Published by :
https://www.ijert.org/
An International Peer-Reviewed Journal

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

AutoCart: An RFID-Enabled Smart Shopping Cart with Weight Verification and Self-Checkout

Sameer B
Department of Computer Science and Engineering
Jain Institute of Technology
Davangere, India

Shafiullah Shariff
Department of Computer Science and Engineering
Jain Institute of Technology
Davangere, India

Mohammed Adil
Department of Computer Science and Engineering
Jain Institute of Technology
Davangere, India

Muralidhara H. N.
Department of Computer Science and Engineering
Jain Institute of Technology
Davangere, India

Mohammed Shahid
Department of Computer Science and Engineering
Jain Institute of Technology
Davangere, India

Rahul S Biradar
Department of Computer Science and Engineering
Jain Institute of Technology
Davangere, India

***Abstract*—AutoCart aims to develop and assess a smart shopping cart that can identify products, track items within a cart, and measure the cart contents. It also provides an interactive proof-of-concept checkout. The cart contains an MFRC522 RFID reader and an ESP32 controller that communicate with a backend API over Wi-Fi. Product and transaction information is stored in a relational database and presented through a web application. The system is designed to update item information after scanning and to compare anticipated item weight with the measured cart weight. The reported prototype results include an RFID detection rate of 96% across 50 scans and an average scan-to-display response time of 1.6 s. An observed HX711 measurement error of 0.812% was also reported; however, the raw measurements and complete calibration records were not available for independent verification. The QR-based payment workflow is simulated and does not perform a real bank or Unified Payments Interface (UPI) transaction. This paper describes the system architecture, implementation workflow, evaluation evidence, limitations, and future improvements.**

***Index Terms*—RFID, Smart Cart, ESP32, Load Cell, HX711, Self Checkout, QR Payment, IoT, Automated Billing.**

I. Introduction

In traditional supermarkets and retail stores, customers can be required to manually scan items in order to pay for them through checkout points. During peak times this can cause delays to customers as queues form. Manual scanning can also cause billing errors. RFID Smart Shopping Carts have been looked into as a way to improve product identification and automate the billing process [1]–[4].

AutoCart is an RFID-based Smart Shopping Cart that enables automatic product identification, real-time billing, weight confirmation, and allows the user to do self-checkout. The system uses an RC522 RFID reader to read the unique ID of each item. The ESP32-WROOM-32UE takes the scanned data and communicates with the backend. This gets the product information, updates the cart, and calculates the current bill.

The system also contains a load cell connected to an HX711 signal-conditioning module. This gives a total weight of the cart contents and compares it to the expected weight. This provides a second confirmation of what should be in the cart.

The overall aim of AutoCart is to make life easier for the customer by allowing them to identify the products, see up-to-date billing information, and complete a QR code checkout. The communication between the cart and backend is done using Wi-Fi. A relational database allows the product, billing, and transaction information to be stored. Monitoring at the exit uses the payment and cart status to cause an alarm if any suspicious conditions are met.

This prototype achieved a 96% detection rate using RFID during testing. It had an average response time of 1.6 seconds between scanning and display. The HX711 had an error rate of 0.812%. These results show that the prototype would work. However, this system has not been compared to traditional checkout methods in a controlled environment nor tested in a retail environment. Therefore, improvements to reduce queues, prevent fraud, and readiness for commercialisation cannot be claimed.

A. Research Objectives and Contributions

AutoCart aims to create and assess a smart shopping trolley that can recognise products, track items within a trolley, and measure their combined weight. It also includes an interactive proof-of-concept checkout. The trolley contains an RFID reader and an ESP32 device that communicate with each other, while data is sent to a backend API using Wi-Fi connectivity. Data is stored in a database and accessed through a frontend web application. The system is designed so that item information can be updated consistently after scanning and actual and anticipated item weights can be analysed.

There are four main objectives. First, RFID tags are used to map physical products to records in a product database. Sec-

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

ond, scan and weight information is communicated from the trolley to the backend through Wi-Fi. Third, trolley sessions, item quantities, item prices, anticipated item weight, and transaction details are stored and updated in the backend database. Finally, a QR-code-enabled proof-of-concept checkout and a configurable weight-validation process are provided. Payment remains simulated; no real connection to a bank or UPI service is implemented.

The key contributions of the prototype are:

- Synchronisation of an MFRC522 RFID reader and ESP32 device to read RFID tags and relay scan information to the backend.
- Standardisation of RFID tag data, product availability lookup, duplicate-scan prevention, and trolley-session updates within the backend.
- Connection of a load cell through an HX711 interface to allow measured weight to be compared with the anticipated trolley weight.
- Adoption of a multi-tier architecture comprising FastAPI, SQLAlchemy, a relational database, and a React/TypeScript web application.
- Implementation of a QR-code payment proof of concept that records simulated payment data and distinguishes it from genuine payment-gateway settlement.

II. Related Work and Research Gap

Research on RFID in retail has considered automated product identification, shopping-cart billing, inventory management, supply-chain operations, and security. RFID-enabled shopping carts have been proposed to reduce manual product scanning and support automated billing. The literature includes intelligent shopping carts for streamlined checkout [1], smart trolleys with automated billing interfaces [2], and RFID-based carts designed to improve shopping efficiency [3], [4]. Other studies consider mobile-application integration, IoT-enabled shelf and trolley systems, and automated inventory-related functions [5]–[8]. Theft prevention and product tracking have also been explored in RFID shopping-cart designs [9]–[14].

RFID security is another relevant area because product identification depends on reliable tag communication and appropriate protection of RFID data. Previous work has examined RFID authentication, security and privacy requirements, and vulnerabilities in RFID protocols [15]–[17]. RFID has also been applied to retail inventory and supply-chain coordination, including autonomous inventory systems and item-level tracking [18]–[20]. Together, these studies provide context for applying RFID technology to retail automation.

AutoCart combines RFID event processing, cart-session control, anticipated-versus-measured weight assessment, backend data handling, and a demonstrative checkout mechanism in a single prototype. Weight verification is treated as a supplementary consistency check: it can indicate a difference between recorded cart contents and measured mass, but it does not independently identify which product caused the difference or prove that theft has occurred. Further controlled experiments are required before claims about improved retail efficiency, fraud prevention, or large-scale reliability can be made.

A. Positioning of the Present Work

The studies considered in this paper cover RFID-based shopping carts, automated billing, IoT-enabled retail systems, RFID security, and retail inventory applications. These works provide the background for using automatic identification in retail environments. AutoCart focuses on integrating RFID event processing with cart-session management, anticipated-versus-measured weight assessment, backend data handling, and a demonstrative checkout interface.

Table I summarises the principal focus indicated by selected references. The table positions the topics addressed by the cited works; it does not claim that a feature is absent from a cited paper unless that paper has been examined in detail. A quantitative comparison of accuracy, latency, cost, or fraud-detection performance would require consistent test conditions and comparable datasets across all systems. Such superiority is therefore not claimed in this work.

The research gap addressed by the present prototype is the implementation-level coordination of these functions within one cart-to-backend workflow. RFID events are associated with a server-managed cart session, while anticipated and measured cart weights are maintained as separate attributes. The current work is an integration study; additional controlled experimentation is necessary before conclusions can be drawn about retail throughput, anti-fraud performance, or large-scale system dependability.

TABLE I
Positioning of Selected Related Work and AutoCart

Reference	Focus identified in the cited work	Related aspect addressed by AutoCart
[1]	RFID-based shopping trolley for automated billing	RFID product identification, cart updates, and billing workflow.
[2]	Smart cart for automatic billing with an integrated RFID system	RFID-based product registration and automated cart calculation.
[4]	Human-friendly smart trolley with automatic billing	RFID-enabled shopping-cart design; AutoCart uses the MFRC522 reader in its supplied firmware.
[5]	Intelligent shopping trolley integrated with a mobile application	Network communication between the cart controller and backend services.
[6]	Smart shelf and trolley systems utilizing IoT technologies	RFID scanning and a checkout-oriented workflow.
[7]	RFID-enabled IoT smart shopping cart for automated billing and enhanced retail experience	Security-oriented cart monitoring; AutoCart includes weight-status and event-handling logic.
This work	Integrated prototype implementation	RFID event processing, duplicate-scan handling, weight assessment, cart sessions, backend records, and simulated QR checkout.

III. System Architecture

The AutoCart system is composed of various components. These include: RFID identification module, microcontroller, weight detection unit, backend, database, and checkout monitor. All perform specific tasks.

The RFID tag is used to identify a product. This is achieved by scanning a product with the RC522 RFID reader. This gives a unique ID (UID) stored within the RFID tag. The

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

UID is sent to the ESP32-WROOM-32UE microcontroller. The UID is then processed and matched to the relevant product information from the backend.

Once the product has been identified, the name, price, and expected weight can be displayed. The controller will also update the shopping cart and calculate the new total. All of this is sent via Wi-Fi to the billing unit in the backend. The backend stores all product information and records within the relational database.

The load cell is attached to the cart so that the total weight can be detected. The load cell produces a very small signal so this must be amplified. This is done with the HX711 module which conditions the signal and gives digital readings to the ESP32. This weight is then compared to the expected weight.

A QR code is used during the checkout process. Once paid, the payment status and cart can be linked to records in the backend. At the time of writing, there is no real-time UPI settlement.

Finally, the exit monitoring unit can be accessed via Wi-Fi. If any suspicious settings have been set up, the buzzer and red LED can be triggered. These are for monitoring purposes only and cannot guarantee they will prevent people from exiting without authorisation.

Overall, AutoCart makes use of many different components including hardware, embedded systems, communication with databases, billing, and security.

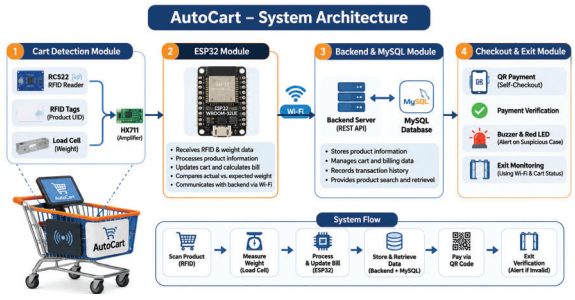

Fig. 1. System Architecture of the AutoCart Smart Shopping Cart

A. Hardware–Software Architecture

The system comprises a cart-side firmware layer, a backend service layer, a relational data layer, and a web-application layer. On the cart, the MFRC522 reader scans and identifies RFID tags, while the ESP32-WROOM-32UE controls the devices. The HX711 module processes and converts the load-cell output. The firmware includes routines for the TFT display, RGB LED, buzzer, Wi-Fi connection, and HTTP communication. The cart transmits RFID and weight data to the backend in defined request formats.

FastAPI is used to implement the backend. Request data are validated through schemas, while services and repositories provide separate application and data-access layers. When an RFID event is received, the RFID service standardises the UID and verifies whether the associated product is active. It then locates the cart session and checks whether the same RFID identifier has already been recorded. A valid new product is added to the cart and an event record is created. The weight service compares the anticipated cart weight with the latest measured weight and returns a status classification.

The frontend is developed using React, TypeScript, and Vite. It provides pages for product display, cart management, checkout, payment demonstration, and transaction confirmation, together with an administrative interface. The web application communicates with the backend through HTTP API requests. Backend services access the database through SQLAlchemy. PostgreSQL is configured for the supplied deployment environment, while SQLite is available for local development and testing.

Figure 1 represents the logical design inferred from the supplied source files. It is not a detailed electrical schematic and does not show the physical interconnections between hardware components. The circuit diagram should remain the reference for hardware connections, subject to verification that its pin assignments match the firmware loaded onto the physical ESP32 device.

IV. Four Main Modules

AutoCart is made up of four different modules. These modules work together to identify, verify, charge and monitor the checkout of products. Each module has its own function but shares important data.

A. Cart Detection Module

This module detects information about the products and their weight. This will include an RC522 RFID reader, RFID tags, and Load Cell. Once a product has been scanned by the customer, the RFID reader will detect the unique UID of the product. The load cell will continuously detect the weight of the cart and send this data to the HX711 module.

B. ESP32 Module

The main processor of the product will be the ESP32-WROOM-32UE. It will receive RFID data and process this data. It will also receive product identifiers, update the cart and calculate the new bill. The digital weight will also be received and compared to the expected weight of the cart. Wi-Fi will be used to communicate between the controller and backend.

C. Backend & Database Module

This module will deal with product information, charging and transaction history. The supplied project uses SQLAlchemy with PostgreSQL configured for the deployment environment and SQLite support for local development. The backend will allow searching for products and keeping track of updated cart information.

D. Checkout & Exit Module

This module will assist with the final part of the shopping journey. It will display the QR payment stage. It will check if any payment information is available. With the Wi-Fi connection it can take into consideration the cart and transaction information. A Buzzer and Red LED will be used to provide

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

alerting when certain suspicious conditions have been met. Live UPI integration has not yet been completed.

The table shows what each module is responsible for and what evidence we currently have.

TABLE II
AUTOCART MODULES AND EVIDENCE STATUS

Module	Responsibility	Status from Project Presentation
Cart sensing	RC522 RFID scans; load cell and HX711 measure cart weight	Prototype measurements reported
ESP32 processing	Product identification, bill updates, and weight comparison	ESP32-WROOM-32UE identified as controller
Backend and database	Product lookup, billing information, and transaction records	SQLAlchemy models and PostgreSQL/SQLite configuration
Checkout and exit	QR payment workflow, Wi-Fi monitoring, and buzzer/LED alerts	Live UPI and end-to-end validation pending

E. Operational Workflow and RFID Event Processing

The shopping-cart application uses a registered cart device with an active cart session. When a customer presents a tagged product to the MFRC522 reader, the ESP32 reads the tag UID and sends an RFID event to the backend over Wi-Fi. The firmware includes a reader-side delay to reduce repeated scans within a short interval. The backend then standardises the UID and verifies whether it corresponds to an active product.

If the UID does not correspond to an active product, the event is rejected. If the same physical UID has already been scanned into the active cart session, the backend ignores the event and does not increment the product quantity. When a valid new UID is received, the backend adds the product to the cart and records the relevant product and price information. Cart quantities, subtotal, total amount, and anticipated weight are then updated. Event logging provides an additional record of cart activity.

Anticipated cart weight is derived from product master data and the quantities of items in the cart. The measured weight is obtained through the load-cell interface and sent to the backend by the ESP32. The backend stores the latest measurement and analyses the difference between the measured and anticipated weights. Updated cart and checkout information can then be displayed on the customer interface based on the backend response.

The backend event-processing sequence is summarised as follows:

1) Receive the device identifier and RFID UID.
2) Normalize the UID and validate the associated active product.
3) Retrieve or create the open cart session for the device.
4) Check whether the UID has already been added to the current session.
5) Reject an unknown UID or ignore a duplicate UID.
6) Add a valid new product, update cart quantity, subtotal, total amount, and expected weight.

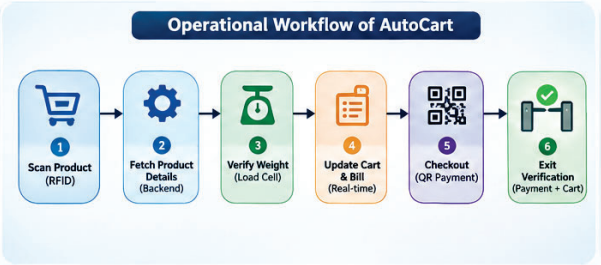

Fig. 2. Operational Workflow of the AutoCart Smart Shopping Cart

7) Record the cart event and return the updated session information.

V. CART DATA AND WEIGHT VERIFICATION

A useful feature included within the AutoCart application is the weight verification. This allows an extra method to verify the contents of the cart by comparing the expected weight with the real weight.

Given the quantity of product i is q_i, the weight per unit of product i is w_i, and the price per unit of product i is p_i, then the total expected weight of the cart and cost of the cart is given by:

$$W_{\text{exp}} = \sum_i q_i w_i, \qquad C_{\text{total}} = \sum_i q_i p_i. \tag{1}$$

Where W_{exp} is the expected weight of all the products and C_{total} is the cost of the items selected. This will update every time a new item is added to the cart or if the quantity of an item has changed.

The real weight of the cart can be found from the load cell through the HX711 module. This gives a value of W_{meas}. The absolute difference can be found by:

$$\Delta W = |W_{\text{meas}} - W_{\text{exp}}|. \tag{2}$$

This can then be compared to a set tolerance to help identify any issues with the weight. If the difference is larger than the tolerance then the system may ask for confirmation. The tolerance must take into account the offset of the sensor, different product weight, movement of the cart, and how the cart is handled.

The project report shows that the HX711 had a calibration factor of 20.653 and there was a measured error of 0.812%. These were observations from prototype data. There is no information on how the sensor was fully calibrated, how many samples were taken, what the measuring range was, and what the tolerance of uncertainty was. This means the error value should not be considered as guaranteed.

While this is a useful second check to the RFID identification, it can show when there are issues with the information being recorded about the products and the measured mass. However, it cannot specifically identify that a product has been swapped.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

A. Backend Weight-Status Classification

The backend implementation defines two configurable weight tolerances. Let W_{exp} denote the expected weight calculated from the registered products and let W_{meas} denote the latest measured cart weight. The absolute difference is

$$\Delta W = |W_{\text{meas}} - W_{\text{exp}}| . \tag{3}$$

The supplied backend configuration uses a match tolerance of 100 g and a security tolerance of 150 g. These values are software policy thresholds and must not be interpreted as a measured accuracy specification for the load cell. The backend classifies a measurement according to Table III.

TABLE III
BACKEND WEIGHT-STATUS CLASSIFICATION

Status	Weight condition	Backend handling
MATCH	$\Delta W \leq 100$ g	Normal status.
MISMATCH	$100 < \Delta W \leq 150$ g	Warning; checkout is permitted by the current service logic.
SECURITY_ALERT	$\Delta W > 150$ g	Checkout is blocked by the current service logic.
UNKNOWN	No measured value is available	Measurement status is unknown; the current checkout service must be checked for the intended missing-measurement policy.

The weight-status routine first checks whether a valid measurement is available. When a measurement exists, it computes the absolute difference and applies the two thresholds. A MATCH result indicates that the difference is within the match tolerance. A MISMATCH result indicates a difference above the match tolerance but not above the security tolerance. A SECURITY_ALERT result indicates that the difference exceeds the security tolerance. In the supplied cart service, a SECURITY_ALERT blocks the checkout transition. The software also defines an UNKNOWN status for the case in which no trustworthy measurement has been received; the missing-measurement policy should be made consistent across the device, backend, and checkout path before deployment.

The firmware in the main project directory contains a sampling routine that collects up to ten weight samples, rejects values outside its configured plausible range, computes an average from the accepted samples, and sends a weight update when the change from the previous stable reading exceeds a configured change threshold. The separate Arduino sketch supplied with the project is an additional firmware variant and contains different pin assignments and a different calibration constant. Consequently, the final calibration factor and pin map must be taken from the exact firmware binary used during physical testing, rather than inferred by combining the two source variants.

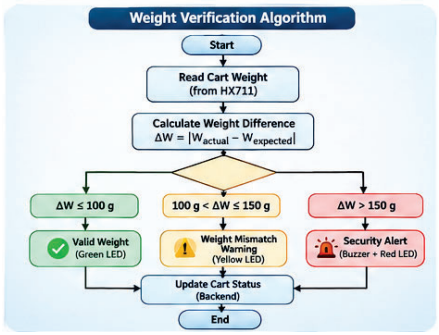

Fig. 3. Weight Verification and Status Classification Flowchart

VI. SOFTWARE AND TRANSACTION FLOW

This controls the flow of the software application by using RFID scanning to identify products. Once they have been identified, the cart will be updated along with the weight, payment, payment status confirmation, and exit monitoring. The ESP32 acts as the main control unit during communication with hardware.

The expected shopping process will begin by scanning the product with an RFID tag. The RC522 reader will then get the unique UID from the product. The UID will then be sent to the ESP32 to query the backend for product information. Once the product has been identified, the cart will be updated with the product details and the total bill will be calculated. The cart will also be updated with the expected weight based on the quantity and weight of the product. The actual weight of the cart will be measured using the load cell. The HX711 module will convert the sensor readings into digital readings. The ESP32 will compare the actual weight to the expected weight.

The cart details and total amount will be displayed using the display and the backend payment system. Product and payment details will be recorded within the relational database. During checkout, the customer will receive a QR payment page. The payment status will be known.

The monitoring system will use the cart and payment status information available. If any suspicious conditions have been met based on the configuration, then the buzzer and red LED will alert the user. This confirms that once a product has been identified it will verify the weight, pay for the product, check the payment, and monitor the exit. Although the QR payment page is included in the project presentation, there is no live UPI connection. This means that the prototype currently does not confirm that a real payment has taken place. Also, the exit monitoring system has not been tested to confirm how effective it is or how many payments it can detect.

A. Backend, Database, and Web Application

The application uses FastAPI to provide REST APIs for product retrieval, cart-session management, device events, weight updates, and payment-session operations. Pydantic schemas validate incoming data and structure responses. Service and repository classes separate application rules from

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

database operations, while SQLAlchemy supports database interaction.

The object-relational data model contains nine principal tables: products, inventory, cart devices, cart sessions, cart items, cart events, orders, order items, and payments. Product records contain RFID identifiers and product information such as price and registered weight. Cart-session records maintain session state, anticipated and measured weights, weight status, and billing totals. Cart-item and order-item records preserve product, quantity, price, and weight information. Event and payment records maintain a history of cart activity and simulated payment interactions.

The supplied environment uses an asynchronous PostgreSQL connection. SQLite is also supported for local development and testing. The database configuration therefore depends on the environment in which the application is executed. The supplied code does not establish a MySQL deployment.

The frontend is implemented using React, TypeScript, and Vite. It allows customers to browse products, manage a cart, and proceed to checkout. It also contains a mock payment section, a payment-success page, and an administrative interface. The frontend and backend communicate through HTTP, while the backend manages cart and order data.

B. QR-Based Demonstration Payment

A mock QR payment method is included in the project. After checkout, an order is generated and the frontend requests a payment session from the backend. The response includes a payment-session identifier, amount, session expiry information, and links for the payment and QR pages. The QR code is generated from the mock payment-session identifier; it is not a UPI address and does not initiate a genuine banking transaction.

The payment interface identifies the process as a demonstration and provides a *Complete Demo Payment* action. When selected, the frontend sends simulated payment data to the payment-validation API, indicating a successful demo payment, after which a payment-success page is displayed. No real bank account, card, UPI, or external payment-gateway transaction is completed. The frontend also contains a demonstration fallback that can navigate to the confirmation page if payment validation fails. Therefore, viewing the payment-success page alone does not prove that a payment record was successfully verified by the backend. The payment function is consequently described as a mock QR-checkout workflow rather than a live digital-payment integration.

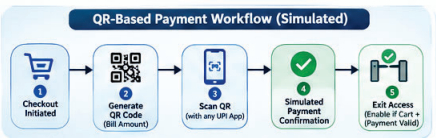

Fig. 4. QR-Based Demonstration Payment Workflow

VII. Implementation Status

The AutoCart prototype comprises the components shown below. Both hardware and software were implemented to help automate the shopping cart process. The hardware components include ESP32-WROOM-32UE microcontroller, RC522 RFID Reader, RFID tags, Load cell, HX711 conditioning module, LCD, buzzer, and red LED. These components are used to identify products, weigh them, display them, and display alerts locally.

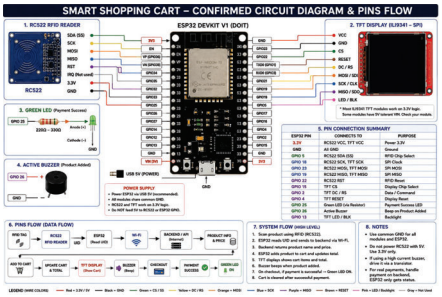

Fig. 5. Circuit Diagram and Pin Connections of the AutoCart Smart Shopping Cart

The ESP32-WROOM-32UE acts as the main controller and allows communication between the sensors and backend. The RC522 RFID reader is used to read the ID of RFID tags. The load cell and HX711 allow the weight of the cart to be determined. The LCD will display information about the cart and billing.

The software components include the following: firmware for the ESP32, product identification, billing, weight verification, communication with database, checkout using QR codes, and exit monitoring. The cart connects to the backend using Wi-Fi to allow billing and transactions to occur. The supplied project uses SQLAlchemy with PostgreSQL configured for deployment and SQLite support for local development.

These components were implemented to create a smart shopping cart system. The ESP32 will process data from the sensors and interact with the backend. The database can be used to manage product and billing information. The checkout and exit monitoring components should allow verification of payments.

Component-level electrical and software specifications should be verified against the respective manufacturers' technical documentation. The research references in this paper provide application context rather than replacing hardware datasheets. Although the system contains the components listed above, there is no evidence that it has been fully implemented in a real store or integrated with real payment services.

VIII. Evaluation & Results

In general the prototype was tested with the data provided within the project presentation. The results have been collected below to showcase the accuracy of the RFID sensor, scan

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

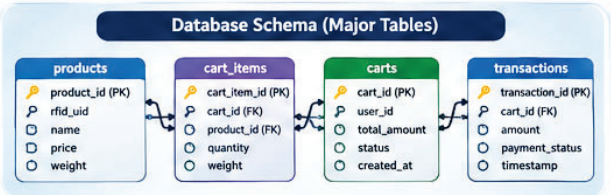

Fig. 6. Database Schema of the AutoCart System

to screen time and load cell results. Overall these provide a general overview of how the prototype performed (Table IV).

TABLE IV
RESULTS REPORTED IN THE AUTOCART PRESENTATION

Measure	Reported Result
RFID detection	96% across 50 trial scans; target 95%
Scan-to-display response	1.6 s average; target 2 s
HX711 calibration factor	20.653
Observed HX711 error	0.812%
Exit monitoring	Wi-Fi-based monitoring listed as a security layer; no detection-rate result provided

The RFID sensor had a very accurate result of 96% over 50 scans. This was only 1% over the target of 95% we wanted to achieve in the project. This shows that we achieved our target for the RFID sensor component. The scan to screen time had an average of 1.6 seconds. This was less than the desired 2 seconds.

The calibration factor for the weight was 20.653. The HX711 had an average error of 0.812% which shows how accurate the sensors were. However there was very little information provided on how the weight was calibrated. There wasn't much information on how many measurements were taken or what range they were. This means we cannot be sure if the measurements were consistent throughout different cases.

The results provided are based on recorded data. There is no information provided on individual test results, confidence intervals or independent tests. There is also no information provided on the Wi-Fi exit detection detection rate.

The live integration with UPI, full testing and store pilot hasn't been completed. In the future testing should be carried out with the RFID scanner at different distances and angles, false/missed readings, calibration of load cell across different weight ranges, network disruption, changing payment status and multiple exit warnings. This will provide a more accurate evaluation of how reliable and suitable AutoCart would be within a retail environment.

A. Evaluation Metrics and Reproducibility

Prototype evaluation should distinguish sensor measurements, software-policy checks, and application workflow demonstrations. RFID detection rate, scan-to-display response time, and load-cell measurement error are sensor-level metrics. Software assessment includes duplicate-scan handling, cart-state transitions, weight-status classification, and payment-session handling. These categories should be assessed separately because they represent different system properties.

The RFID detection rate is calculated as

$$R_{\mathrm{RFID}} = \frac{N_{\mathrm{successful}}}{N_{\mathrm{attempted}}} \times 100\%, \tag{4}$$

where $N_{\mathrm{successful}}$ is the number of successful tag detections and $N_{\mathrm{attempted}}$ is the total number of scan attempts. For response-time evaluation, the elapsed time for trial i is $t_i = t_{\mathrm{display},i} - t_{\mathrm{scan},i}$. The average response time over n trials is

$$\bar{t} = \frac{1}{n} \sum_{i=1}^{n} t_i. \tag{5}$$

For a reference-weight test, the percentage measurement error is

$$E_{\mathrm{weight}} = \frac{|W_{\mathrm{meas}} - W_{\mathrm{ref}}|}{W_{\mathrm{ref}}} \times 100\%, \tag{6}$$

where W_{ref} is the reference mass measured using a suitable calibrated weighing instrument. The reference value, test load, number of repetitions, and calibration procedure should be recorded for each experiment.

The current manuscript reports an RFID detection rate of 96% across 50 scans, a mean scan-to-display response time of 1.6 s, and an observed HX711 error of 0.812%. These values are retained as results previously reported in the project presentation and manuscript. The supplied source archive does not contain the per-scan RFID log, timestamp dataset, or raw reference-weight measurements required to independently recompute the values. Original laboratory records should therefore be retained to support these results before submission. The calibration constant also differs between the supplied firmware variants, so the value used during physical testing must be confirmed.

B. Software-Level Verification Evidence

The source archive contains test cases for product lookup, cart creation, inventory operations, checkout validation, weight-status classification, duplicate RFID handling, new cart-session creation, and duplicate-payment idempotency. These tests represent the software checks implemented in the codebase. However, the archive does not include a complete test-execution report; therefore, this paper does not report a software pass percentage or code-coverage percentage.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

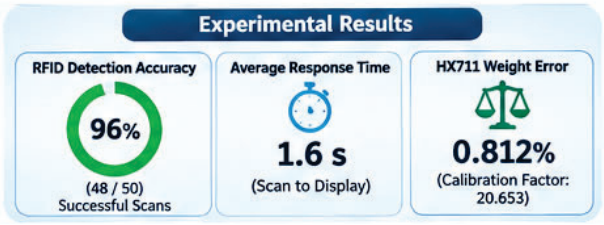

Fig. 7. Experimental Results of the AutoCart Prototype

TABLE V
EVIDENCE AVAILABLE IN THE SUPPLIED AUTOCART PROJECT ARCHIVE

Subsystem	Evidence found in the source archive	Evidence limitation
RFID processing	MFRC522 firmware code; UID normalization; active-product lookup; duplicate UID handling; reader cooldown.	No raw per-scan log was included for independent recalculation of the reported detection rate.
Weight processing	HX711 sampling code; backend expected-versus-measured comparison; configurable 100-g and 150-g status thresholds.	No raw calibration sheet or repeated reference-weight dataset was included.
Cart and database	FastAPI routes, SQLAlchemy models, cart-session services, event records, and relational schema.	Deployment and database environment should be stated consistently with the environment used for testing.
Web application	React/TypeScript pages for products, cart, checkout, payment demonstration, and confirmation.	A browser demonstration does not establish retail-site performance.
Payment	Mock payment-session creation, QR-image display, and a demo verification action.	No live UPI, bank, card, or payment-gateway settlement.
Automated tests	Test code for cart, inventory, weight classification, duplicate RFID, session lifecycle, and payment idempotency.	Test source files are present; no complete execution report was supplied.

The reported sensor results provide an initial indication of prototype operation, but the evidence available in the archive is not sufficient to establish repeatability across different tag orientations, reading distances, product arrangements, cart loads, or network conditions. A reproducible evaluation should preserve raw observations and state the firmware version, calibration factor, sensor configuration, reference instrument, number of trials, and test environment. These details are necessary for an independent reader to repeat the measurements.

IX. DISCUSSION

The AutoCart implementation combines cart-side identification and sensing with a server-managed shopping session. RFID tags identify products, while the backend duplicate check ensures that the same UID is not added repeatedly to one active cart session. This server-side check complements the reader-side cooldown: the cooldown limits rapid reader events, whereas the backend check determines whether a UID is already associated with the current cart session.

Weight is used as a supplementary verification mechanism. Anticipated cart weight is derived from product data and the quantities recorded in the cart, while actual cart weight is captured through the load cell. The backend analyses the difference between these values using predefined thresholds. This can highlight a discrepancy between recorded cart contents and measured mass, but it does not identify the product responsible for the discrepancy. It should therefore be treated as an additional consistency check rather than a definitive method of detecting product substitution.

The backend separates API validation, application services, and database access. The database stores product information, cart sessions, cart items, events, orders, and payment records. The web interface supports the customer shopping workflow and the demonstration payment interface. These components demonstrate the integration of cart hardware and software, but do not by themselves establish reduced checkout duration, shorter queues, or improved loss-prevention performance in a commercial store.

The payment workflow demonstrates the sequence from checkout to QR display and confirmation. Since the project uses a mock payment provider, the displayed confirmation represents a simulated state transition rather than an authenticated financial transaction. A production implementation would require a payment provider, server-side verification through a trusted gateway response or webhook, transaction reconciliation, and appropriate security controls.

X. LIMITATIONS AND FUTURE WORK

Several limitations remain in the supplied implementation. First, the source archive contains two firmware variants with different pin assignments and calibration constants. The exact firmware version used for physical testing must be identified, and the circuit diagram, firmware configuration, and reported calibration value must be made consistent. The load cell should be calibrated using known reference masses over the intended operating range, and repeated measurements should be retained for analysis.

Second, the available project archive does not contain raw experimental logs for all values reported in the current manuscript. Further RFID trials should vary tag orientation, distance, and product placement. Weight experiments should cover multiple reference loads and repeated trials. Response-time measurements should use recorded timestamps and clearly define the start and end events. Network tests should consider latency, temporary disconnection, request failure, and recovery of the cart session.

Third, the payment provider is configured as a mock provider. Future work may integrate a real payment gateway, but such integration should include server-side signature or webhook verification and should not rely on a client-side success button. Fourth, the exit-monitoring function requires a defined test protocol and measured detection outcomes before its effectiveness can be quantified. Finally, a controlled retail pilot is needed to evaluate the system with multiple products, multiple carts, real customer interactions, and realistic network conditions.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

XI. Conclusion

This paper presents AutoCart, an RFID-enabled smart shopping cart prototype that combines product identification, cart-session management, weight monitoring, backend data processing, and a demonstrative QR-checkout interface. The implementation uses an MFRC522 RFID reader and ESP32-based cart controller, an HX711 load-cell interface, a FastAPI backend, SQLAlchemy data models, a relational database, and a React/TypeScript web application. Backend functionality includes product validation, duplicate RFID handling, cart updates, weight-status classification, order records, and a simulated payment workflow.

The manuscript reports a 96% RFID detection rate across 50 scans, a mean scan-to-display response time of 1.6 s, and an observed HX711 error of 0.812%. These figures require the original raw test records and calibration details for independent verification. The configured backend weight thresholds represent a software decision policy, while the payment interface remains a simulation without real UPI or banking settlement. Future work will focus on reconciling the firmware variants, performing repeatable sensor experiments, validating the complete cart-to-backend workflow, integrating a real payment gateway if required, and evaluating the prototype under controlled retail conditions.

References

[1] "RFID-Powered Intelligent Shopping Cart: Streamlined Checkout and Automated Billing," in *Proc. 2024 IEEE International Conference for Women in Innovation, Technology & Entrepreneurship (ICWITE)*, 2024, doi: 10.1109/ICWITE59797.2024.10503373.

[2] "Smart Trolley Using Automated Billing Interface," in *Proc. 2022 International Conference on Advancements in Smart, Secure and Intelligent Computing (ASSIC)*, 2022, doi: 10.1109/ASSIC55218.2022.10088393.

[3] "Efficient Shopping: RFID-Powered Cart with Automated Billing System," in *Proc. 2024 Second International Conference Computational and Characterization Techniques in Engineering & Sciences (IC3TES)*, 2024, doi: 10.1109/IC3TES62412.2024.10877636.

[4] "Human Friendly Smart Trolley with Automatic Billing System," in *Proc. 2020 4th International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, 2020, doi: 10.1109/ICECA49313.2020.9297439.

[5] "Design and Implementation of an Intelligent Shopping Trolley Integrated with a Mobile Application," in *Proc. 2023 International Conference on Sustainable Communication Networks and Application (ICSCNA)*, 2023, doi: 10.1109/ICSCNA58489.2023.10370352.

[6] "Enhancing Retail Efficiency: Design and Implementation of Smart Shelf and Trolley Systems Utilizing IoT Technologies," in *Proc. 2024 First International Conference on Technological Innovations and Advance Computing (TIACOMP)*, 2024, doi: 10.1109/TIACOMP64125.2024.00039.

[7] "RFID-Enabled IoT Smart Shopping Cart for Automated Billing and Enhanced Retail Experience," in *Proc. 2025 5th International Conference on Soft Computing for Security Applications (ICSCSA)*, 2025, doi: 10.1109/ICSCSA66339.2025.11171103.

[8] "IoT-Based Smart Shopping Trolley for Automated Billing and Inventory Management," in *Proc. 2025 International Conference on Computing and Communication Technologies (ICCCT)*, 2025, doi: 10.1109/ICCCT63501.2025.11019404.

[9] "RFID-Enabled Smart Shopping Cart for Enhancing Shopping Efficiency and Preventing Theft," in *Proc. 2024 5th International Conference on Recent Trends in Computer Science and Technology (ICRTCST)*, 2024, doi: 10.1109/ICRTCST61793.2024.10578386.

[10] "RFID based Smart Shopping Cart," in *Proc. 2025 5th International Conference on Evolutionary Computing and Mobile Sustainable Networks (ICECMSN)*, 2025, doi: 10.1109/ICECMSN68058.2025.11383392.

[11] "Smart Shopping Cart with Product Navigation and Automated Billing," in *Proc. 2026 4th International Conference on Inventive Computing and Informatics (ICICI)*, 2026, doi: 10.1109/ICICI68773.2026.11580594.

[12] "Smart Shopping Cart using RFID Technology for Automated Billing and Enhanced Retail Experience," in *Proc. 2025 9th International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, 2025, doi: 10.1109/ICECA66444.2025.11382568.

[13] "Smart Shopping Carts: Reducing Checkout Queues with RFID and AWS Facial Recognition," in *Proc. 2024 IEEE XXXI International Conference on Electronics, Electrical Engineering and Computing (INTERCON)*, 2024, doi: 10.1109/INTERCON63140.2024.10833475.

[14] "Smart Trolley System with Automated Product Tracking and Billing," in *Proc. 2025 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*, 2025, doi: 10.1109/CONECCT65861.2025.11306633.

[15] "Secure RFID Authentication Schemes Based on Security Analysis and Improvements of the USI Protocol," *IEEE Access*, vol. 7, 2019, pp. 8376–8384, doi: 10.1109/ACCESS.2018.2890282.

[16] "Analysis on Security and Privacy Guidelines: RFID-Based IoT Applications," *IEEE Access*, vol. 10, 2022, pp. 131528–131554, doi: 10.1109/ACCESS.2022.3227449.

[17] "The Vulnerabilities Analysis and Design of the Security Protocol for RFID System," in *Proc. Sixth IEEE International Conference on Computer and Information Technology (CIT)*, 2006, doi: 10.1109/CIT.2006.180.

[18] "A Metric for Assessing, Comparing, and Predicting the Performance of Autonomous RFID-Based Inventory Robots for Retail," *IEEE Transactions on Industrial Electronics*, vol. 69, no. 10, 2022, pp. 10354–10362, doi: 10.1109/TIE.2021.3128917.

[19] "Item-Level RFID in a Retail Supply Chain With Stock-Out-Based Substitution," *IEEE Transactions on Industrial Informatics*, vol. 7, no. 2, 2011, pp. 362–370, doi: 10.1109/TII.2010.2068305.

[20] "Coordination and Risk Analysis of VMI Supply Chains With RFID Technology," *IEEE Transactions on Industrial Informatics*, vol. 7, no. 3, 2011, pp. 497–504, doi: 10.1109/TII.2011.2158830.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)