot_commands	7	7	0
6	robot_progress	5	5	0
7	robot_session	9	9	0
8	slot	8	8	0
9	esp_status	3	3	0
10	settings	4	4	0
11	users	7	7	0
	Total (11 tables)	81	81	0

Based on Table XIX, the database test success rate = $(81/81) \times 100\% = 100\%$. All 81 scenarios across all 11 tables passed without any failures, placing the result in the Very Good category. Item data, transaction history, robot activities, slot status, device connectivity, system configuration, and user authentication data are all stored, updated, and maintained correctly without any data loss or inconsistency throughout the testing period.

E. System Performance and Latency Analysis

The functional testing conducted in the previous subsections confirmed that all SmartASRS features operate as designed. However, an equally important dimension is system performance in terms of timing: how long the robot takes to complete a mission, how progress updates are transmitted to the database, and whether network-induced delays affect the accuracy of information displayed on the web monitoring dashboard. This analysis is based on data recorded in the robot_commands and robot_progress tables during actual system operation.

1) Robot Mission Completion Time

Robot mission duration was measured as the elapsed time between the created_at column (command received with pending status) and the updated_at column (status changed to completed) in the robot_commands table. Two mission types exist: PUT missions (item storage to a slot, command codes 1–9 for slots A1–C3) and TAKE missions (item retrieval from a slot, command codes A–I). Table

XX presents sample PUT mission data and Table XXI presents sample TAKE mission data.

TABLE XX. SAMPLE PUT MISSION COMPLETION TIMES

No	Cmd ID	Slot	Start Time	End Time	Duration (s)
1	156	B1	2026-04-01 08:19:17	2026-04-01 08:20:55	98
2	162	A1	2026-04-01 08:44:14	2026-04-01 08:45:59	105
3	163	A2	2026-04-02 04:41:50	2026-04-02 04:43:26	96
4	165	A1	2026-04-08 08:34:04	2026-04-08 08:35:39	95
5	171	A1	2026-04-10 07:04:49	2026-04-10 07:06:18	89
6	172	A2	2026-04-10 07:36:21	2026-04-10 07:38:10	109
7	173	A3	2026-04-10 07:40:07	2026-04-10 07:41:42	95
8	174	B1	2026-04-10 07:42:14	2026-04-10 07:44:15	121
9	175	B2	2026-04-10 07:47:50	2026-04-10 07:49:16	86
10	176	B3	2026-04-10 07:51:18	2026-04-10 07:52:44	86

Based on the 10 PUT samples in Table XX, mission duration ranges from 86 to 121 seconds. Across all 60 recorded PUT missions, the overall average is 125.1 seconds (approximately 2 minutes 5 seconds). Duration variation is primarily influenced by the target slot's distance from the robot's home position — slots farther along the X and Y axes require longer motor travel time, resulting in proportionally longer mission durations.

TABLE XXI. SAMPLE TAKE MISSION COMPLETION TIMES

No	Cmd ID	Slot	Start Time	End Time	Duration (s)
1	91	B1	2026-03-12 09:01:48	2026-03-12 09:03:09	81
2	92	B1	2026-03-12 09:07:14	2026-03-12 09:08:39	85
3	94	A3	2026-03-12 09:12:24	2026-03-12 09:13:57	93
4	95	B2	2026-03-12 09:16:46	2026-03-12 09:18:07	81
5	96	B3	2026-03-12 09:19:29	2026-03-12 09:20:51	82
6	101	B1	2026-03-12 10:00:42	2026-03-12 10:02:04	82
7	106	C3	2026-03-12 10:15:03	2026-03-12 10:16:34	91
8	112	A2	2026-03-13 07:48:19	2026-03-13 07:49:45	86

No	Cmd ID	Slot	Start Time	End Time	Duration (s)
9	113	C3	2026-03-13 08:24:11	2026-03-13 08:25:41	90
10	114	A3	2026-03-13 08:26:09	2026-03-13 08:27:37	88

Based on Table XXI, TAKE mission duration in the 10 samples ranges from 81 to 93 seconds. The overall average across all 53 valid TAKE records is 149.2 seconds — longer than PUT missions on average. The additional time is required because the robot must ensure the gripper successfully grips the item and fully extracts it from the slot before traveling to the conveyor discharge point, adding a mechanical gripping and extraction phase not present in PUT missions.

2) Robot Progress Stage Analysis

Each robot mission is recorded in incremental stages through the robot_progress table. ESP32-ESP2 reports its progress to the server after each movement stage is completed, and these records drive the real-time progress bar displayed on the web monitoring dashboard. Table XXII presents the detailed stage breakdown for TAKE mission command_id 91 (slot B1) as a representative example.

TABLE XXII. ROBOT PROGRESS STAGE DETAIL — TAKE MISSION (CMD_ID=91, SLOT B1)

No	Progress	Stage Description	Timestamp	Gap (s)
1	5%	Initiating item retrieval from storage slot	09:01:52	—
2	15%	Robot arrived at slot, gripper open, ready to retrieve	09:02:10	18
3	25%	Gripper fully open, robot advancing into slot	09:02:11	1
4	35%	Robot inside slot, gripper aligned with item	09:02:14	3
5	45%	Gripper gripping item firmly, item ready to lift	09:02:15	1
6	50%	Item successfully lifted and extracted from slot	09:02:18	3
7	60%	Robot retreating from slot, moving to discharge point	09:02:22	4
8	70%	Robot arrived at discharge point, advancing to conveyor	09:02:33	11
9	78%	Robot at conveyor, gripper opening to release item	09:02:36	3
10	85%	Gripper open, item released, robot retreating from conveyor	09:02:41	5
11	90%	Robot retreating, gripper closing	09:02:43	2

No	Progress	Stage Description	Timestamp	Gap (s)
12	95%	Item on conveyor, robot returning to home position	09:02:45	2
13	100%	Item successfully dispatched, mission complete	09:03:09	24

Based on Table XXII, the TAKE mission from slot B1 was completed in 13 stages with a total duration of 77 seconds. The three longest inter-stage gaps are: stage 2 (robot traveling to the slot front, 18 seconds), stage 13 (robot returning to home position after releasing the item, 24 seconds), and stage 8 (robot moving from the slot to the conveyor discharge point, 11 seconds). These stages involve stepper motor travel over greater distances. In contrast, mechanical gripper actions (open/close) require only 1–3 seconds per stage, reflecting the fast response of the servo mechanism.

3) Progress Update Delays and Network Anomalies

The inter-step delay measured as the difference between consecutive timestamps in the robot_progress table for each command_id — reflects the time ESP32-ESP2 takes to complete each physical movement and transmit the report to the server. Under normal operating conditions, analysis of 1,455 inter-step intervals shows delays ranging from 1 to 30 seconds with a mean of 7.8 seconds per stage, consistent with the robot's physical movement durations.

However, 16 anomalous delay cases were identified in which the inter-step gap significantly exceeded the normal range, with the longest reaching 226 seconds. These anomalies do not reflect mechanical robot failures. Instead, they indicate temporary Wi-Fi connectivity disruptions between ESP32-ESP2 and the server. When ESP2 loses its Wi-Fi connection mid-mission, the progress report for that stage is buffered locally and transmitted only after the connection is restored. During this gap, the robot continues executing its programmed movement sequence locally, but the web monitoring dashboard cannot update the progress bar because no new data reaches the server. Table XXIII lists selected anomaly cases identified during testing.

TABLE XXIII. ANOMALOUS PROGRESS UPDATE DELAYS DUE TO NETWORK DISRUPTION

Cmd ID	Mode	Slot	Affected Stage	Gap (s)	Description
115	TAKE	A3	25%→50%	124	Progress 25% sent but 50% delayed — ESP2 lost connection while gripping item
135	TAKE	B2	45%→60%	134	ESP2 failed to send 60% report on time after robot lifted item successfully
147	TAKE	A1	90%→100%	123	Mission completion report (100%) delayed; dashboard did not immediately show done
193	PUT	A1	60%→78%	134	Delayed report while robot moved from slot

Cmd ID	Mode	Slot	Affected Stage	Gap (s)	Description
					to conveyor; likely weak Wi-Fi signal

As shown in Table XXIII, all anomaly cases occurred at a single stage within an otherwise complete mission and did not cause the mission to fail entirely. After the Wi-Fi connection was restored, ESP2 successfully transmitted the remaining progress reports and all missions reached 100% completion. This demonstrates that the system has inherent resilience against transient network disruptions: ESP32-ESP2 stores the mission logic locally and does not depend on server responses to continue robot movement. The only observable impact is a delayed progress bar update on the dashboard, not an operational failure. These anomalies can be mitigated in future development by adopting the MQTT protocol with Quality of Service (QoS) guarantees, and by implementing a local message queue on ESP2 to ensure sequential delivery of all progress reports upon connection recovery.

4) Database Synchronization Delay

A secondary timing dimension is the database synchronization delay the elapsed time between the robot_progress table recording 100% completion and the robot_commands table updating to completed status. This delay arises because two API endpoints are called sequentially: update_progress3.php writes the final progress record, then complete_command.php updates the command status. Under normal conditions (85 valid missions), synchronization delay ranges from 0 to 30 seconds with a mean of 4.3 seconds negligible in practice and imperceptible to users.

However, 4 extreme delay cases were identified ranging from 373 seconds to over one day. These occurred when the ESP2 Wi-Fi connection dropped immediately after the robot completed its final movement, preventing the complete_command.php request from being delivered. The completed status was only recorded when connectivity was restored in a subsequent testing session. Importantly, the physical state of the robot and items was unaffected in all cases the robot had already completed its movement and the item was correctly placed before the connection was lost.

5) System Performance Summary

All measured timing parameters for the SmartASRS system are summarized in Table XXIV, covering both normal operating conditions and anomalous cases attributable to network disruptions.

TABLE XXIV. SMARTASRS SYSTEM PERFORMANCE STATISTICS SUMMARY

Parameter	Min	Max	Mean	N Samples
PUT Mission Duration	73 s	352 s	125.1 s	60 missions
TAKE Mission Duration	79 s	394 s	149.2 s	53 missions
Inter-step Delay (normal)	1 s	30 s	7.8 s	1,455 intervals
Inter-step Delay (anomaly)	67 s	226 s	—	16 cases

Parameter	Min	Max	Mean	N Samples
DB Sync Delay (normal)	0 s	30 s	4.3 s	85 missions
DB Sync Delay (anomaly)	373 s	> 1 day	—	4 cases

Based on Table XXIV, SmartASRS achieves an average mission completion time of 125.1 seconds for PUT operations and 149.2 seconds for TAKE operations. The mean inter-step progress update delay of 7.8 seconds under normal conditions reflects responsive real-time communication between ESP2 and the server. The 4.3-second mean database synchronization delay confirms that mission status is updated promptly following robot completion. The presence of anomalous delay cases highlights Wi-Fi connection stability as the primary performance bottleneck. Since anomalies result exclusively from network disruptions not mechanical or software failures and all affected missions ultimately completed successfully, the findings indicate that the SmartASRS system achieves reliable functional operation while identifying network reliability as the key area for improvement in future development.

F. Overall System Integration Analysis

The integrated testing of all system components confirms that SmartASRS operates as a cohesive, synchronized system. When a barcode is scanned by the GM65-S scanner, data flows from ESP32-ESP1 through the HTTP API to the server, where it is processed and stored in the barang and history tables. The web dashboard immediately reflects this change in the storage grid and history page without manual refresh.

For robot operations, the command flow from web user to physical robot was verified to be complete and traceable: user selection on the web creates output queue and robot commands records, ESP32-ESP1 polls and picks up the command, transmits it to ESP32-ESP2 via ESP-NOW, the robot executes the physical movement while reporting 13 progress stages back to the server (5% to 100%), and all state changes slot status, history records, robot_progress, robot_session are committed to the database accordingly.

The 100% functional success rates across both web monitoring and database testing confirm that the SmartASRS integration achieves complete functional correctness. The performance analysis further reveals that the system operates with an average mission time of 125.1 s (PUT) and 149.2 s (TAKE), with normal inter-step communication delays averaging 7.8 seconds. The 16 identified anomalous delay cases and 4 synchronization delay extremes are attributable exclusively to transient Wi-Fi disruptions and did not result in operational failures, demonstrating that the system maintains functional resilience under intermittent network conditions. No data inconsistencies were detected between related tables, confirming a robust and reliable data management architecture.

IV. CONCLUSION

This study successfully designed and implemented the SmartASRS system an IoT-based miniature ASRS monitoring platform with barcode identification using ESP32. Three main conclusions are drawn.

First, the web monitoring application integrated with an 11-table MySQL database was successfully implemented and validated. Black Box Testing achieved a 100% success rate across all 42 web monitoring scenarios (6 modules), confirming reliable operation of all features from authentication and real-time slot visualization to FIFO/LIFO/FEFO retrieval, robot command generation, and full activity logging.

Second, the barcode identification system using the GM65-S scanner and ESP32 was successfully implemented. The system generates unique barcodes (PROD-[date]-[random]), transmits scan data via Wi-Fi/HTTP, and stores item data without duplication. Database testing achieved 100% success across 81 scenarios in all 11 tables, confirming structural data integrity throughout.

Third, performance analysis across 113 real robot missions showed average completion times of 125.1 s (PUT) and 149.2 s (TAKE) with a mean inter-step delay of 7.8 s. Sixteen Wi-Fi-induced anomalous delays were identified, none causing operational failures, confirming system resilience. Overall, SmartASRS demonstrates that an integrated IoT-based ASRS monitoring system can be effectively realized at miniature scale serving as both a functional warehouse management tool and a practical learning medium for industrial automation.

Future work should adopt MQTT with QoS guarantees, implement a local message queue on ESP2, and add automated expiry date notifications. Additionally, three interface improvements are recommended for industrial-scale deployment: (1) a role-based access control system differentiating Administrator, Operator, and Viewer roles so each user only accesses features relevant to their responsibilities; (2) a list-based slot display replacing the fixed 3×3 grid, enabling scalable visualization for environments with thousands of storage locations; and (3) a dedicated inventory monitoring page showing real-time stock levels, remaining capacity, and cumulative input/output counts to support data-driven warehouse management decisions.

ACKNOWLEDGMENT

The author expresses gratitude to the supervisor, Tatyantoro Andrasto, S.T., M.T., for guidance and direction throughout this research, and to the Department of Electrical Engineering, Faculty of Engineering, Universitas Negeri Semarang, for providing facilities and support for this study.

REFERENCES

- [1] A. Jarasoniene, K. Ciziuniene, and A. Cereska, "Research on Impact of IoT on Warehouse Management," *Sensors*, vol. 23, no. 4, 2023.
- [2] M. G. Khan, N. Ul Huda, and U. K. Uz Zaman, "Smart Warehouse Management System: Architecture, Real-Time Implementation and Prototype Design," *Machines*, vol. 10, no. 2, 2022.
- [3] S. Antomarioni, L. Lucantoni, F. E. Ciarapica, and M. Bevilacqua, "Data-driven decision support system for managing item allocation in an ASRS," *Expert Systems with Applications*, vol. 185, 2021.
- [4] D. Hercog, T. Lerher, M. Truntic, and O. Tezak, "Design and Implementation of ESP32-Based IoT Devices," *Sensors*, vol. 23, no. 15, 2023.
- [5] C. R. Sahara and A. M. Aamer, "Real-time data integration of an internet-of-things-based smart warehouse: a case study," *Int. J. Pervasive Computing and Communications*, vol. 18, no. 5, pp. 622-644, 2022.
- [6] E. Anisa and A. S. Juhri, "Rancang Bangun Sistem Manajemen Gudang Terintegrasi Barcode Berbasis Web," vol. 12, no. 4, 2025.
- [7] S. A. P. Pratama, W. G. E. Rahim, and M. A. Nugraha, "IoT-Based Barcode Scanning System Implementation Using ESP32 for Enhancing Warehouse Stock Opname Efficiency," *Int. J. Research and Applied Technology*, vol. 5, no. 1, 2025.
- [8] E. O. Sodiya, U. J. Umoga, O. O. Amoo, and A. Atadoga, "AI-driven warehouse automation: A comprehensive review of systems," *GSC Advanced Research and Reviews*, vol. 18, no. 2, pp. 272-282, 2024.
- [9] B. Y. Ekren and A. Akpunar, "An open queuing network-based tool for performance estimations in a shuttle-based storage and retrieval system," *Applied Mathematical Modelling*, vol. 89, pp. 1678-1695, 2021.
- [10] E. Abdullah et al., "Development of Real-Time Energy Monitoring System and Data Log Using NodeMCU ESP8266 and MySQL Database," *Journal of Mechanical Engineering*, vol. 10, no. 1, 2021.