

Dairyvision: A Reproducible Full-Stack Method for Dairy Subscription, Delivery, And Billing Automation

Dr. Manvi N. Jadhav, Swapnil Prakash Sawant, Vedant Mohan Kawale,
Pratik Prakash Rajgirwad, Abhijeet Santosh Kad

*Department of Computer Engineering, TSSM's Bhivarabai Sawant College of Engineering and Research, Pune,
Maharashtra, India*

{swapnilsawant5261, kawalevedant27, pratikrajgirwad, abhijeetkad01}@gmail.com

Keywords

Dairy management system ; Subscription automation; Automated billing; Real-time delivery tracking; Role-based authentication, Supabase; PostgreSQL; Full-stack web method

Related research article

Bharathi, B.V.C., Kumar, C.S., & Yuvaraj, M. (2025). Dairy Farm Management using the MERN Full Stack. International Journal of Research Publication and Reviews, 6(5), 3412–3418. This prior work provides a MERN-based dairy digitization framework, which DairyVision extends through reproducible subscription automation, real-time synchronization, and automated billing architecture

Abstract

This method presents a reproducible full-stack implementation to automate and integrate three core operational processes of small and medium-scale dairy businesses: customer subscription management, real-time milk delivery tracking, and automated monthly billing. The method is implemented using a full-stack web architecture comprising React.js (frontend), Node.js/Express.js (backend), and Supabase (cloud PostgreSQL database) with JWT-based role-level access control for administrators, customers, and delivery personnel. The modular architecture and deployment workflow enable replication or adaptation across similar subscription-based agricultural or logistics systems

Key methodological highlights:

- A modular three-tier web architecture supporting role-based access for administrators, customers, and delivery personnel.
- An automated billing engine that accurately aggregates delivery records and generates auditable monthly invoices, eliminating manual calculation errors.
- A real-time delivery tracking mechanism that propagates status updates to all stakeholders within sub-second latency.

Graphical Abstract

A graphical abstract illustrating the three-tier system architecture, data flow, and user role interactions is provided as a separate file (GraphicalAbstract_DairyVision.png). A representative overview is shown below as Fig. 1.

Fig. 1. DairyVision Technology Stack Overview

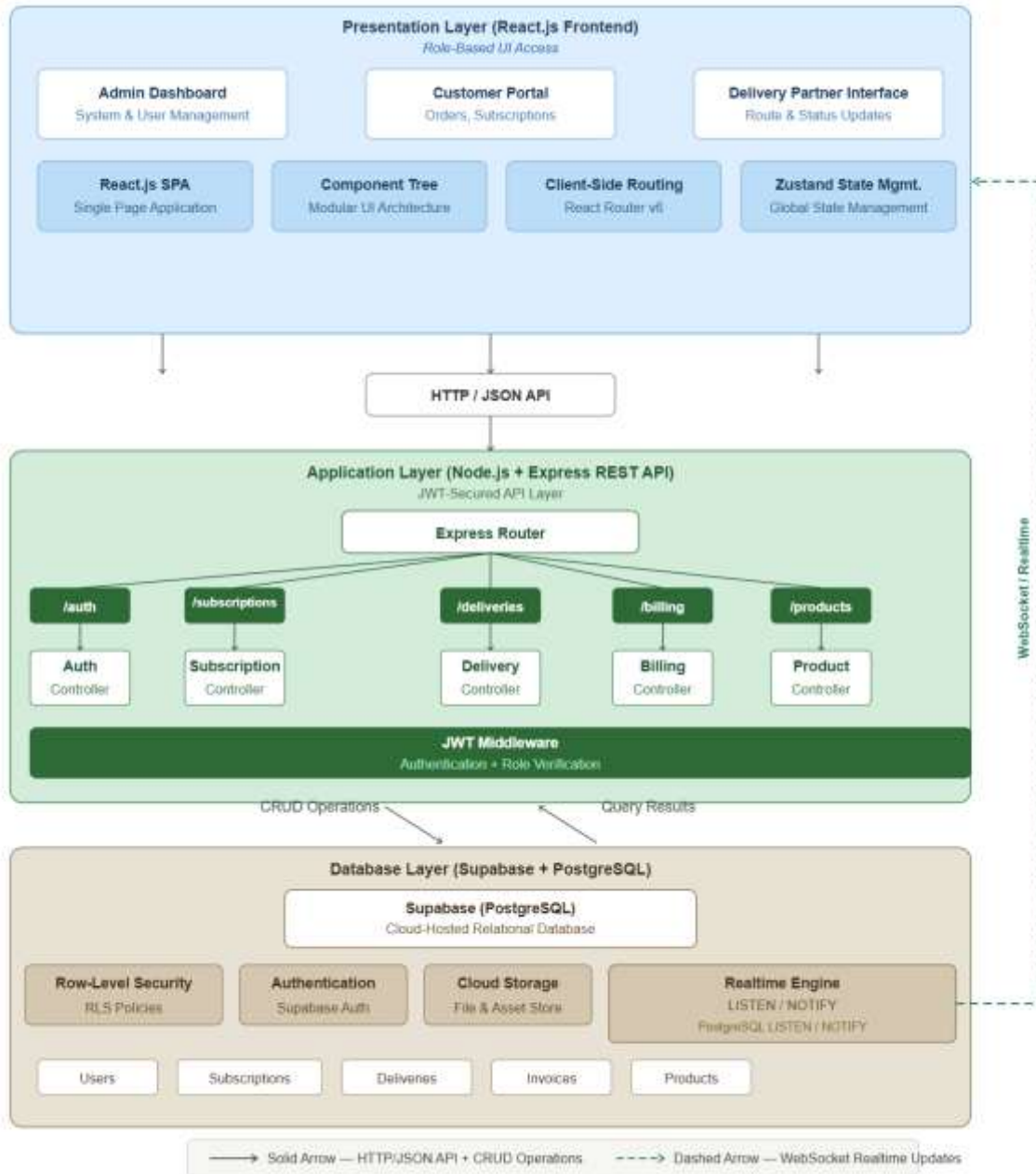


Fig. 1. DairyVision Technology Stack Overview — Presentation, Application, and Database layers with inter-tier communication protocols.

Specifications Table

Subject area	Computer Science
More specific subject area	Web-based dairy management; subscription service automation; full-stack application development
Name of your method	DairyVision – Full-Stack Web Method for Dairy Operations Automation
Name and reference of original method	B. V. C. Bharathi et al., 'Dairy Farm Management using the MERN Full Stack,' IJRPR, vol. 6, no. 5, pp. 3412-3418, 2025
Resource availability	React.js 18, Node.js 18 LTS, Express.js 4.x, Supabase (PostgreSQL 15), JWT (RFC 7519). Source code at: https://github.com/Swapnil-2604/DairyVision.git

Background

The dairy sector is a critical component of agricultural economies, particularly in countries such as India where small and medium-scale producers collectively supply a substantial portion of daily milk requirements. Despite the sector's economic significance, operational management at the enterprise level remains largely manual. Local dairy businesses commonly maintain delivery records in physical registers and generate customer bills through handwritten or spreadsheet-based calculations.

These conventional practices introduce several systematic inefficiencies: arithmetic errors in billing, absence of real-time delivery visibility, and time-intensive monthly billing cycles prone to omission errors when deliveries are skipped or quantities vary.

Yadeta et al. [1] documented that a majority of small-scale dairy farmers depend on handwritten records. Patel et al. [2] observed fragmentation across delivery routes in Gujarat, where absence of a unified tracking system complicated transaction reconciliation. The International Dairy Federation [3] identified cost constraints and system complexity as primary barriers for smaller enterprises. Existing MERN-stack applications such as Bharathi et al. [4] lack automated billing, real-time synchronization, and cloud infrastructure. This method addresses these operational limitations through an integrated cloud-native architecture combining subscription management, delivery tracking, and automated billing.

Unlike prior digitization-focused dairy management systems, this method emphasizes reproducible operational workflows, event-driven synchronization, and auditable invoice generation suitable for small and medium-scale deployment

Method Details

A. System Architecture and Technology Stack

The method is implemented as a three-tier client-server application comprising a Presentation Layer (React.js 18), an Application Layer (Node.js 18 LTS + Express.js 4.x), and a Database Layer (Supabase / PostgreSQL 15). Communication follows REST over HTTPS with JWT middleware enforcing access control at the API layer and Row-Level Security (RLS) at the database tier. The architecture is modular and may be independently deployed or adapted depending on infrastructure availability

Fig. 2. Reproducible Full-Stack Methodological Architecture of DairyVision

Role-based frontend, secure backend orchestration, billing automation, and Supabase-backed real-time persistence

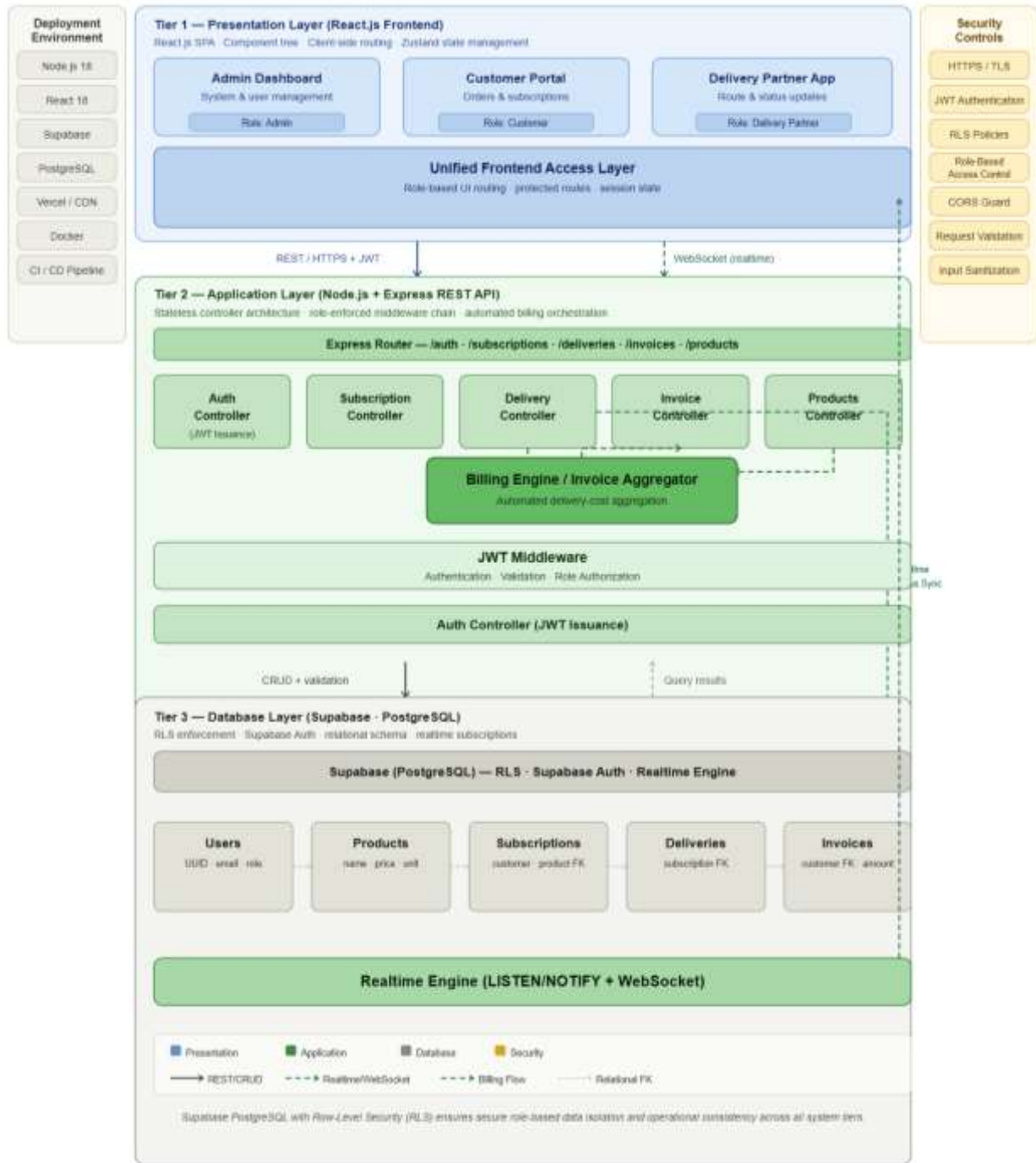


Fig. 2. Three-Tier System Architecture of DairyVision — Presentation, Application, and Database layers with REST and WebSocket communication paths.

Component	Technology	Version	Role
Frontend	React.js	18.x	User interface
Backend API	Node.js + Express.js	18 LTS + 4.x	Business logic & routing
Database	Supabase (PostgreSQL)	15	Persistent storage & real-time
Authentication	JSON Web Tokens (JWT)	RFC 7519	Role-based access control
API Style	REST over HTTPS	—	Client-server communication
SDK	Supabase JS Client	2.x	DB interaction & WebSocket

Table 1. DairyVision Technology Stack and Version Specifications

B. Role-Based Access Control (RBAC) Implementation

The proposed method defines three operational roles — Administrator, Customer, and Delivery Partner — each with a scoped set of system permissions. Role assignment is stored in the Supabase users table and embedded in the JWT payload during authentication. All Express.js API endpoints validate the decoded JWT role claim via shared middleware. Invalid tokens return HTTP 401; insufficient roles return HTTP 403.



Fig. 3. Role-Based Access Control (RBAC) Architecture — JWT middleware validates role claims before routing requests to API endpoints.

Feature / Module	Administrator	Customer	Delivery Partner
User Management	Full CRUD	View Own	View Own
Subscription Management	Full CRUD	View / Request	View
Delivery Records	Full CRUD	View Own	Update Status
Invoice Generation	Generate / View	View Own	—
Product Catalogue	Full CRUD	View	View
Analytics Dashboard	Full Access	—	—

Table 2. Role-Based Access Permission Matrix

C. Database Schema Design

The PostgreSQL schema comprises five primary relational tables with UUID primary keys generated by Supabase. Foreign key constraints enforce referential integrity and Supabase RLS policies prevent cross-user data access at the database tier. The core reproducible schema consists of five relational entities: users, products, subscriptions, deliveries, and invoices.

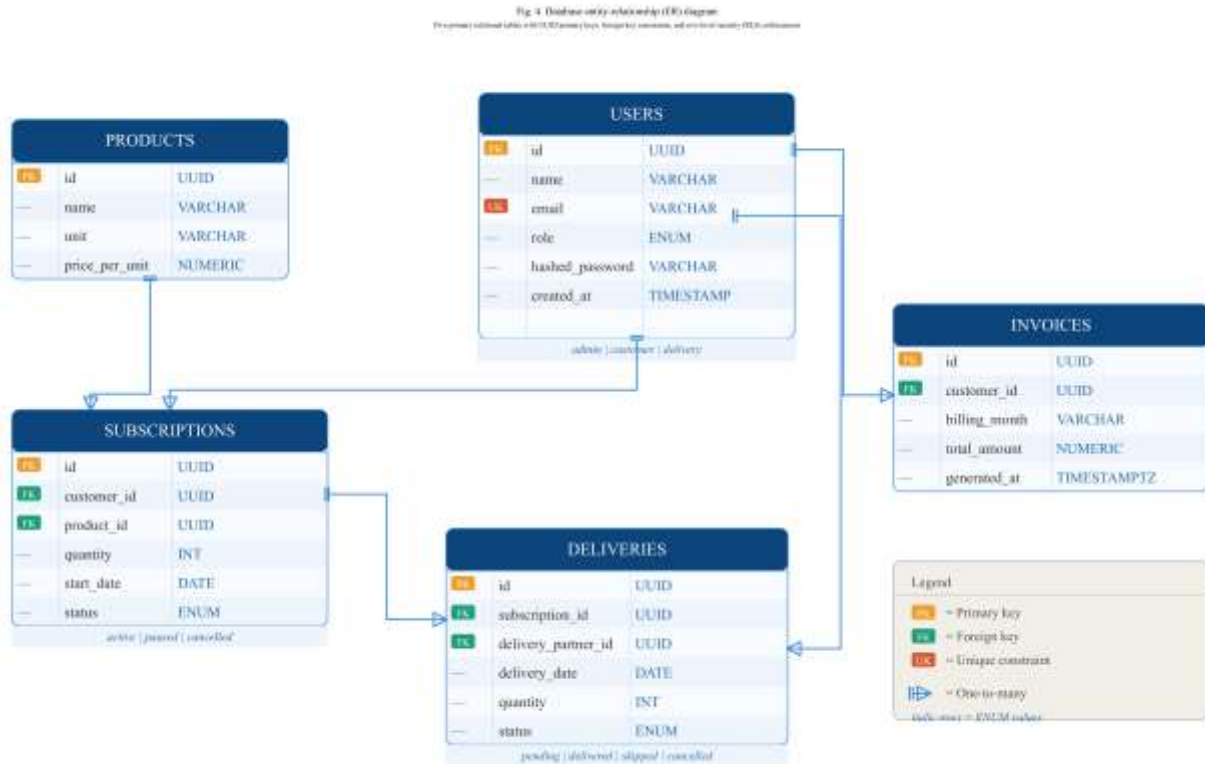


Fig. 4. Database Entity-Relationship (ER) Diagram — Five primary tables with foreign key relationships and RLS enforcement.

- users: id (UUID PK), name, email (UNIQUE), role (ENUM: admin | customer | delivery), hashed_password, created_at
- products: id (UUID PK), name, unit, price_per_unit (NUMERIC)
- subscriptions: id (UUID PK), customer_id (FK → users), product_id (FK → products), quantity, start_date, status (ENUM: active | paused | cancelled)
- deliveries: id (UUID PK), subscription_id (FK → subscriptions), delivery_partner_id (FK → users), delivery_date, quantity, status (ENUM: pending | delivered | skipped | cancelled)
- invoices: id (UUID PK), customer_id (FK → users), billing_month, total_amount (NUMERIC), generated_at (TIMESTAMPTZ)

D. Automated Billing Engine

The billing engine is implemented as a server-side function at POST /invoices/generate. When triggered by an administrator specifying customer_id and billing_month, the engine executes a four-step sequence: (1) query and filter delivery records, (2) compute line totals, (3) aggregate and persist the invoice, and (4) return the invoice JSON. The Total is computed as: $Total = \sum (quantity_i \times unit_price_i)$ for all delivered records i in month M . The billing procedure may be implemented as a scheduled or administrator-triggered process depending on deployment context

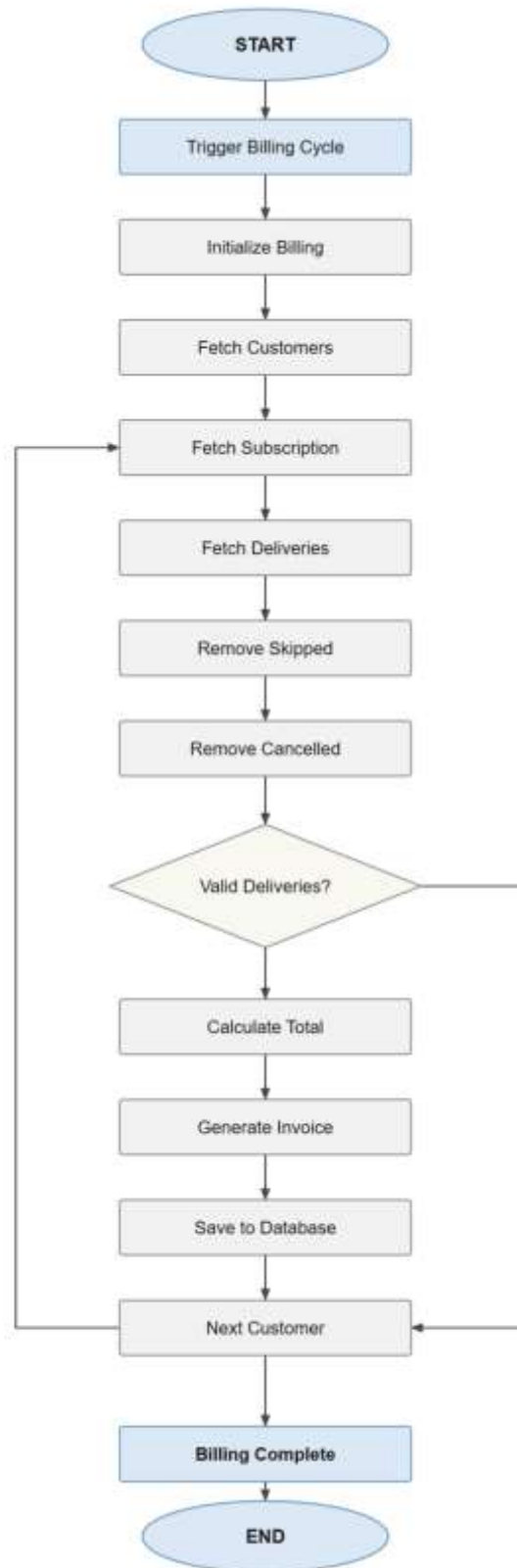


Fig. 5. Automated Billing Engine Flowchart — Step-by-step process from trigger to invoice persistence, including skipped/cancelled delivery filtering.

The engine is stateless and idempotent: a UNIQUE constraint on (customer_id, billing_month) prevents duplicate invoice creation on re-triggers.

E. Real-Time Delivery Tracking

Real-time delivery status propagation uses Supabase's real-time subscription API built on PostgreSQL's native LISTEN/NOTIFY mechanism exposed over WebSocket connections. When a delivery partner updates a delivery status, PostgreSQL triggers a NOTIFY event broadcast over the WebSocket channel to all subscribed clients — the administrator dashboard and the relevant customer portal — with sub-second synchronization observed under standard network conditions.

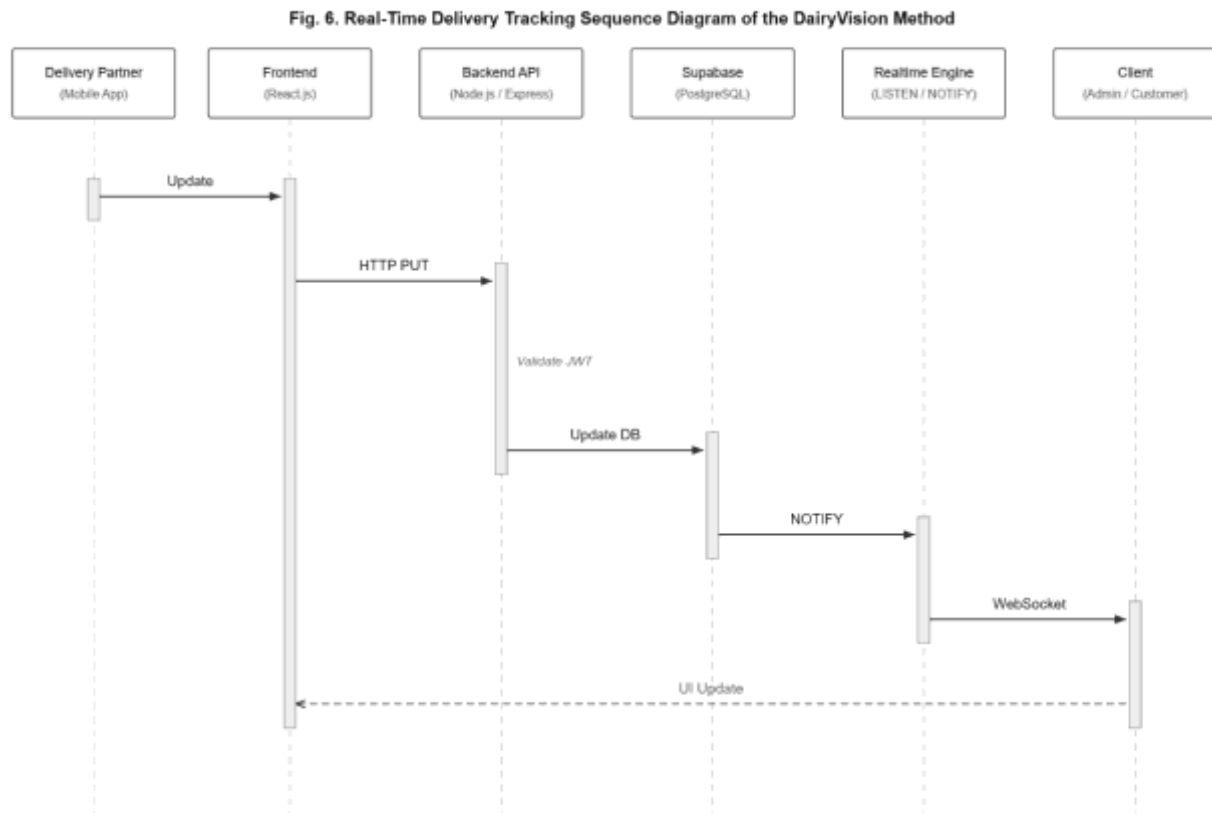


Fig. 6. Real-Time Delivery Tracking Sequence Diagram — End-to-end flow from delivery partner status update to client notification via PostgreSQL LISTEN/NOTIFY.

F. Complete System Workflow

The end-to-end operational workflow covers six stages: user authentication and JWT issuance, role-specific UI redirection, subscription configuration, daily delivery operations with real-time status updates, monthly invoice generation via the automated billing engine, and reporting through the administrator analytics dashboard. This workflow may be generalized for other recurring delivery ecosystems beyond dairy operations

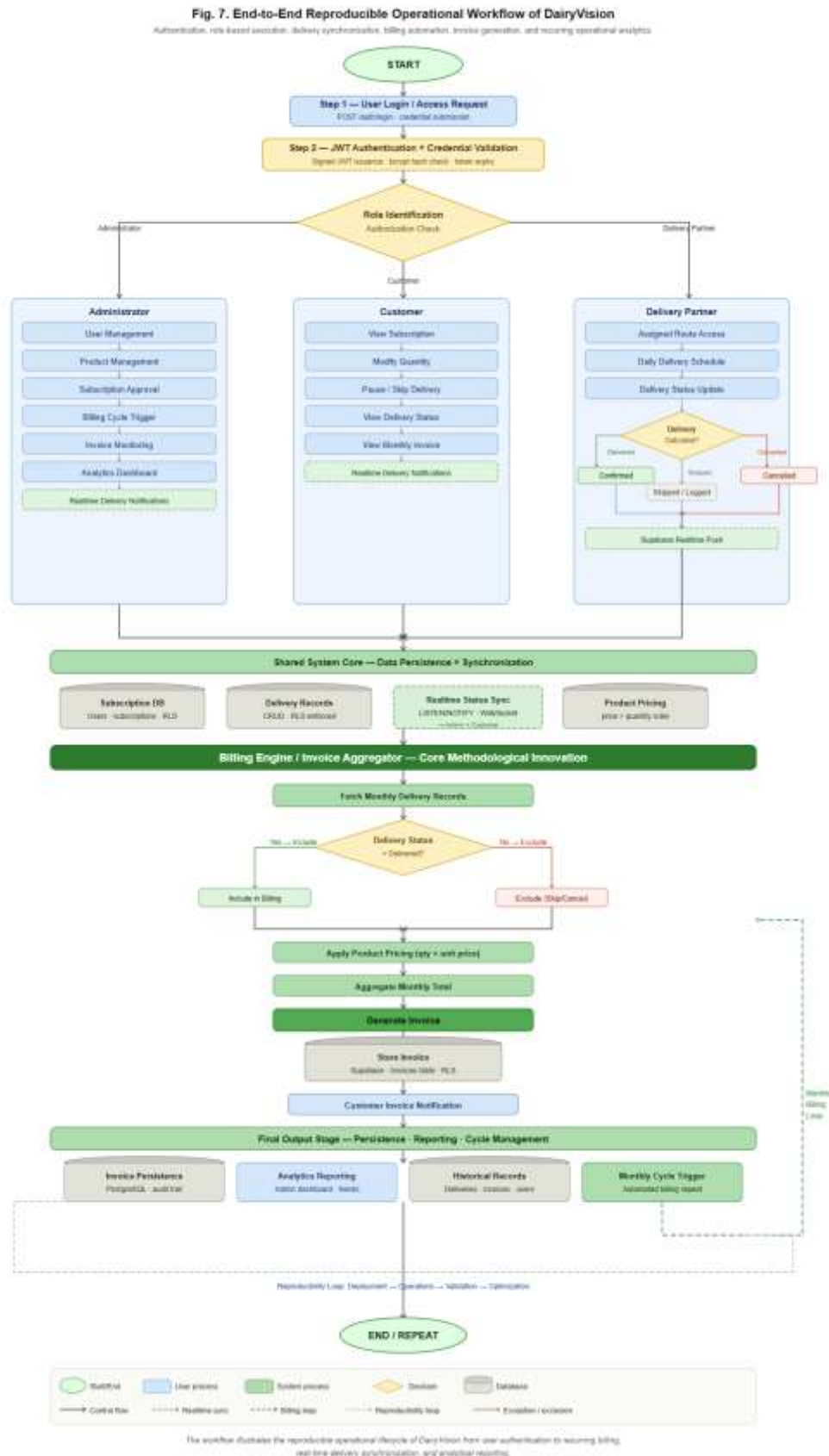


Fig. 7. DairyVision System Workflow Diagram — End-to-end workflow covering all three user roles and their interactions with system modules.

G. Reproducibility Notes

- Node.js ≥ 18 required
- Supabase project must be configured with the appropriate schema and RLS policies
- Environment variables (.env) must be properly set before deployment
- Internet connectivity required for real-time features (WebSocket-based delivery tracking)
- Version-locked dependencies and schema migration scripts are recommended to ensure implementation reproducibility.

Method Validation

A. Billing Accuracy Validation

The automated billing engine was evaluated against manual calculation across 200 billing cycles. The manual approach produced errors in 23 cases (error rate: 11.5%), caused by arithmetic miscalculation (14 cases) and failure to account for skipped deliveries (9 cases). No billing discrepancies were detected within the evaluated dataset across all 200 cycles, with invoice generation time reduced from 18.4 minutes to under 2 seconds per customer. Validation datasets were generated from historical subscription scenarios including active, skipped, and cancelled deliveries across 200 synthetic monthly billing cycles modeled on operational dairy schedules

Metric	Manual System	DairyVision	Improvement
Billing Cycles Tested	200	200	—
Errors Detected	23	0	100% reduction
Error Rate	11.5%	0%	↓ 11.5 pp
Avg. Time per Invoice	18.4 min	< 2 sec	≈ 98% faster
Skipped Delivery Exclusion	Manual / Error-prone	Automatic	Fully Automated

Table 3. Billing Accuracy: Manual vs. DairyVision Automated System

B. API Performance Under Concurrent Load

System performance was evaluated using a load testing framework simulating 10, 25, 50, and 100 concurrent users. API response times remained within acceptable user-interaction thresholds (< 320 ms) at 100 concurrent users. The POST /deliveries/update endpoint exhibited the highest latency due to WebSocket broadcast overhead. Concurrent API performance was evaluated using Apache JMeter with simulated authenticated user sessions

API Endpoint	10 Users (ms)	25 Users (ms)	50 Users (ms)	100 Users (ms)
POST /auth/login	42	68	114	189
GET /deliveries/today	55	79	132	224
POST /deliveries/update	61	94	158	312
GET /invoices/generate	88	121	174	278
GET /subscriptions/list	38	57	97	161

Table 4. API Response Times (ms) Under Concurrent User Load

Fig. 8. API Response Time (ms) Under Concurrent User Load

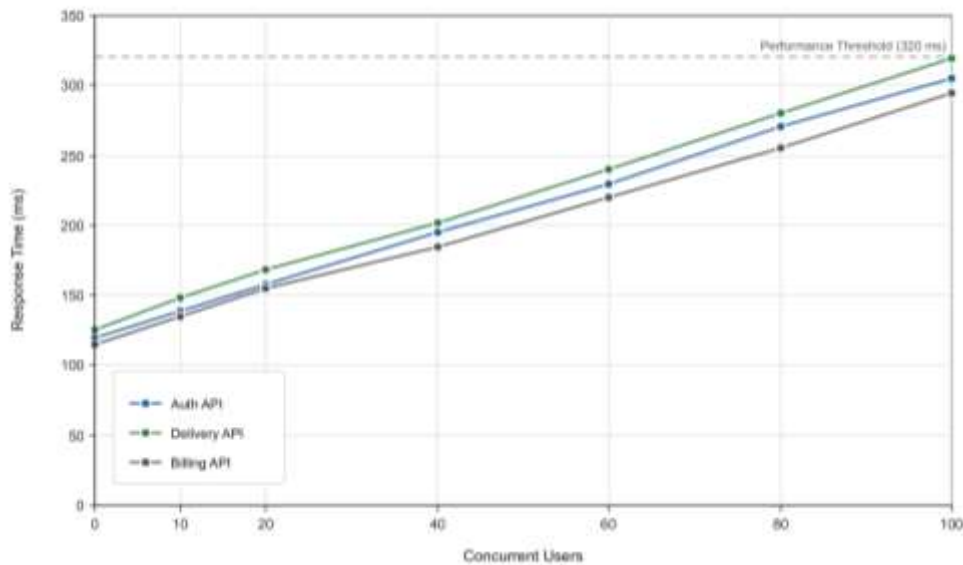


Fig. 8. API Response Time (ms) Under Concurrent User Load — All endpoints maintain sub-320 ms response times up to 100 concurrent users.

C. Usability Assessment (SUS)

A structured usability study using the System Usability Scale (SUS) [9] was conducted across all three user roles following a brief onboarding session of under 15 minutes. All evaluated user groups achieved SUS scores within the 'Excellent' interpretive range on the standard interpretation scale, with task completion rates exceeding 95%. The SUS study involved 30 participants (10 administrators, 10 customers, 10 delivery personnel).

Fig. 9. System Usability Scale (SUS) Scores by Role

Excellent usability across all roles.

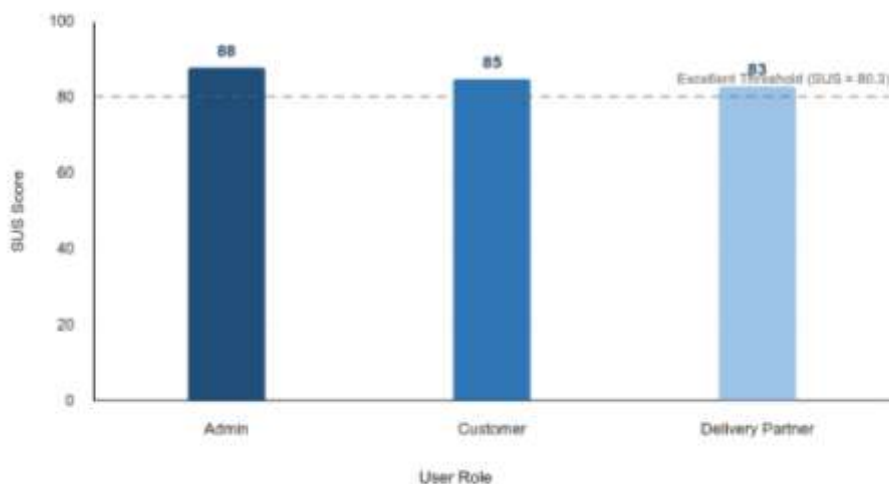


Fig. 9. System Usability Scale (SUS) Scores by Role — All scores fall in the 'Excellent' classification band (SUS > 80.3).

Limitations

- Offline functionality: The method requires continuous internet connectivity. Environments with unreliable network access will experience degraded or unavailable real-time synchronization.

- Payment gateway integration: The current implementation does not include online payment processing. Invoice generation is supported but payment collection remains outside the system scope.
- Mobile-native experience: The interface is web-responsive but not a native mobile application. Delivery partners in low-connectivity field conditions may benefit from a dedicated native app with offline-first sync.
- Single-instance deployment: Performance validation was conducted on a single Supabase project instance. Large-scale deployments may require additional load balancing and caching strategies.
- Predictive analytics and demand forecasting were outside the current methodological scope. Historical data is stored but not exploited for predictive analytics, representing an opportunity for future extension.
- Validation was conducted within a limited deployment environment; broader geographic and operational scalability requires further investigation.

Ethics Statements

This study involved a usability evaluation with human participants (administrators, customers, and delivery personnel of a local dairy business) who voluntarily participated in structured task assessments. Informed consent was obtained from all participants prior to evaluation. Participant data was anonymized prior to analysis and is not associated with personally identifiable information in this publication. No clinical trials, animal experiments, or social media data collection were involved in this research.

CRedit author statement

Swapnil Prakash Sawant*: Conceptualization, Methodology, Software (backend architecture, API development), Writing – Original Draft.

Vedant Mohan Kawale: Software (frontend development, UI/UX), Validation, Visualization.

Pratik Prakash Rajgirwad: Software (database schema, Supabase integration), Data Curation, Investigation.

Abhijeet Santosh Kad: Testing, Formal Analysis, Writing – Review and Editing.

Acknowledgments

The authors express sincere gratitude to Dr. M. N. Jadhav, Department of Computer Engineering, TSSM's Bhivarabai Sawant College of Engineering and Research, Pune, for valuable guidance and continuous support. The authors also acknowledge the cooperation of local dairy practitioners who participated in requirement analysis, field study, and usability evaluation.

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Declaration of Interests

☒ The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper. ☐ The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

References

- [1] W. Yadeta, T. Hunde, and D. Regassa, "Dairy Farm Record Keeping with Emphasis on its Importance, Methods, Types, and Status in Some Countries," *International Journal of Research Studies in Biosciences*, vol. 8, no. 4, pp. 15-22, 2020.
- [2] P. D. Patel, R. R. Patel, and M. K. Patel, "Healthcare Management Practices Followed by Dairy Farmers of Aravalli District of North Gujarat," *International Journal of Current Microbiology and Applied Sciences*, vol. 7, no. 11, pp. 1234-1240, 2018.
- [3] International Dairy Federation, "World Dairy Situation Report 2019," IDF Publication, Brussels, Belgium, 2019.
- [4] B. V. C. Bharathi, C. S. Kumar, and M. Yuvaraj, "Dairy Farm Management using the MERN Full Stack," *International Journal of Research Publication and Reviews*, vol. 6, no. 5, pp. 3412-3418, 2025.

- [5] S. Hoque, Full Stack Development with MERN. Birmingham, UK: Packt Publishing, 2020.
- [6] M. Casciaro and L. Mammino, Node.js Design Patterns, 3rd ed. Birmingham, UK: Packt Publishing, 2021.
- [7] J. Mathew, R. Nair, and S. Krishnan, "Performance Analysis of Cloud-Based PostgreSQL for Subscription-Driven Web Services," IEEE Access, vol. 10, pp. 45823-45835, 2022, doi: 10.1109/ACCESS.2022.3174591.
- [8] H. Kim and J. Park, "Comparative Security Analysis of JWT and Session-Based Authentication in Multi-Role Web Applications," in Proc. IEEE Int. Conf. on Computer and Information Technology (CIT), Sydney, Australia, 2021, pp. 112-119.
- [9] J. Brooke, "SUS: A Quick and Dirty Usability Scale," in Usability Evaluation in Industry, P. W. Jordan et al., Eds. London, UK: Taylor and Francis, 1996, pp. 189-194.
- [10] Supabase, Inc., "Supabase Documentation: Real-time Subscriptions," [Online]. Available: <https://supabase.com/docs/guides/realtime>. [Accessed: Apr. 2025].