Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

ExpiAlert: A Real Time Product Expiry Alert Management System for Small Retail Businesses

Nnaemeka Ugwumba

nnaemeka_ugwumba@uniport.edu.ng

University of Port Harcourt https://orcid.org/0009-0000-2493-9846

Barisua C. Ndeekor

University of Port Harcourt, Choba

Njionye Ndubuisi David

University of Port Harcourt, choba

Research Article

Keywords: expiry date monitoring, inventory management, real time notification, WebSocket, Socket.io, small and medium enterprises, Nigeria

Posted Date: November 3rd, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11570656/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: The authors declare no competing interests.

ExpiAlert: A Real Time Product Expiry Alert Management System for Small Retail Businesses

Barisua C. Ndeekor
Department of Computer Science
University of Port Harcourt
Port Harcourt, Nigeria
clement.ndeekor@uniport.edu.ng

Nnaemeka Kingsley Ugwumba[1]
Department of Computer Science
University of Port Harcourt
Port Harcourt, Nigeria
nnaemeka_ugwumba@uniport.edu.ng

Njionye Ndubuisi David
Department of Computer Science
University of Port Harcourt
Port Harcourt, Nigeria

Abstract

Pharmacies, supermarkets and provision stores in Nigeria still track product expiry dates largely by hand, through periodic shelf checks and paper logs. Expired goods are therefore found late, often after they have been sold or after they have accumulated on shelves, which creates financial loss, regulatory exposure and risk to consumers. This paper presents ExpiAlert, a web based system that keeps a continuous account of every registered product, classifies each item as Fresh, Expiring Soon or Expired against a threshold chosen by the business, and delivers alerts through two complementary channels: a persistent Socket.io connection for users who have the dashboard open, and a scheduled daily email digest for users who do not. The system was built with React, Node.js, Express and MongoDB, follows a three tier architecture with token based authentication and per business data isolation, and was deployed on free cloud hosting so that a small business can use it without capital outlay. The work followed a design science approach, with requirements drawn from interviews at a community pharmacy and a provision store. Fifteen system test cases passed on the deployed instance, in app alerts arrived in about 500 ms after a triggering event, against a target of two seconds, and email digests arrived within two to three minutes in most runs. We discuss the design decisions behind these results, the limits of the evaluation, and the steps needed before wider field deployment.

CCS Concepts

• **Information systems** → **Web applications**; • **Applied computing** → *Business process management*; • **Software and its engineering** → *Designing software*.

Keywords

expiry date monitoring, inventory management, real time notification, WebSocket, Socket.io, small and medium enterprises, Nigeria

1 Introduction

Any business that sells consumable goods carries stock that loses value on a fixed date. For a pharmacy, an expired drug is a direct write off and a possible danger to the patient who takes it. For a supermarket, expired food is a source of complaints, refunds and waste. The World Health Organization has documented how substandard and falsified medical products harm patients and waste household and health system spending, with the heaviest burden in low and middle income countries [30]. At the scale of the food system, the Food and Agriculture Organization has estimated that roughly one third of food produced for human consumption is lost or wasted, with retail among the stages where losses occur [12].

In Nigeria, the sale of regulated food, drugs and cosmetics falls under the National Agency for Food and Drug Administration and Control (NAFDAC), which has statutory power to inspect premises and seize unfit products [8]. A study of community pharmacies in Anambra State found that more than half of the surveyed outlets did not comply with the national guideline for disposing of expired and unused medicines [17]. Disposal is the last step of a longer process. Before a product can be disposed of correctly, someone has to notice that it has expired, and in most small outlets that notice depends on a staff member reading printed dates during a shelf check.

Small and medium enterprises (SMEs) are where this gap is widest. Large retail chains run enterprise resource planning software that tracks batches and dates. Small outlets in Nigeria usually do not, and research on Nigerian SMEs reports that many have not adopted ICT despite its potential benefits [1]. The businesses that lose the largest share of revenue to expired stock are therefore the ones least able to buy tools that prevent it.

This paper reports the design, implementation and evaluation of ExpiAlert, a web application that addresses this problem for small retail businesses. Its central idea is simple. Every product is recorded once, at the point of stock intake, with its expiry date. From then on, the system and not the staff carries the burden of watching the date, and it tells the business owner when action is needed, whether or not the owner is looking at the screen at that moment.

The paper makes four contributions.

(1) A requirements analysis grounded in interviews at two small Nigerian outlets, which identifies continuous monitoring, timely notification, per business isolation and zero direct cost as the properties a usable tool must have.

(2) An architecture that combines a scheduled status computation with event driven push delivery over WebSocket, so that detection and notification are separated but linked within seconds.

(3) A working implementation deployed on free cloud hosting, with a three level expiry classification that each business can tune through a single threshold.

(4) A functional evaluation on the deployed system, including latency observations, security checks for cross business access, and an analytical comparison of push and polling delivery.

2 Background and Related Work

2.1 Perishable Inventory and Expiry Loss

Inventory theory has treated perishability as a distinct problem for decades. Nahmias reviewed the early models in which stock has a fixed or random lifetime and showed how quickly the optimal ordering problem becomes complex once age matters [19]. Bakker, Riezebos and Teunter extended that review to the large body of deterioration models published after 2001 [3]. More recent operations research focuses on what retailers can do once a product is close to its date. Buisman and colleagues showed that discounting items near expiry reduces fresh food waste at retailers [4], and Haijema and colleagues studied discount policies tied directly to

[1]Corresponding author: Nnaemeka Kingsley Ugwumba, nnaemeka_ugwumba@uniport.edu.ng

the remaining days before the expiration date [13]. These models share an assumption that matters for our work: they presume the retailer knows, at any moment, how many days each item has left. In a store that relies on manual checks, that assumption does not hold, and every downstream policy, whether discounting, returning stock to a supplier or removing it from the shelf, starts late.

2.2 Real Time Communication on the Web

A real time system is one whose correctness depends on when a result is delivered as well as on its value [5]. Business web applications fall into the soft real time class: a delay of a few seconds degrades service but does not cause failure. For expiry alerts, the relevant comparison is with manual checks that run weekly or monthly, so a delay of seconds is negligible.

Browsers historically received server updates by polling, where the client asks the server for news at fixed intervals. The WebSocket protocol, standardised as RFC 6455, instead keeps one full duplex connection open so that the server can send data when an event occurs [10]. Ogundeyi and a colleague at the University of Lagos compared WebSocket with polling, long polling, server sent events and Comet and recommended WebSocket for real time web applications because of its lower communication overhead [22]. A large measurement study of the top one million websites nonetheless found that HTTP polling remained more common than WebSocket in practice [18], which suggests that many deployed systems still accept the latency and traffic cost of polling. Socket.io, the library used in this work, builds on WebSocket and adds automatic reconnection, fallback transports and named rooms for addressing groups of clients [27]. Node.js suits this pattern because its event driven, non blocking model handles many open connections in a single process [28].

2.3 Existing Approaches to Expiry Tracking

Systems that track expiry can be grouped into four architectural classes.

Desktop systems on premises run the application and database on one workstation. Expiry information is available only through reports that staff must remember to run. Access is limited to one location and there is no alerting.

Mobile applications for individual users store items on the device and use operating system notifications. Mastorakis and colleagues describe FoodSaveShare, a household application that estimates expiry dates and helps users reduce waste [16]. Tools of this kind work well for a single person but have no notion of a business account shared by several staff.

Web systems that poll keep data on a central server, which allows access from several devices, but the browser has to ask repeatedly whether anything has changed. Alert latency is bounded by the polling interval, and most requests return nothing new.

Sensor based systems attach RFID tags or other sensors to stock and feed readings to a gateway and a cloud dashboard. Sarac and colleagues reviewed the benefits of RFID across supply chains and noted that tag and infrastructure cost remains a major barrier to adoption [26]. For a small pharmacy, buying readers and tags for every batch is not realistic.

Commercial cloud inventory suites form a fifth group. They offer batch and expiry tracking alongside accounting and sales features, but they are priced and designed for established firms, and their breadth adds training cost for an owner who mainly wants to know what is about to expire.

Table 1 summarises how these classes compare on the properties identified in our requirements analysis. No class satisfies all of them at once. That combination, rather than any single feature, is the gap ExpiAlert addresses.

Table 1: Comparison of existing system classes with ExpiAlert

Class	Push	Multi	Cloud	Email	No HW
Desktop	No	No	No	No	Yes
Mobile app	Partial	No	No	No	Yes
Polling web	No	Partial	Yes	No	Yes
Sensor based	Yes	No	Yes	No	No
Commercial suite	Partial	Yes	Yes	Partial	Yes
ExpiAlert	Yes	Yes	Yes	Yes	Yes

Push: server initiated alerts. Multi: several independent businesses with isolated data. No HW: no special hardware needed.

2.4 Adoption of Technology by Small Firms

Whether a small business uses a tool depends on more than its features. The Technology Acceptance Model holds that intention to use a system follows from its perceived usefulness and its perceived ease of use [7], and the later unified model adds facilitating conditions, meaning the infrastructure available to support use [29]. Diffusion of Innovations theory names relative advantage, compatibility, complexity, trialability and observability as the attributes that govern how fast an innovation spreads [25]. Apulu, Latham and Moreton examined why a number of Nigerian SMEs have not adopted ICT and identified several issues that hold them back despite the potential of the technology [1]. Cloud hosting removes the need for a local server and its power supply [2], and a browser interface runs on phones that owners already have. We used these frameworks as design constraints: the system had to be free to try, quick to learn and usable from a phone.

3 Research Method

The study follows design science research, in which the main output is a purpose built artefact evaluated against a defined problem [14]. We adopted the process model of Peffers and colleagues, which moves through problem identification, definition of objectives, design and development, demonstration, evaluation and communication [23].

Problem identification. We held informal structured interviews with the owner operators of a neighbourhood pharmacy and a provision store. In both outlets, expiry checks were an extra duty added to normal shop work rather than a scheduled task, and they were the first duty dropped when customer traffic was high. Expired items were typically found during customer complaints, supplier visits or inspections.

Objectives. From the interviews and the literature we derived six objectives: independent business accounts with product records that include expiry dates; continuous automatic classification of every product; immediate in app alerts together with an email channel; a live dashboard; simple manual registration with an optional barcode field; and deployment that requires only a browser.

Design and development. Development ran in three iterations, each ending with a working increment that could be tested on its own. The first delivered authentication, the data models and the create, read, update and delete operations of the REST API, verified with Postman before any interface work began. The second delivered the React interface against that stable API. The third added the real time and email channels on top of the existing models without changing them.

4 System Design

4.1 Requirements

Table 2 lists the main requirements. Each functional requirement traces to one of the problems found in the interviews, and each is checked by at least one test case in Section 6.

Table 2: Principal requirements

ID	Requirement
FR1	Register a business with name, email, password, phone and address; authenticate with email and password.
FR2	Add, edit, delete, search and filter products with name, category, quantity, unit, expiry date and optional supplier, location, barcode and notes.
FR3	Compute each product's status as Fresh, Expiring Soon or Expired from its expiry date and the business threshold.
FR4	Push an in app alert when a product is added in, or moves into, a warning state.
FR5	Send a daily email listing products that need attention, subject to the business preference.
FR6	Show a dashboard with counts per status, items expiring within seven days and recent alerts.
FR7	Let each business set its warning threshold and email preference.
NFR1	Deliver in app alerts within two seconds of the triggering event.
NFR2	Restrict every business to its own data.
NFR3	Run in any modern browser on desktop or phone, with no installation.
NFR4	Be usable without technical training.
NFR5	Separate routes, controllers, models and middleware to ease maintenance.

4.2 Architecture

ExpiAlert uses a three tier architecture (Figure 1). The presentation tier is a React single page application. The application tier is a Node.js and Express server that holds all business logic, including status computation, alert generation, the Socket.io server and the scheduled jobs. The data tier is a MongoDB database on MongoDB Atlas, reached only through the application tier. Keeping all classification logic on the server gives one authoritative place where a product's status is decided, so the dashboard, the alert feed and the email digest can never disagree.

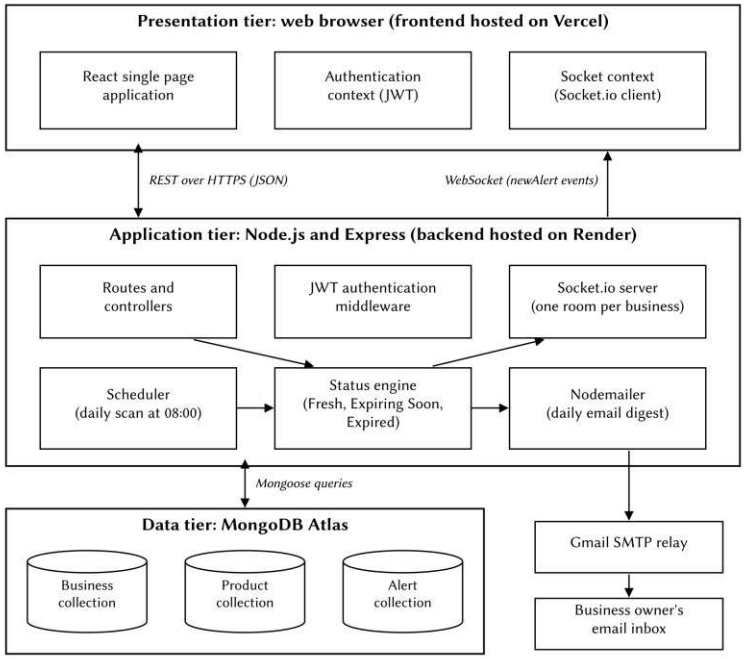


Figure 1: ExpiAlert system architecture.

Two actors use the system (Figure 2). The business owner registers, manages products, sets the alert threshold and views the dashboard. The scheduler acts without human input: it recalculates expiry status, which always includes generating and pushing an alert when a status changes, and it sends the daily email digest. Generating an alert also extends the manage products use case, because a product saved in a warning state raises an alert at once. All business owner use cases except registration and login require a valid token.

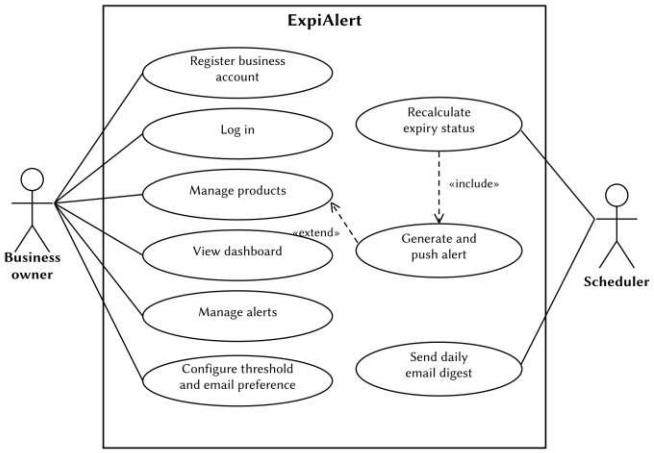


Figure 2: Use case diagram.

4.3 Data Flow

At the context level (Figure 3), the business owner supplies registration details, credentials, product data and settings and receives dashboard views and real time alerts. The scheduler supplies a daily trigger, and the email service carries the daily digest to the owner. Figure 4 decomposes the system into four processes. Store D1 is drawn twice, with the usual duplicate mark, to keep the flows from crossing. Process 1.0 authenticates users against the Business store and issues a token. Process 2.0 manages products. Process 3.0 monitors expiry status, reading expiry dates from D2 and thresholds from D1 and writing to the Alert store whenever a product's status changes. Process 4.0 reads new alerts and delivers them through the socket channel and by email.

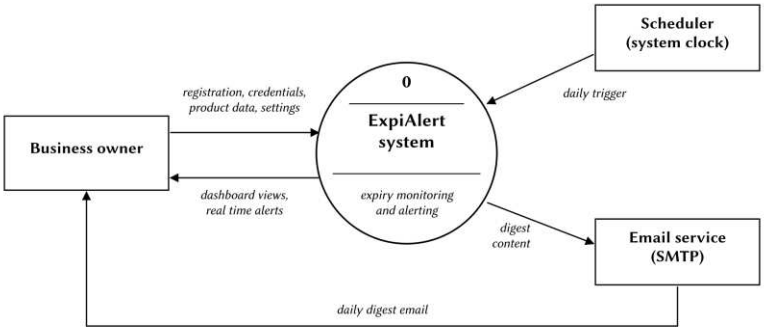


Figure 3: Context level data flow diagram.

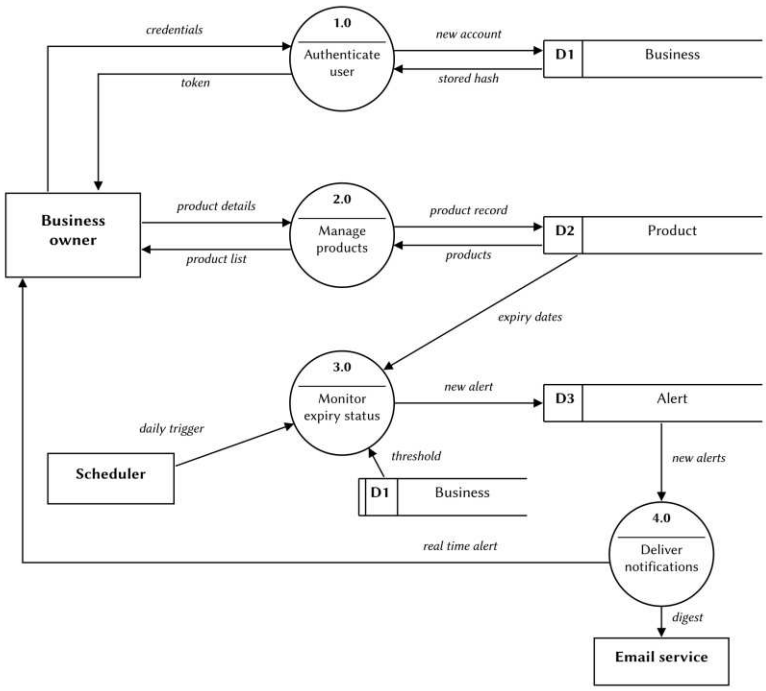


Figure 4: Level 1 data flow diagram.

4.4 Expiry Classification

The classification rule is the core of the system, so we state it precisely. Let Δ be the time remaining from the current moment, in UTC, to a product's expiry date, and let θ be the business threshold in days, which defaults to seven. The number of days left is

$$d = \lceil \Delta \,/\, 86400 \text{ s} \rceil \tag{1}$$

and the status is

$$s(d) = \begin{cases} \text{Expired} & \text{if } d \le 0, \\ \text{Expiring Soon} & \text{if } 0 < d \le \theta, \\ \text{Fresh} & \text{otherwise.} \end{cases} \tag{2}$$

Both boundaries are inclusive in the direction of caution: a product due today counts as expired, and a product exactly θ days away already counts as expiring soon. All dates are stored and compared in UTC, and conversion to local time happens only in the browser, so the result does not depend on the settings of the device used to view it.

Algorithm 1 shows the scan that applies this rule. An alert is created only when the computed status differs from the stored one, which prevents the same product from producing a new alert on every pass. Figure 5 gives the same logic as a flowchart.

Algorithm 1: Status recalculation and alert dispatch

Input: All products P; threshold θ_b for each business b
foreach product $p \in P$ **do**
  $b \leftarrow$ owner of p
  $\Delta \leftarrow$ time from now to $p.expiry$
  $d \leftarrow \lceil \Delta \,/\, 1 \text{ day} \rceil$
  $s \leftarrow s(d)$ using θ_b
  if $s \neq p.status$ **then**
    $p.status \leftarrow s$; save p
    $a \leftarrow$ new Alert(p, b, s, d); save a
    emit *newAlert*(a) to the room of b

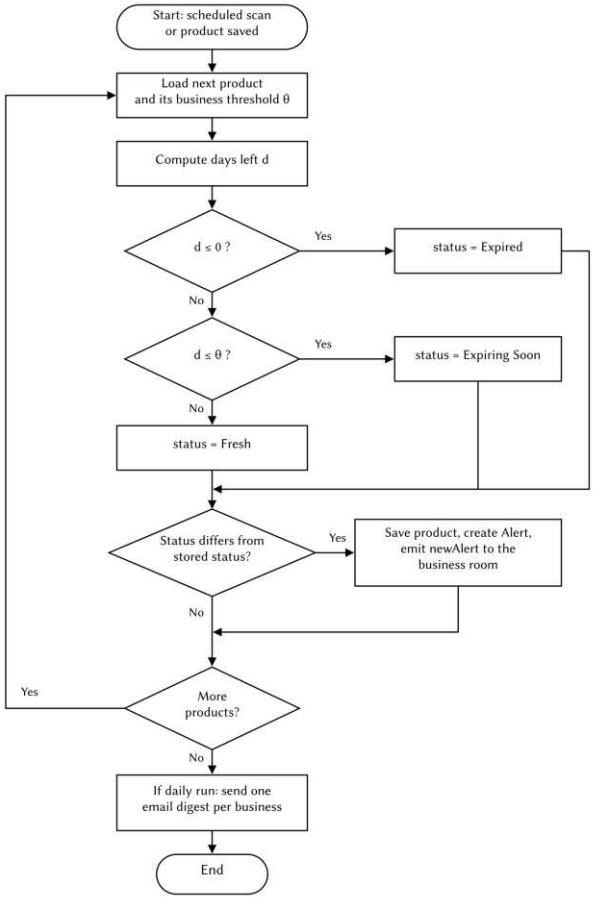


Figure 5: Flowchart of the alert generation process.

4.5 Data Model

The data model has three collections (Figure 6). A business owns many products, and a product may collect many alerts over its life as its status changes. Each alert also stores the business identifier directly, so alerts can be listed per business without a join through products, which suits MongoDB's document model [6]. Table 3 summarises the main fields.

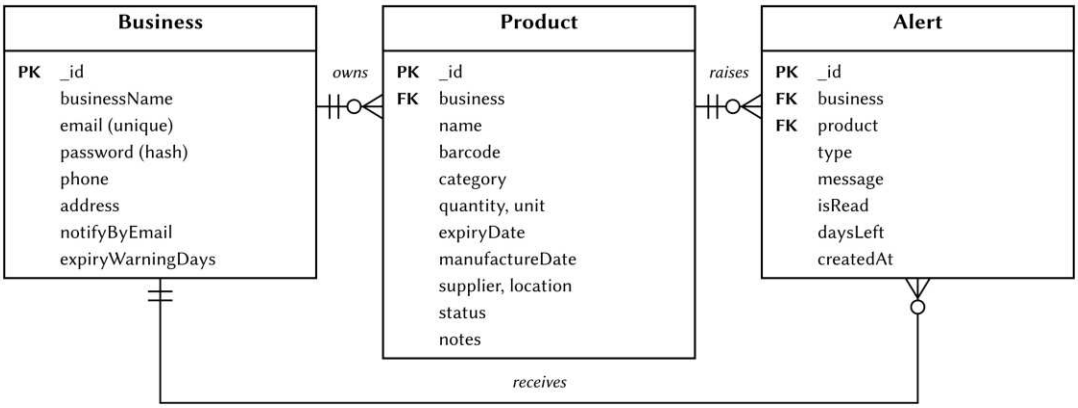


Figure 6: Entity relationship diagram.

Table 3: Main fields of the three collections

Collection	Fields
Business	businessName, email (unique), password (bcrypt hash), phone, address, notifyByEmail, expiryWarningDays, timestamps
Product	business (reference), name, barcode, category, quantity, unit, expiryDate, manufactureDate, supplier, location, status, notes, timestamps
Alert	business (reference), product (reference), type, message, isRead, daysLeft, createdAt

MongoDB was chosen over a relational database for three reasons. Several product fields are optional and fit a flexible document without nullable columns. The relationships are shallow, so the main weakness of document stores, complex joins, does not arise. And documents map directly onto the JSON exchanged by the API and the socket events.

4.6 Alert Delivery

Figure 7 shows how an alert travels. The scheduler starts a scan, the server reads products from the database, computes each status, saves changes, creates an alert and emits a *newAlert* event to the Socket.io room named after the business identifier. Each browser joins its business room after login, so an event reaches every staff member signed in to that business and no one else. The client adds the alert to shared state, and the counters, product list and alert feed update without a page reload.

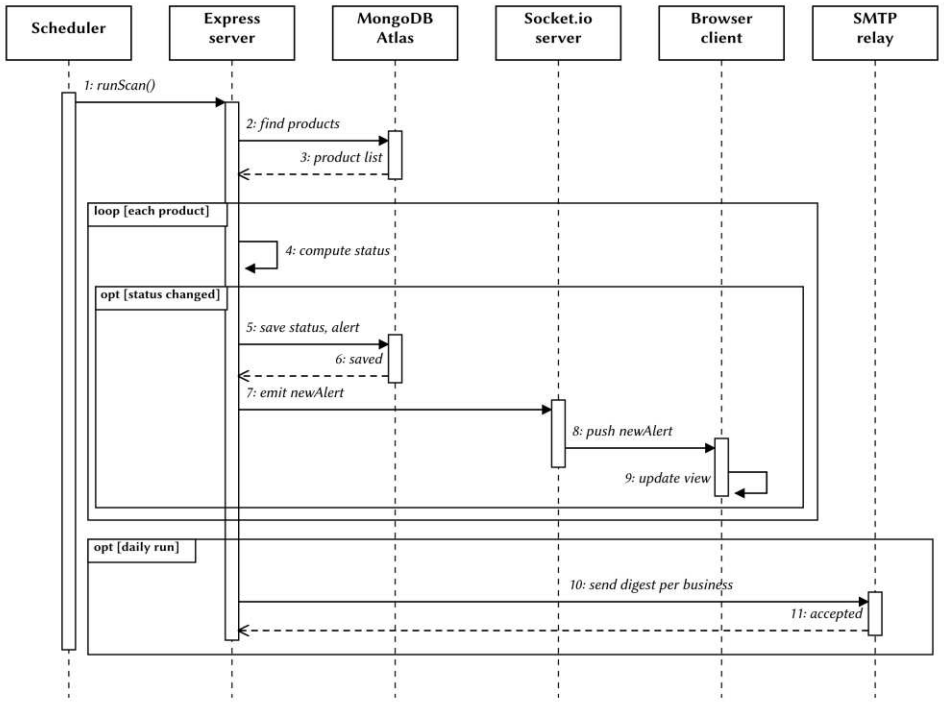


Figure 7: Sequence diagram for real time alert delivery.

The socket channel only helps users who are online. A second, independent path covers the rest: a job runs every day at 08:00, gathers products in a warning state for each business that has email enabled, and sends one digest per business. Combining one immediate channel with one scheduled channel also limits alert fatigue, since routine reminders are grouped into a single daily message instead of one email for each change.

4.7 Security

Passwords are hashed with bcrypt, an adaptive and salted scheme designed to make brute force attacks expensive [24]. After login the server issues a signed JSON Web Token [15] that carries the business identifier and an expiry time, and the client sends it as a bearer token. Middleware rejects any request to a protected route whose token is missing, expired or invalid. Every database query is scoped to the identifier in the verified token, so a user who guesses another document's identifier still cannot read it. Traffic between browser, server and database is encrypted with TLS by the hosting platforms. Because no session state is held on the server, the API can be scaled across several instances later without architectural change.

5 Implementation

Table 4 lists the main components and their versions. The backend follows the REST style [11] and is organised into models, routes, controllers and middleware, as required by NFR5. Table 5 lists the main endpoints. Two further alert endpoints return the unread count and mark all alerts as read.

Table 4: Main software components

Component	Version	Role
Node.js	22.x LTS	Server runtime
Express	4.19.x	Web framework
MongoDB / Mongoose	8.x / 8.4.x	Database and modelling
Socket.io	4.7.x	Real time channel
React	18.3.1	User interface
Vite	5.4.x	Frontend build
Tailwind CSS	3.4.x	Styling
Nodemailer	6.9.x	Email delivery
jsonwebtoken	9.0.x	Token handling
bcryptjs	2.4.x	Password hashing

Table 5: Main REST endpoints

Method	Path	Purpose
POST	/api/auth/register	Create business account
POST	/api/auth/login	Authenticate, return token
GET	/api/auth/me	Current business profile
PUT	/api/auth/settings	Update threshold and preferences
GET	/api/products	List with search and filters
POST	/api/products	Add product
GET	/api/products/:id	Read one product
PUT	/api/products/:id	Update product
DELETE	/api/products/:id	Delete product and its alerts
GET	/api/alerts	List alerts, newest first
PUT	/api/alerts/:id/read	Mark one alert as read
DELETE	/api/alerts/:id	Delete alert
GET	/api/dashboard	Counts and recent items

Status computation on write. When a product is created or edited, the controller applies the classification rule at once. If the new item is already in a warning state, an alert is saved and emitted immediately, so the owner does not wait for the next scheduled scan.

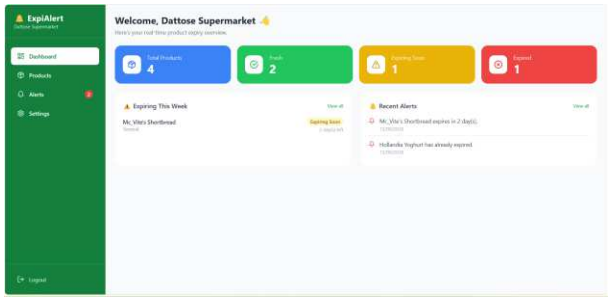
Scheduled scan. A cron style job runs Algorithm 1 daily at 08:00 for all businesses, updates statuses that have changed with the passage of time, avoids creating a second alert for the same product on the same day, emits socket events and then hands the results to the email step.

Email digest. Nodemailer sends an HTML message through the Gmail SMTP relay [21], grouping products by status. The relay required an application specific password rather than the account password, and it imposes a daily sending limit, which is acceptable at current volumes.

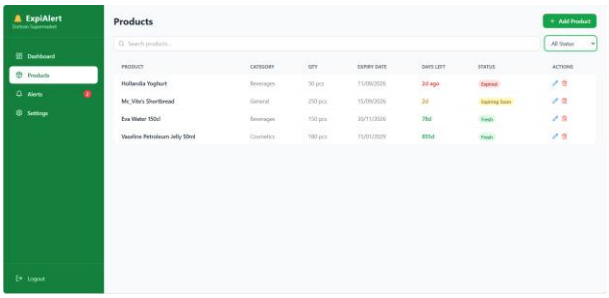
Client state. Two React context providers hold the authentication state and the socket connection. The socket provider joins the business room after login and keeps one listener that updates shared state for every dependent component, which avoided duplicate subscriptions and inconsistent counters.

Interface. The interface follows a traffic light convention: green for Fresh, amber for Expiring Soon and red for Expired. The registration form asks only for the fields needed to begin. Figure 8 shows the main screens of the deployed system.

Deployment. The frontend is hosted on Vercel and the backend on Render, both rebuilt automatically on each push to the main branch of the GitHub repository. The database runs on MongoDB Atlas. Secrets such as the database connection string, the token signing key and the SMTP credentials are kept in each platform's environment settings, not in source control. Every service used runs on a free tier, so the direct monetary cost of operation is zero.



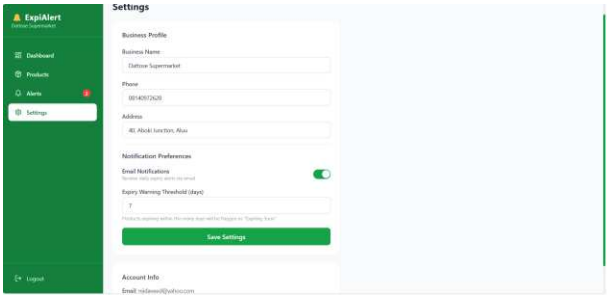
(a) Dashboard



(b) Product list



(c) Alerts



(d) Settings

Figure 8: Main screens of the deployed ExpiAlert system.

6 Evaluation

6.1 Test Procedure

Testing moved from narrow to broad scope. Unit tests targeted the classification function with expiry dates far in the future, inside the threshold window, exactly on the threshold and in the past. Integration tests checked that a product created through the API was stored correctly and that moving a product's expiry date backward, which simulates the passage of days, produced both a database update and a socket event seen by a connected test client. System tests ran on both the local and the deployed environments, and results were compared to confirm that the build and deployment process introduced no change in behaviour. Table 6 lists the fifteen system test cases.

Table 6: System test cases executed on the deployed instance

ID	Case	Input	Expected output	Result
TC01	Business registration	Valid business details	Account created, token returned, dashboard shown	Pass
TC02	Duplicate registration	Email already registered	Message: email already registered	Pass
TC03	Login	Valid credentials	Token returned, dashboard shown	Pass
TC04	Invalid login	Wrong password	Message: invalid email or password	Pass
TC05	Add fresh product	Expiry more than 30 days ahead	Saved with status Fresh	Pass
TC06	Add expiring product	Expiry 3 days ahead	Saved as Expiring Soon, alert shown at once	Pass
TC07	Add expired product	Expiry date in the past	Saved as Expired, alert shown	Pass
TC08	Edit product	New expiry date	Product updated, status recomputed	Pass
TC09	Delete product	Existing product identifier	Product and its alerts removed	Pass
TC10	Dashboard load	Log in and open dashboard	Correct counts shown and kept current	Pass
TC11	Alert management	Mark an alert as read	Alert marked, unread count reduced	Pass
TC12	Email notification	Expiry check with expired items	Email received with product details	Pass
TC13	Search	Text in product name	Matching products listed	Pass
TC14	Filter by status	Status set to Expired	Only expired products listed	Pass
TC15	Settings update	New warning threshold	Settings saved, statuses recomputed on next write	Pass

6.2 Results

All fifteen system test cases passed. Five further observations matter for the requirements.

Latency. In app alerts appeared about 500 ms after the triggering event, well within the two second target of NFR1. Email digests arrived within two to three minutes of the scheduled job in most runs, with occasional delays of up to five minutes that we attribute to processing on the SMTP relay.

Isolation. Requests to protected endpoints without a valid token were rejected with HTTP status 401. A request for another business's products using a valid token from a different account returned HTTP status 403 and no data. A token altered on the client was rejected at signature verification. These checks were repeated after every change to the authentication middleware or query logic.

Boundaries. A product due on the current day was classified as Expired, and a product exactly θ days away as Expiring Soon, matching Equation 2. Products with past dates could be registered on purpose, since businesses sometimes need to record stock already awaiting disposal. A business with no products saw an empty dashboard rather than an error.

Recovery and concurrency. After the network was cut and restored during a session, the Socket.io client reconnected on its own and resumed receiving events. Rapid repeated submission of the product form did not create duplicate records, because the form locks while a request is pending. When two staff members of the same business were signed in at the same time, a product added by one appeared on the other's dashboard without a refresh.

Acceptance. A small business owner unfamiliar with the system was asked to register, add three products with different dates and find the alert for the item closest to expiry, without guidance. All three tasks were completed in one short session, and the participant said that the colour coded status labels made the dashboard easy to read at a glance.

6.3 Push Versus Polling

To place the latency result in context, consider a polling client that checks for news every T seconds. If an event occurs at a uniformly random moment, the mean wait before the client sees it is $T/2$, and each open client sends $86400/T$ requests per day whether or not anything has changed. With $T = 60$ s, that is a mean delay of 30 s and 1440 requests per client per day. With push delivery, the delay is bounded by network and processing time, which we observed at about 500 ms, and traffic is proportional to the number of status changes, typically a few per product over its whole shelf life. This analysis agrees with the comparison of WebSocket and polling reported in [22]. The figures for polling are analytical and were not measured in this study.

7 Discussion

7.1 Interpretation

The results show that the properties the interviews asked for can be delivered together on free infrastructure: continuous classification, immediate notification for users online, a fallback for users offline, and isolation between businesses. The design choice that made this practical was to separate detection from delivery. Detection is a deterministic function of the expiry date, the current time and one threshold, which made it easy to test without waiting for real days to pass. Delivery is event driven, so detection results reach users in under a second. Placing all classification on the server also meant that the three views of status, the dashboard, the alert feed and the email, could not disagree.

The scope was kept narrow on purpose. In terms of Diffusion of Innovations [25], the system aims for low complexity and high trialability: an owner can register and enter products in minutes and pay nothing to find out whether the tool helps. In terms of the Technology Acceptance Model [7], the traffic light scheme and the minimal form are attempts to raise perceived ease of use for staff with limited training. We did not measure these constructs, and claims about adoption remain hypotheses for later work.

ExpiAlert also complements the operations research reviewed in Section 2. Discount and removal policies [4, 13] need an accurate, current count of days left for each item. A system like ExpiAlert supplies that input to a small retailer who could not otherwise compute it, and a natural extension is to suggest a discount when an item enters the Expiring Soon state.

7.2 Threats to Validity and Limitations

Internal validity. Latency was observed during manual testing and not recorded by instrumentation over many runs, so the 500 ms figure is indicative rather than a statistical estimate. Most tests were manual and run by the developers. Automated tests covered the classification function; the API and interface were not covered by an automated end to end suite.

External validity. Requirements came from two outlets, and acceptance testing involved one participant. The results show technical feasibility, not that small businesses in general will adopt

the tool or reduce losses with it. A field study over several months, with records of expired stock before and after adoption, is needed to measure impact.

Construct validity. The system can only be as accurate as the expiry dates entered. It has no way of checking a date against the physical package, and manual entry remains the main cost for a new user with a large existing stock.

Operational limits. The system needs an internet connection. The free backend tier suspends idle services, which delays the first request after a quiet period; a loading indicator and an external uptime ping reduced, but did not remove, this effect. The Gmail relay limits daily volume. A deployment with several server instances would need a shared adapter, such as Redis, so that a socket event raised on one instance reaches clients connected to another. One threshold applies to a whole business, although perishable food and stable cosmetics might reasonably need different warning windows.

Privacy. The system stores business and product data only and collects no customer records, which limits its obligations under the Nigeria Data Protection Act [9]. Business data is still commercially sensitive, which is why isolation between accounts was treated as a priority test.

8 Conclusion and Future Work

This paper presented ExpiAlert, a web system that monitors product expiry dates for small retail businesses and alerts owners through an immediate WebSocket channel and a daily email digest. A precise three level classification, computed in one place on the server, drives both channels. The system was built entirely on open source tools, deployed on free cloud hosting, and passed fifteen system test cases on the deployed instance, with in app alerts arriving in about 500 ms.

Future work falls into four areas. First, data entry: barcode or camera scanning would reduce the time needed to register stock, and the existing optional barcode field already allows for it. Second, reach: SMS alerts and a mobile application with native push notifications would serve owners who rarely check email, and support for Hausa, Igbo and Yoruba would widen access. Third, decision support: per category thresholds, first in first out guidance, discount suggestions for items nearing expiry and reports of losses avoided. Fourth, evidence: a longitudinal field study with a defined group of pharmacies and supermarkets, instrumented latency measurement and a structured usability evaluation based on established heuristics [20]. That study is the step that would turn the technical feasibility shown here into evidence of real reductions in expired stock.

References

[1] Idisemi Apulu, Ann Latham, and Robert Moreton. 2013. Issues of ICT Adoption amongst SMEs in Nigeria. *International Journal of Management Practice* 6, 1 (2013), 58–76. doi:10.1504/IJMP.2013.052317

[2] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. 2010. A View of Cloud Computing. *Commun. ACM* 53, 4 (2010), 50–58. doi:10.1145/1721654.1721672

[3] Monique Bakker, Jan Riezebos, and Ruud H. Teunter. 2012. Review of Inventory Systems with Deterioration since 2001. *European Journal of Operational Research* 221, 2 (2012), 275–284. doi:10.1016/j.ejor.2012.03.004

[4] Marjolein E. Buisman, René Haijema, and Jacqueline M. Bloemhof-Ruwaard. 2019. Discounting and Dynamic Shelf Life to Reduce Fresh Food Waste at Retailers. *International Journal of Production Economics* 209 (2019), 274–284. doi:10.1016/j.ijpe.2017.07.016

[5] Giorgio C. Buttazzo. 2011. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications* (3 ed.). Springer, New York, NY, USA. doi:10.1007/978-1-4614-0676-1

[6] Kristina Chodorow. 2013. *MongoDB: The Definitive Guide* (2 ed.). O'Reilly Media, Sebastopol, CA, USA.

[7] Fred D. Davis. 1989. Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. *MIS Quarterly* 13, 3 (1989), 319–340. doi:10.2307/249008

[8] Federal Republic of Nigeria. 2004. National Agency for Food and Drug Administration and Control Act, Cap. N1, Laws of the Federation of Nigeria 2004. Federal Government of Nigeria.

[9] Federal Republic of Nigeria. 2023. Nigeria Data Protection Act, 2023. Official Gazette of the Federal Republic of Nigeria.

[10] Ian Fette and Alexey Melnikov. 2011. *The WebSocket Protocol*. RFC 6455. Internet Engineering Task Force. doi:10.17487/RFC6455

[11] Roy Thomas Fielding. 2000. *Architectural Styles and the Design of Network-Based Software Architectures*. Ph.,D. Dissertation. University of California, Irvine, Irvine, CA, USA.

[12] Jenny Gustavsson, Christel Cederberg, Ulf Sonesson, Robert van Otterdijk, and Alexandre Meybeck. 2011. *Global Food Losses and Food Waste: Extent, Causes and Prevention*. Food and Agriculture Organization of the United Nations, Rome, Italy.

[13] René Haijema, L. Duijvestijn, Renzo Akkerman, and Frans Cruijssen. 2026. Dynamic Expiration Date-Based Discounting of Fresh Food Products. *International Journal of Production Economics* 292 (2026), 109824. doi:10.1016/j.ijpe.2025.109824

[14] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. 2004. Design Science in Information Systems Research. *MIS Quarterly* 28, 1 (2004), 75–105. doi:10.2307/25148625

[15] Michael B. Jones, John Bradley, and Nat Sakimura. 2015. *JSON Web Token (JWT)*. RFC 7519. Internet Engineering Task Force. doi:10.17487/RFC7519

[16] George Mastorakis, Ioannis Kopanakis, John Makridis, Christina Chroni, Katerina Synani, Katia Lasaridi, Konstadinos Abeliotis, Ioannis Louloudakis, Ioannis N. Daliakopoulos, and Thrassyvoulos Manios. 2024. Managing Household Food Waste with the FoodSaveShare Mobile Application. *Sustainability* 16, 7 (2024), 2800. doi:10.3390/su16072800

[17] Iweh Michael, Brian Ogbonna, Nduka Sunday, Maureen Anetoh, and Okonta Matthew. 2019. Assessment of Disposal Practices of Expired and Unused Medications among Community Pharmacies in Anambra State Southeast Nigeria: A Mixed Study Design. *Journal of Pharmaceutical Policy and Practice* 12, 1 (2019). doi:10.1186/s40545-019-0174-1

[18] Paul Murley, Zane Ma, Joshua Mason, Michael Bailey, and Amin Kharraz. 2021. WebSocket Adoption and the Landscape of the Real-Time Web. In *Proceedings of the Web Conference 2021 (WWW '21)*. ACM, Ljubljana, Slovenia, 12 pages. doi:10.1145/3442381.3450063

[19] Steven Nahmias. 1982. Perishable Inventory Theory: A Review. *Operations Research* 30, 4 (1982), 680–708. doi:10.1287/opre.30.4.680

[20] Jakob Nielsen and Rolf Molich. 1990. Heuristic Evaluation of User Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '90)*. ACM, Seattle, WA, USA, 249–256. doi:10.1145/97243.97281

[21] Nodemailer. 2024. Nodemailer Documentation. https://nodemailer.com/. Accessed October 2026.

[22] K. E. Ogundeyi and C. Yinka-Banjo. 2019. WebSocket in Real Time Application. *Nigerian Journal of Technology* 38, 4 (2019), 1010–1020. doi:10.4314/njt.v38i4.26

[23] Ken Peffers, Tuure Tuunanen, Marcus A. Rothenberger, and Samir Chatterjee. 2007. A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems* 24, 3 (2007), 45–77. doi:10.2753/MIS0742-1222240302

[24] Niels Provos and David Mazières. 1999. A Future-Adaptable Password Scheme. In *Proceedings of the 1999 USENIX Annual Technical Conference, FREENIX Track*. USENIX Association, Monterey, CA, USA, 81–91.

[25] Everett M. Rogers. 2003. *Diffusion of Innovations* (5 ed.). Free Press, New York, NY, USA.

[26] Aysegul Sarac, Nabil Absi, and Stéphane Dauzère-Pérès. 2010. A Literature Review on the Impact of RFID Technologies on Supply Chain Management. *International Journal of Production Economics* 128, 1 (2010), 77–95. doi:10.1016/j.ijpe.2010.07.039

[27] Socket.IO. 2024. Socket.IO Documentation, Version 4. https://socket.io/docs/v4/. Accessed October 2026.

[28] Stefan Tilkov and Steve Vinoski. 2010. Node.js: Using JavaScript to Build High-Performance Network Programs. *IEEE Internet Computing* 14, 6 (2010), 80–83. doi:10.1109/MIC.2010.145

[29] Viswanath Venkatesh, Michael G. Morris, Gordon B. Davis, and Fred D. Davis. 2003. User Acceptance of Information Technology: Toward a Unified View. *MIS Quarterly* 27, 3 (2003), 425–478. doi:10.2307/30036540

[30] World Health Organization. 2017. *A Study on the Public Health and Socioeconomic Impact of Substandard and Falsified Medical Products*. World Health Organization, Geneva, Switzerland.