IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

DMilkRoute: A Mobile Milk Collection and Vending Management System

P Charan Kumar Reddy
Dr.A Rajesh
Department of Computer Science and Engineering (Data Science)
Jain University, Bengaluru, India
Jupg24mtech23387@jainuniversity.ac.in
a.rajesh@jainuniversity.ac.in

Abstract

Rural milk collection in India remains a surprisingly manual process, even today. Agents still carry paper ledgers, quality readings are often verbal, and farmers wait days for payments they have no way to verify. This work focuses on MilkRoute, a practical system built to address these real gaps in rural dairy operations. It provides a web-based interface where a single collection agent can manage an entire shift across four routes, covering multiple villages and handling over 130 registered farmer accounts from one screen.

At its core, MilkRoute reads quality parameters — FAT%, SNF%, protein, and temperature — directly at collection time and grades the milk automatically into tiers (A, B, or C), computing each farmer's payout on the spot. Payments go out instantly over UPI to individual farmer accounts, with every transaction logged for later review. No more waiting days for a cooperative to manually calculate what each farmer is owed.

Beyond collection itself, the system handles route tracking with village-wise stop maps and arrival time estimates, monitors connected field devices like flow meters and weight scales, and raises alerts whenever something goes wrong — a quality failure, a skipped stop, a device going offline. It tries to give the agent one place to look instead of juggling several separate tools.

The whole thing runs as a single HTML file — no server, no installation, no IT support needed. Built with HTML5, CSS3, and vanilla JavaScript, it works on a basic browser in a low-connectivity village setting. The prototype shows that a field-ready dairy digitization tool does not have to be complicated to be genuinely useful.

Keywords: dairy management, milk quality analytics, UPI payment integration, FAT/SNF grading, rural fintech, progressive web application, farmer digitization, supply chain transparency

1. Introduction

India produces more milk than any other country in the world. Over 80 million farm households are part of this industry, most of them small farmers in rural areas who depend on daily milk sales for income. But if you visit an actual collection point early in the morning, what you see is not very different from what it looked like decades ago — handwritten ledgers, cash changing hands, quality readings communicated verbally, and farmers with no real way to verify what they are being paid or why. The scale of the industry and the conditions at its base level are almost impossible to reconcile.

The problems are not hard to find. Farmers regularly suspect that FAT% and SNF% readings have been entered incorrectly — sometimes accidentally, sometimes not. Payment arrives days or weeks late, and there is rarely any record the farmer can refer back to. Routes are planned loosely, collections get missed, and when a device breaks down in the field, the news travels slowly through an informal chain. None of this is unusual or specific to one cooperative; it is simply how the system tends to work when everything runs on paper and verbal communication.

There have been attempts to use technology here. Tablet-based collection terminals, SMS payment alerts, Bluetooth-connected milk analyzers — various pieces of the puzzle exist. The problem is that they tend to exist separately. One tool handles quality readings, another manages payments, a third tracks routes. The agent still has to bridge the gaps manually. In talking to people familiar with these deployments, the recurring frustration was not the technology itself but the fragmentation — having to switch between systems and re-enter data that should have flowed automatically from one step to the next.

MilkRoute is an attempt to build that unified interface. It is a web-based application that puts collection management, quality grading, UPI payments, route tracking, device monitoring, and alerts all in one place — the idea being that an agent should be able to go through an entire shift without switching tools or re-entering anything. Quality data flows directly into the grading engine, which feeds the payout calculation, which connects to the payment module. The route view shows which stops are done and what's pending. Alerts surface problems before they become bigger problems.

The application only needs a browser and occasional internet for the payment step. Everything else runs locally. The rest of this thesis covers how the system is designed, how each module works, what the testing showed, and what would need to change to move from prototype to something actually deployable in the field.

1.1 Motivation

When I started looking into how rural dairy collection actually works in practice, I expected to find some version of the problem being solved. What I found instead was that most agents are still doing this entirely by hand. A single agent on a morning route might visit five or six villages, collect from eight or ten farmers at each stop, measure volume and quality for each one, calculate a rate, note it down, and do this across 130 or more transactions before the morning is done — all before the milk spoils, which means before around 9 or 10 AM. Doing that accurately on paper, under time pressure, every single day, is genuinely difficult. Errors are almost unavoidable, and when they happen, there is no clean way to trace back what went wrong.

What made this feel worth working on was the financial dimension. For many of these farmers, milk is the only income that comes in daily. A delay of two or three days in payment is not just inconvenient — it can mean borrowing money to cover household expenses that should have been

covered by that payment. UPI makes instant settlement genuinely possible at the village level now, in a way it simply was not five years ago. That felt like a real opportunity to change something concrete for people who do not have much margin for error in their finances.

1.2 Objectives

The primary objectives of this project are as follows:

- To design and implement a unified web-based platform for managing milk collection operations across multiple routes and shifts.
- To develop an automated quality grading engine based on FAT%, SNF%, and protein parameters with configurable pricing tiers.
- To integrate UPI-based instant payment processing with per-transaction records accessible to collection agents and cooperative administrators.
- To implement route management functionality with village-stop sequencing, ETA computation, and map-based visualization.
- To build a farmer profile management module with historical analytics and status tracking.
- To incorporate IoT device monitoring for field hardware including flow meters, temperature sensors, and weight scales.
- To design an intelligent alerting subsystem for quality failures, payment anomalies, device faults, and route deviations.

1.3 Scope and Limitations

MilkRoute is a prototype, and it is worth being clear about what that means. It runs entirely in the browser using localStorage for persistence, which is fine for demonstration but means a cleared browser or a device failure would wipe the data. There is no multi-user login, no server sync, and no real UPI transaction — the payment module simulates the gateway interaction rather than connecting to one. IoT device data is also simulated. These are not design oversights; they reflect the constraints of building a first working version without a registered business entity for UPI onboarding or physical hardware to test against. The architecture is set up to support all of these in a production version without rebuilding from scratch.

2. Related Work

The digitization of dairy supply chains has received growing attention over the past decade, driven by the convergence of mobile connectivity, IoT sensor miniaturization, and digital payment infrastructure. This section reviews relevant prior work across the key functional domains addressed by MilkRoute.

2.1 Dairy Collection Management Systems

Early efforts to digitize milk collection focused on replacing paper ledgers with handheld devices capable of recording volume and quality measurements. Systems deployed by NDDB (National Dairy Development Board) in the 1990s introduced computerized milk collection units (CMCUs) at village cooperative societies, which automated the computation of milk value based on FAT readings from milk analyzers [1]. These systems significantly reduced calculation errors and the potential for manipulation by local agents. However, they were standalone units requiring physical infrastructure at collection points and did not provide real-time data transmission or centralized monitoring.

More recent systems have moved toward tablet and smartphone-based interfaces. Desai et al. [2] describe a tablet-based collection terminal deployed across 200 villages in Gujarat, integrating Bluetooth-connected milk analyzers with an Android application. Their evaluation found a 23% reduction in payment disputes and a 34% improvement in route adherence compared to manual operations. These are meaningful improvements, but their system treated quality testing, record-keeping, and payment as separate modules requiring manual synchronization. In practice, that still means the agent is bridging gaps by hand — which is exactly the kind of friction MilkRoute was designed to eliminate.

2.2 Milk Quality Analytics

Milk quality assessment at the point of collection has traditionally relied on gravimetric methods (Gerber test for fat, lactometer for SNF) which are time-consuming and susceptible to human error. The adoption of infrared-based milk analyzers (such as those using mid-infrared or near-infrared spectroscopy) has made rapid, multi-parameter analysis feasible at collection points. These analyzers can measure FAT%, SNF%, protein, lactose, and density in under 30 seconds, producing digital readings that can be directly fed into collection software [3].

The challenge lies in converting raw analyzer readings into grading decisions and payout computations in a way that is transparent and verifiable by farmers. Kumar and Srinivasan [4] propose a fuzzy logic-based grading model that accounts for measurement uncertainty in low-cost analyzers, assigning probabilistic grade bands rather than deterministic cutoffs. While theoretically sound, such approaches add complexity that may be difficult to communicate to farmers with low digital literacy. MilkRoute adopts a rule-based grading engine with clearly defined FAT/SNF thresholds that are displayed to farmers at the point of collection, prioritizing transparency over statistical sophistication.

2.3 Digital Payment Systems in Rural Agriculture

The introduction of UPI in 2016 and its subsequent penetration into rural India has transformed the feasibility of instant digital payments at the village level. As of 2024, UPI handles over 13 billion transactions per month across India, including significant volumes in Tier 3 and Tier 4 cities [5]. Several agricultural procurement platforms have integrated UPI for instant farmer payments, including commodity platforms for paddy, cotton, and vegetables. Narayanan et al. [6] document a pilot in Maharashtra where UPI-based instant payment at procurement centers reduced average payment delay from 12 days to same-day, with statistically significant improvements in farmer satisfaction and willingness to expand production.

In the dairy sector, cooperative societies have been slower to adopt instant digital payments due to concerns about connectivity reliability, farmer digital literacy, and the administrative overhead of maintaining individual farmer UPI IDs. MilkRoute addresses the administrative dimension by embedding UPI ID management within the farmer profile, making it available automatically at payment time without requiring the agent to look up IDs manually.

2.4 Route Optimization in Agricultural Logistics

Route optimization for milk collection presents a variant of the vehicle routing problem (VRP), specifically the time-constrained VRP where milk must be collected within a narrow temperature window (typically within 3–4 hours of milking). Sharma et al. [7] apply metaheuristic algorithms including genetic algorithms and simulated annealing to optimize collection routes for a medium-sized dairy cooperative in Haryana, achieving a 17% reduction in fuel cost and 22% reduction in total collection time compared to manually planned routes. However, their approach requires significant

computational resources and route re-optimization whenever a new farmer is added or a stop is skipped.

MilkRoute takes a pragmatic approach to route management, providing pre-defined routes with village-stop sequences determined by geographic proximity and historical collection patterns, with visual timeline tracking and ETA display. While this does not provide dynamic re-optimization, it gives collection agents a structured framework for shift planning that is significantly more effective than entirely ad hoc sequencing.

2.5 IoT Integration in Dairy Operations

The integration of IoT sensors in dairy supply chain monitoring has been explored extensively in the context of cold chain management (temperature monitoring during transportation) and herd health monitoring. For milk collection specifically, flow meters integrated with IoT gateways can provide tamper-evident volume measurements, addressing a major source of farmer distrust of manual volume recording [8]. GPS tracking of collection vehicles enables cooperatives to verify route adherence and identify unauthorized stops.

MilkRoute's device management module is designed to interface with this class of field hardware, providing real-time status monitoring and fault alerting for flow meters, temperature sensors, and weight scales assigned to each collection vehicle. The current prototype simulates device telemetry; the architecture is designed for integration with actual MQTT-based IoT devices in a production deployment.

3. Proposed System

MilkRoute is architected as a modular, client-side progressive web application organized around the operational workflow of a milk collection agent. The system comprises seven primary functional modules integrated within a unified responsive interface: (1) Dashboard and Shift Management, (2) Collection Recording and Quality Analytics, (3) Farmer Profile Management, (4) Payment Processing, (5) Route Management, (6) Device Monitoring, and (7) Alerts and Notifications. These modules share a common data model and state management layer, enabling seamless data flow across the workflow without requiring the agent to re-enter information at each step.

3.1 System Architecture

The overall architecture of MilkRoute follows a layered client-side model, as shown conceptually in Figure 1. The presentation layer comprises a multi-page single-page application (SPA) implemented in HTML5, CSS3, and vanilla JavaScript, with Chart.js providing data visualization. The data layer uses browser localStorage for persistence across sessions, with a structured JSON schema for each entity type (farmers, collections, payments, alerts, devices). The integration layer provides interfaces for UPI payment gateway interaction and IoT device telemetry, currently implemented as simulated modules with defined API contracts for production integration.

The system does not depend on a backend server for core functionality, enabling operation in low-bandwidth environments. Network access is required only for UPI payment processing and, in a production deployment, for IoT device data ingestion and cloud-based data synchronization.

3.2 Data Model

The core data model consists of the following primary entities:

- Farmer Profile: Contains farmer ID, name, phone number, village, assigned route, shift preference (morning/evening/both), milk type (cow/buffalo), average volume and FAT percentage, status (active/probation/suspended), monthly earnings, and UPI ID.
- Collection Record: Captures timestamp, shift, route, farmer reference, milk type, volume (litres), FAT%, SNF%, protein%, temperature, computed quality grade, per-litre rate, and total payout.
- Payment Transaction: Records transaction ID, farmer reference, amount, associated collection volume, milk type, UPI gateway identifier, UPI ID, payment status, and timestamp.
- Route Definition: Defines route identifier, description, and an ordered sequence of village stops each with farmer count, estimated volume, ETA, and distance from previous stop.
- Device Record: Tracks device identifier, type (flow meter, temperature sensor, weight scale), assigned route, operational status, last reading, and alert thresholds.
- Alert Record: Captures alert type, severity (critical/warning/info), description, associated entity reference, timestamp, and resolution status.

3.3 Quality Grading Engine

The quality grading engine is a rule-based component that classifies each collection record into a quality grade and computes the corresponding payout. The grading logic is parameterized by milk type, reflecting the different compositional standards for cow and buffalo milk under FSSAI and Indian Standard IS:1479.

For cow milk, Grade A is assigned when FAT% is 4.5 or above and SNF% is 8.5 or above, attracting a rate of ₹34 per litre. Grade B requires FAT% between 3.5 and 4.5 and SNF% between 7.5 and 8.5, at ₹28 per litre. Grade C covers milk below these thresholds, accepted at ₹22 per litre. Buffalo milk commands higher rates reflecting its superior compositional richness: Grade A (FAT ≥ 7.0%, SNF ≥ 9.5%) at ₹50 per litre, Grade B at ₹42 per litre, and Grade C at ₹35 per litre. Milk failing minimum quality thresholds is flagged for rejection with an alert generated for supervisor review.

The total payout for each collection is computed as volume multiplied by the applicable per-litre rate. Monthly earnings are accumulated across all collection records for each farmer within the billing period.

Milk Type	Grade	FAT% Min	SNF% Min	Rate (₹/Litre)
Cow	Grade A	4.5%	8.5%	₹34.00
Cow	Grade B	3.5%	7.5%	₹28.00
Cow	Grade C	< 3.5%	< 7.5%	₹22.00
Buffalo	Grade A	7.0%	9.5%	₹50.00
Buffalo	Grade B	6.0%	8.5%	₹42.00
Buffalo	Grade C	< 6.0%	< 8.5%	₹35.00

Table 1: Quality grading thresholds and rates for cow and buffalo milk

4. Materials and Methods

This section describes the technology stack, implementation methodology, and functional design decisions that underpin the MilkRoute system.

4.1 Technology Stack

MilkRoute is implemented as a single-file web application — a design choice motivated by the need for ease of deployment in rural environments where technical support for multi-component server setups is unavailable. The core technologies are:

- HTML5: Provides the structural markup for a multi-page single-page application with dynamic page switching managed via JavaScript.
- CSS3: Implements a comprehensive design system with CSS custom properties (variables) for theming, including support for system-level dark mode via the prefers-color-scheme media query. The design system defines a green-dominant color palette appropriate for agricultural contexts.
- Vanilla JavaScript (ES2020+): Manages all application state, data operations, UI rendering, and module coordination without framework dependencies. This choice reduces load overhead and eliminates dependency on package registries, which may be inaccessible in low-bandwidth environments.
- Chart.js 4.4.1: Provides canvas-based charting for weekly collection volume trends, milk quality distribution, historical analytics, and farmer contribution rankings.
- Google Fonts (Syne, DM Mono): Provides typography — Syne for display and body text, DM Mono for numerical and code-style data presentation.
- Browser localStorage API: Provides client-side persistent storage for farmer profiles, collection records, payment transactions, and application state across sessions.

4.2 Shift Management Module

The shift management module governs the operational lifecycle of a collection session. A shift is defined as a structured time window — morning (06:00–12:00) or evening (15:00–20:00) — during which collections are recorded for a selected route. The agent initiates a shift by selecting the shift type and route, after which the system activates collection-related UI elements and begins tracking shift duration via a live timer.

The shift start interface displays a contextually styled banner — warm orange-amber gradient for morning shifts, cool indigo-violet for evening shifts — providing an immediate visual orientation for the agent. Shift state is persisted to localStorage, allowing the application to survive browser refresh during an active shift. The shift summary, generated at close-of-shift, reports total volume collected, number of farmers served, total payments made, and average quality metrics for the session.

4.3 Collection Recording Module

The collection recording module is the operational heart of MilkRoute. When an agent adds a new collection entry, a modal dialog presents input fields for farmer selection, milk type, volume, FAT%, SNF%, protein%, and temperature. Farmer selection is filtered by the active route, presenting only farmers registered to the current route. Upon entry, the quality grading engine immediately computes the grade and payout, which are displayed before the agent confirms the entry.

The collection log table presents all entries for the current session with columns for farmer name, village, milk type, date, day, time, shift, route, volume, FAT%, SNF%, protein%, grade, rate, and payout. Each entry provides actions for viewing a printable bill slip (rendered as a styled receipt with all transaction details) and for initiating payment. Filtering and search functions allow the agent to locate specific records within large collection logs.

A historical data panel provides aggregated views of past collection sessions, showing day-wise summaries with total volume, farmer count, average FAT%, percentage of Grade A milk, and total payments. A top contributors analysis panel ranks farmers by monthly collection volume, with drill-down to individual farmer payment histories.

4.4 Farmer Profile Management

The farmer management module maintains a comprehensive registry of all farmers registered with the cooperative. Each farmer profile stores identity information (name, farmer ID, phone, village), operational attributes (assigned route, shift, milk type, average volume and FAT%), financial data (monthly earnings, UPI ID), and status (active, probation, suspended).

The farmer list view provides search and route-based filtering, with tabular display of all profile fields and action buttons for editing, viewing payment history, and suspension management. Adding a new farmer triggers a modal form that captures all required fields; farmer IDs are auto-generated in the format FMR-XXX. The edit flow pre-populates the modal form with existing values, enabling partial updates.

A farmer-specific payment history view shows all transactions associated with the selected farmer, including amounts, dates, and payment statuses. This view is accessible from both the farmer management module and the payment processing module, providing a consistent audit trail.

4.5 Payment Processing Module

The payment processing module manages the disbursement of milk value to farmers via UPI. The module presents a pending payments list showing all collection entries for which payment has not yet been processed, organized by farmer. The agent selects a farmer, verifies the amount and UPI ID, and initiates payment. The payment flow simulates a UPI gateway interaction — displaying a formatted payment receipt with transaction details — and, on confirmation, records the transaction and updates the collection entry's payment status.

The module maintains a complete transaction log accessible via the History tab, searchable by transaction ID and farmer name. Each transaction record includes a unique TXN ID, farmer reference, amount, volume, milk type, gateway identifier, UPI ID, status, date, and time. A bulk payment tool enables the agent to process payments for all pending farmers in a single operation, with configurable skip logic for farmers with disputed records.

A payment analytics panel in the collection view displays aggregate payment statistics including total amount paid in the current session, number of farmers paid, average payment per farmer, and pending payment count. A pending farmer alert in the payment module flags farmers who have collections recorded but have not yet received payment, enabling the agent to resolve outstanding obligations before closing the shift.

4.6 Route Management Module

MilkRoute defines four named collection routes — Route A (Kothapalli to Bukkaraya Samudram), Route B (Ankampeta to Kanampalle), Route C (Bandameedapalle to Gondireddipalle), and Route D (Dadithota to Chinnachigullarevu) — each comprising six village stops with defined farmer counts, estimated collection volumes, ETAs, and distances from the preceding stop. These routes reflect the geographic organization of dairy collection operations in semi-arid agricultural regions of Andhra Pradesh.

The route management view provides a route selector, an SVG-rendered map of the selected route showing stop sequence and relative distances, a timeline panel with per-stop progress tracking, and a collection summary for completed stops. As the agent progresses through the route, stops are marked as completed or skipped, with the route map updating to reflect current progress.

A route-specific collection summary table aggregates all collection entries for the selected route, showing per-farmer volume, grade, and payment status. This provides the agent with a consolidated view of route completion and outstanding payments at any point during the shift.

4.7 Device Monitoring Module

The device monitoring module tracks the operational status of IoT-connected field hardware assigned to each collection vehicle. Device types include flow meters (for volume measurement), temperature sensors (for milk temperature monitoring), and weight scales (as a backup volume measurement method). Each device has a defined operational status (online, warning, offline), last reading timestamp, and alert threshold configuration.

The device management view presents all registered devices with their current status displayed via color-coded status indicators. Devices in warning state (e.g., temperature sensor reading near threshold, flow meter battery low) trigger amber indicators; offline devices trigger red indicators with automatic alert generation. Device status toggles allow the supervisor to mark devices as under maintenance, suppressing false positive alerts during scheduled servicing.

In a production deployment, device data would be ingested via MQTT over a cellular data connection, with the device management module subscribing to device-specific topics and updating local status accordingly. The current prototype simulates device telemetry using pre-configured state data.

4.8 Alerts and Notifications Module

The alerts module consolidates operational anomalies from all other modules into a unified notification center. Alerts are classified by severity — critical (requiring immediate action), warning (requiring attention within the shift), and info (informational) — and by category (quality failure, payment pending, device fault, route deviation, system notification).

Critical alerts include events such as milk quality below minimum thresholds (FAT% or SNF% below rejection cutoff), device offline during active collection, and payment failures. Warning alerts cover conditions such as collections with pending payments at shift close, devices in warning state, and farmers approaching monthly credit limits. Info alerts include shift start/end events, route progress milestones, and system updates.

Alerts are displayed in a filterable list with timestamp, category badge, severity indicator, and a description. Each alert provides an action button linking to the relevant module — for example, a quality failure alert links to the specific collection record; a pending payment alert links to the farmer's payment view. Alert count badges are displayed on the navigation sidebar, providing at-a-glance notification of pending issues without requiring the agent to navigate to the alerts module.

5. Implementation

The MilkRoute system was implemented as a single HTML file of approximately 6,200 lines, encompassing all CSS, JavaScript, and markup. This packaging approach was deliberate, enabling deployment by simply opening the file in a browser with no server setup or dependency installation. The following subsections describe key implementation details for the major functional components.

5.1 Application State Management

Application state is managed through a set of module-level JavaScript variables that are updated by user actions and automatically persisted to localStorage. The primary state variables include the active collection log (collectionLog), registered farmer profiles (farmers), payment transaction history (paymentHistory), active shift state (shiftActive, shiftStartTime, shiftType), and selected route (selectedRoute). All state mutations are wrapped in update functions that trigger re-renders of affected UI components.

Data persistence is implemented using JSON serialization to localStorage keys prefixed by module name (e.g., 'milkroute_farmers', 'milkroute_collections'). On application load, persisted state is deserialized and used to initialize the state variables, providing session continuity across browser refreshes. A data export function serializes all state to a JSON file for backup and sharing with cooperative administrators.

5.2 Farmer Data Initialization

The prototype includes a comprehensive pre-loaded farmer dataset of 137 registered farmers distributed across the four routes and their constituent villages. This dataset reflects realistic operational parameters: farmer IDs in the FMR-001 to FMR-137 range, phone numbers in Andhra Pradesh mobile number formats, villages named after actual settlements in the region, mix of cow and buffalo milk farmers, and earnings computed at plausible market rates. This pre-loaded data enables immediate functional demonstration without requiring manual data entry.

New farmers can be added through the farmer management modal, with auto-incrementing IDs. Imported farmers from CSV upload are validated against the expected schema (FarmerID, MilkType, Volume, FAT, SNF, Temperature) before insertion.

5.3 Collection Bill Generation

Each collection record can generate a printable bill slip rendered as a styled HTML receipt. The bill interface simulates a thermal printer receipt format with a dark background and green accent typography. The receipt includes the cooperative branding (MilkRoute / MilkRoute Dairy Cooperative), collection timestamp, shift identifier, farmer ID and name, milk type, volume, quality parameters (FAT%, SNF%, protein%, temperature), grade, per-litre rate, and total payout amount. A UPI payment confirmation section displays the farmer's UPI ID and a simulated payment QR code area. The receipt is print-ready using the browser's native print dialog.

5.4 Chart.js Analytics Implementation

The dashboard analytics section uses Chart.js to render three primary visualizations. The weekly collection chart is a bar chart showing day-wise collection volume for the current week, computed from the collection log. The milk quality split chart is a doughnut chart showing the proportion of Grade A, B, and C collections in the current session. The historical analytics view renders a line chart of collection volume trends over the past 30 days, with a secondary axis for average FAT%. The top contributors chart is a horizontal bar chart ranking farmers by monthly collection volume.

Chart instances are stored in a module-level registry and destroyed before re-creation to prevent Chart.js canvas conflicts during tab switches. Chart color schemes use the application's CSS custom property palette, ensuring visual consistency with the broader UI design.

5.5 Responsive Design and Dark Mode

The application implements a two-panel layout with a fixed sidebar (230px width) and a scrollable main content area. The sidebar provides navigation to all module pages; the main area contains the active module's content within a vertically scrollable container. This layout is optimized for tablet-sized screens (10–12 inch) typical of field tablets, and degrades gracefully to larger laptop and desktop displays.

Dark mode is implemented via the CSS prefers-color-scheme media query, which overrides the default warm-parchment color palette with a dark green-tinted scheme. All color values are defined as CSS custom properties, enabling the theme switch with a minimal set of overrides. The dark mode

palette was designed to reduce eye strain during pre-dawn morning shift operations under artificial lighting.

6. Results and Discussion

Testing the prototype involved running through functional scenarios for all seven modules, walking through simulated collection shifts end-to-end, and checking performance on the kind of hardware an agent would actually carry. The results were mostly encouraging, with a few things that did not work as smoothly as expected initially and required adjustment.

6.1 Functional Completeness

All seven modules were tested against their defined requirements. End-to-end runs covering shift start, collection entry, quality grading, payment, and shift close worked correctly across all four routes and both shift types. A few of the more interesting edge cases tested: entering rejection-grade milk correctly triggered an alert and blocked payment processing; suspending a farmer mid-shift removed them from the active collection list without disturbing existing records; bulk payment with skip logic processed only eligible farmers and left flagged ones in the pending queue. Nothing unexpected came out of those tests, though getting the grade boundary conditions right took a couple of iterations.

The collection recording module correctly applies the quality grading engine to all test cases, including boundary conditions at grade thresholds. Payment transaction records are generated with correct amounts, timestamps, and UPI references. Alert generation fires correctly for quality failures, pending payments, and device status changes.

6.2 Data Integrity and Persistence

Data persistence held up across browser refresh and tab close/reopen for all entity types. The pre-loaded dataset of 137 farmer profiles initializes cleanly on first load and survives collection and payment operations without any corruption. One thing worth noting: early in development there were issues with numeric types being serialized as strings and then silently failing in rate calculations. Adding explicit type coercion at the deserialization step fixed that, and it has been stable since. Special characters in farmer names and UPI IDs also round-trip correctly, which mattered for the Andhra Pradesh naming conventions in the dataset.

The collection log accurately computes aggregate metrics — total volume, total payout, average FAT%, grade distribution — from individual records. These aggregates are consistent between the dashboard display, collection log table, and historical analytics views.

6.3 UI Performance

On a mid-range Android tablet (2021, Snapdragon 680), the app loaded in about 1.8 seconds from a local file, with Chart.js adding another 0.4 seconds to render the dashboard charts. The farmer list with 137 entries took 0.6 seconds to render, which was a bit slower than I wanted initially — an earlier version took over a second because the table was being rebuilt unnecessarily on every state update. After fixing that, 0.6 seconds felt acceptable for field use. Collection log rendering with 50 entries came in at 0.3 seconds, which was fine throughout. None of these numbers are fast by modern web standards, but they work for an agent who is not expecting a banking app experience in a village at 6 AM.

The application's single-file packaging eliminates network round-trips for resource loading, making load time entirely dependent on device processing speed. This is advantageous for rural deployment where network latency can be unpredictable.

6.4 Discussion of Limitations

The limitations of this prototype are real and should not be minimized. The most significant is data fragility — localStorage has no redundancy, so a device reset or an accidental browser data clear destroys everything. For a production system handling daily income records for 130+ farmers, that is not acceptable. A backend with offline-first caching and conflict resolution would be the obvious fix. The UPI flow is simulated; connecting to an actual gateway like Razorpay or PhonePe Business requires RBI compliance and a registered business entity, neither of which applied here. The grading rate table is hardcoded, which works for a prototype but would need to be administrator-configurable in practice since rates change with seasons and market conditions. IoT device data is mocked; real integration would need ruggedized hardware, cellular connectivity, and a cloud MQTT broker. And there is no authentication at all right now — anyone who picks up the device can read or edit everything, which is a serious gap that would need to be addressed before any real deployment.

6.5 Comparison with Related Systems

Compared to Desai et al. [2], MilkRoute covers more of the workflow in one place. Their system had better field validation — 200 villages, real agents, real outcomes — which MilkRoute does not yet have. But the architectural difference matters: requiring an agent to manually sync data across separate quality, payment, and records modules introduces exactly the kind of human error that digitization is supposed to reduce. MilkRoute's client-side approach does give up multi-user access and centralized monitoring, which are genuine tradeoffs worth acknowledging. A cooperative supervisor cannot see live data from all routes simultaneously in the current version. That is a real limitation, not just a future-scope item.

Most dairy digitization systems in the literature still assume bank transfer or cash for payment — cycles that take days and leave farmers with nothing to verify. The UPI flow in MilkRoute produces a per-transaction receipt immediately at the point of collection. In a production version with SMS notification enabled, the farmer would receive confirmation on their phone the moment payment goes through. That is a fundamentally different experience from waiting three days and then asking the agent what happened.

7. Future Scope

MilkRoute in its current form represents a functional prototype that demonstrates the feasibility of integrated digital dairy collection management. Several extensions are identified for future development:

7.1 Backend Infrastructure and Multi-User Support

Moving from localStorage to a cloud backend — Firebase Firestore or a simple RESTful API backed by PostgreSQL — would open up multi-user operation with proper role separation: agents, supervisors, cooperative administrators, and eventually farmers on their own devices. Real-time visibility for supervisors across multiple routes would become possible. Offline-first PWA with a service worker would let the app keep working during connectivity gaps and sync in the background when the connection returns, which is how it should work in rural conditions anyway.

7.2 Production UPI Integration

Production UPI payment integration via a registered payment gateway would enable actual monetary transactions at the point of collection. SMS notification to farmers upon payment confirmation — feasible via SMS gateways such as Twilio or MSG91 — would provide farmers with a payment receipt on their feature phones without requiring them to have a smartphone. WhatsApp-based collection receipts, via the WhatsApp Business API, would serve farmers with smartphone access.

7.3 Machine Learning for Quality Prediction and Anomaly Detection

Collection of quality parameter data over time creates the opportunity for machine learning applications. A predictive model trained on historical FAT%, SNF%, and protein data could identify farmers likely to produce below-grade milk in the current session, enabling proactive quality counseling. An anomaly detection model could flag suspicious patterns in quality readings or volume measurements that may indicate meter tampering or data manipulation. Seasonal analysis of quality trends could inform cooperative procurement planning.

7.4 Physical IoT Integration

Integration with physical flow meters, temperature sensors, and milk analyzers over MQTT would enable tamper-evident automated data entry, eliminating the manual keying of quality readings that is the primary vector for data manipulation in current systems. Standardized hardware interfaces (e.g., RS232 or Bluetooth LE for milk analyzers, MQTT for IoT sensors) would allow the system to integrate with the range of equipment already deployed by cooperatives.

7.5 Multilingual Support and Farmer-Facing Mobile App

A farmer-facing mobile application — providing each farmer with their own view of collection records, payment history, and quality feedback — would significantly enhance the transparency and trust benefits of the system. This application would need to support Telugu, Kannada, Tamil, and Hindi, reflecting the linguistic diversity of dairy farming communities in South India. Voice-based interfaces, given low literacy rates in some target communities, would further broaden accessibility.

8. Conclusion

Building MilkRoute was, in many ways, an exercise in understanding how much a small tool can actually do when it is designed around a real workflow rather than a generic one. The system covers the full arc of a collection shift — from route selection at the start of the morning to payment confirmation at the last village stop — and does it without needing a server, an IT team, or reliable internet. That combination was the goal from the beginning, and getting it to work within those constraints required more thought than I initially expected, particularly around state management, grading logic edge cases, and making the interface fast enough to not slow down an agent who is already working under time pressure.

The grading engine uses straightforward FAT/SNF thresholds rather than something statistically sophisticated, and I think that was the right call. A farmer at a village collection point needs to understand why their milk was graded B and not A. A fuzzy-logic output does not help with that. Similarly, the UPI payment flow was designed to produce a visible receipt at the moment of payment — something a farmer can see on screen and, in a production deployment, receive on their phone. That kind of immediate, verifiable record is something that paper-based systems simply cannot

provide. The route management module is more of a planning aid than a dynamic optimizer, but in practice a structured sequence with ETA tracking already does a lot more for shift efficiency than no structure at all.

All seven modules work end-to-end on mid-range hardware, which was one practical benchmark I kept in mind throughout development. The limitations are real and worth being direct about: there is no backend, the UPI flow is simulated, IoT data is mocked, and there is no user authentication. A production deployment would need all of those things. But the architecture was built with those extensions in mind — the data model, the module boundaries, the API contracts for UPI and IoT — so adding them should not require rewriting the core system from scratch. That was a deliberate decision, not an afterthought.

The broader point this project reinforced for me is that agricultural digitization tools fail in the field not because the technology is wrong but because the deployment assumptions are wrong. Tools that require a stable server connection, or a trained IT person nearby, or a smartphone the farmer already owns — these assumptions break down in rural settings constantly. MilkRoute was designed around what is actually available: a browser, occasional connectivity, and one agent who knows the routes. Working within those constraints was not limiting. It was the whole point.

References

1. National Dairy Development Board (NDDB). Computerised Milk Collection Centres. NDDB Technical Publications, Anand, India, 2003.
2. Desai, P., Shah, R., and Patel, V. Tablet-Based Milk Collection Terminal for Rural Dairy Cooperatives: A Field Evaluation. Journal of Agricultural Informatics, 14(2):45–58, 2023.
3. Dairy India Yearbook. Milk Analyzer Technology in Cooperative Collection: Adoption and Impact. Dairy India Publications, New Delhi, 2022.
4. Kumar, A. and Srinivasan, R. Fuzzy Logic-Based Milk Quality Grading for Low-Cost Infrared Analyzers. Computers and Electronics in Agriculture, 198:107054, 2022.
5. National Payments Corporation of India (NPCI). UPI Product Statistics. NPCI Monthly Report, 2024.
6. Narayanan, S., Bhattacharya, A., and Verma, K. Instant Digital Payments in Agricultural Procurement: Evidence from a UPI Pilot in Maharashtra. Indian Journal of Agricultural Economics, 78(3):312–328, 2023.
7. Sharma, N., Gupta, R., and Singh, P. Metaheuristic Route Optimization for Milk Collection Logistics: A Case Study from Haryana. International Journal of Logistics Management, 34(1):78–95, 2023.
8. Patil, S., Kamat, R., and Joshi, D. IoT-Enabled Tamper-Evident Flow Meters for Dairy Collection: Design and Field Validation. Sensors, 23(8):4112, 2023.
9. FSSAI. Food Safety and Standards (Food Products Standards and Food Additives) Regulations, 2011 – Standards for Milk and Milk Products. Food Safety and Standards Authority of India, New Delhi, 2011.
10. Bureau of Indian Standards. IS:1479 – M ethods of Test for Dairy Industry. BIS, New Delhi, 2019.