

Web-Based Smart Dairy Management Platform Using Python and the Django Framework

Dr. K. Ravi Kumar¹, K.V. SGanesh², Mahendra Kumar Guru³, Kalluru Chiranjeevi⁴, Gembali Akhil⁵

¹Professor: Department of Electronics and Communication Engineering Avanthi Institute of Engineering & Technology, Tamaram, Makavarapalem, Anakapalle – 531113, Andhra Pradesh, India

²Assistant Professor: Department of Electronics and Communication Engineering Avanthi Institute of Engineering & Technology, Tamaram, Makavarapalem, Anakapalle – 531113, Andhra Pradesh, India

³⁻⁵UG Students: Department of Electronics and Communication Engineering Avanthi Institute of Engineering & Technology, Tamaram, Makavarapalem, Anakapalle – 531113, Andhra Pradesh, India

Abstract

Dairy businesses of small and medium scale still run largely on handwritten registers. Customer details, daily milk quantities and payment records are entered by hand, totals are calculated by hand, and the register is the only copy. The result is arithmetic error, lost records and slow month-end settlement. This paper describes a web-based dairy management platform built to replace that register. The system is developed in Python using the Django framework with a Bootstrap front end and a relational database, and follows Django's Model-View-Template architecture so that data storage, business logic and presentation stay separate. It supports customer registration with rate and shift details, daily milk-quantity entry with the bill computed automatically from quantity and rate, payment status tracking marked as paid, pending or unpaid, filtering of records by customer and by shift, and date-wise payment summaries. A dashboard reports total customers, total revenue, amount received and amount pending, together with the most recent transactions. The platform was tested against ten functional test cases covering record creation, input validation, automatic billing, status update, filtering and deletion, all of which passed. Because the system is web-based it runs on ordinary hardware and is reachable from any device on the network, which makes it practical for the small and medium dairies that are the intended users.

Index Terms— Dairy management, web application, Python, Django framework, Model-View-Template architecture, Bootstrap, automated billing, payment tracking, database management, digital agriculture.

I. Introduction

The dairy sector occupies a substantial place in the agricultural economy, supplying milk and milk products to households and to commercial buyers. The operational side of a small dairy is simple in principle: a set of customers, a quantity of milk supplied to each on each day, a rate per litre, and a balance that is settled periodically. In practice this is still handled with a register. Customer registration, daily quantity recording, payment tracking and revenue calculation are all entered and totalled by hand.

Manual handling produces three recurring problems. Arithmetic errors enter the accounts and are not discovered until a customer disputes a bill. Records are lost when a register is damaged or misplaced, and there is no second copy. And producing a monthly summary means re-reading and re-totalling every page, which is slow enough that many dairies simply do not produce one.

Spreadsheet software is a partial answer and some dairies use it. It automates the arithmetic, but it does not give centralised access, it offers no structured relationship between a customer record and that customer's daily entries, and a spreadsheet file is as easy to lose or overwrite as a register.

The platform described here replaces the register with a web application. A dairy administrator registers customers once, records a quantity against a customer each day, and the system computes the amount, records the payment status and keeps the history. A dashboard shows the current position of the business at a glance. Because the application is web-based it can be reached from any device on the network, and because it is built on an established web framework the data is held in a proper database with the usual protections rather than in a file.

The remainder of this paper reviews related work on digital technology in dairy and livestock farming, analyses the limitations of the manual system and the requirements of a replacement, describes the design and implementation of the platform, presents the working screens and the test results, and concludes with the directions in which the system can be extended.

II. Literature Survey

Research on digital technology in dairy and livestock farming has concentrated on sensing and analytics rather than on the record-keeping and billing layer that this work addresses, but the findings are relevant

because they establish both the value of structured farm data and the practical barriers to adoption on small farms.

Wolfert and colleagues reviewed the role of big data in smart farming and described how connected sensors, cloud computing and analytics can gather large volumes of agricultural data and feed it back into farm decisions. They argue that the effect extends beyond the farm to the whole supply chain through improved efficiency and transparency. The study is largely conceptual, however; it offers little on practical implementation for small farms, and questions of data privacy, governance and supporting infrastructure are left open.

Kamilaris and Prenafeta-Boldú surveyed deep learning in agriculture and found it applied to crop monitoring, livestock behaviour analysis and disease detection, with image and sensor data processed at high accuracy. The limitations they report are the ones that keep such methods out of small dairies: large training datasets are required, computational resources are substantial, and most reported solutions remain experimental.

Berckmans examined precision livestock farming technologies for welfare management, describing sensors and automated monitoring that track animal behaviour, feeding and health and allow early detection of disease. Here too the barriers are installation and maintenance cost, the technical expertise needed to interpret the data, and consequently limited adoption on small and medium farms.

Neethirajan reviewed the role of sensors and the Internet of Things specifically in dairy farming, covering wearable sensors, automated milking and environmental monitoring, and the real-time data they provide for production and health decisions. The reported obstacles are the initial cost of the devices, dependence on reliable connectivity, and unresolved problems of data integration and security.

Two things follow from this body of work. The benefit of holding dairy data in structured, queryable form is well established. But the sensing-led approaches that dominate the literature carry a cost and complexity that small dairies cannot absorb. A system that digitises the records and the billing without requiring any instrumentation addresses the same underlying need at a cost these businesses can meet, and it also puts in place the database that a later sensing layer would write into.

III. System Analysis

A. Existing System

The existing arrangement in most small and medium dairies is manual or semi-digital. Records are kept in

handwritten registers, or in some cases in spreadsheets that automate the arithmetic but provide no real-time update, no centralised access and no structured database. The specific limitations observed are: arithmetic errors from manual entry; risk of total data loss if the register is damaged or misplaced; no automatic billing; no single view of revenue and payments; difficulty in identifying which payments are outstanding; no access control; slow monthly summary generation; no way to filter by shift; no link between the customer record and that customer's daily entries; and no practical way to keep long-term history.

B. Proposed System

The proposed platform addresses each of these. Billing is computed automatically from quantity and rate. Data is held in a centralised database rather than a file or a book. The dashboard reports total revenue, amount received and amount pending in real time. Records can be filtered by shift or by individual customer, and date-wise payment summaries are generated on demand. Payment status is tracked explicitly as paid, pending or unpaid, so outstanding amounts are visible rather than inferred. The interface is built with Bootstrap and is usable on a phone as well as a desktop browser.

C. Feasibility

Technically the system is feasible because every component is open source and well documented. Python and Django are widely supported, the database can be SQLite for a single dairy or MySQL where more capacity is needed, Bootstrap handles the responsive layout, and the application runs on ordinary hardware with no specialised equipment, deployed either on a local server or on a hosted platform.

Operationally it is feasible because the interface was designed for an administrator with no technical background. The dashboard presents summaries directly, the forms are short, and the training required amounts to showing someone the three screens they will use daily.

Economically it is feasible because there is no licence cost, no instrumentation to buy, and no hardware beyond a computer the dairy is likely to own already.

D. Requirements

Functionally the system must register customers with their contact details, milk quality, rate per litre and shift; record a milk quantity against a customer for a given date; compute the amount automatically from quantity and rate; record and update payment status; display a dashboard of totals; filter records by customer and by shift; generate date-wise summaries; and allow customer records to be removed.

B. Use Cases

Non-functionally, pages must load quickly and filtering must return results without a noticeable wait; the interface must be clean, responsive on mobile devices and clear about status; the system must protect data through Django's cross-site request forgery protection, form validation and restricted administrative access; and the architecture must allow the database to be upgraded and the deployment to move to a cloud host as the business grows.

The hardware and software required are summarised in Table I.

Table I. System Requirements

Item	Requirement
Processor	Intel i3 or above
RAM	4 GB minimum
Storage	20 GB free
Operating system	Windows, Linux or macOS
Language	Python 3
Framework	Django 4.x / 5.x
Front end	HTML, CSS, Bootstrap, JavaScript
Database	SQLite or MySQL
Client	Any modern web browser

IV. System Design

The platform follows Django's Model-View-Template architecture. The model layer defines the database structure, the view layer holds the business logic including the billing calculation, and the template layer renders the interface. Separating the three keeps each part independently modifiable, which is what makes later extension practical.

A. Data Flow

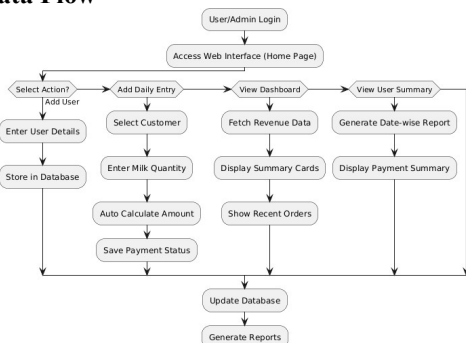


Fig. 1. Data flow diagram of the platform.

Fig. 1 shows how data moves through the system. Customer details and daily quantities enter through the web forms, pass through validation into the database, and are read back by the dashboard and summary views.

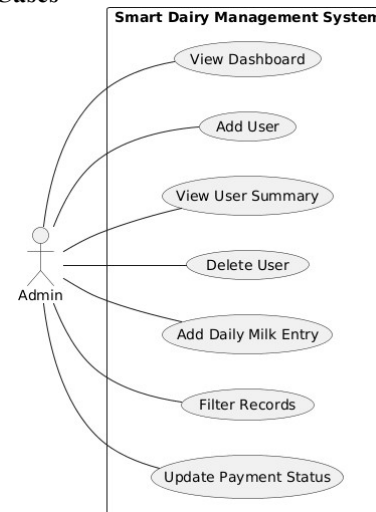


Fig. 2. Use case diagram showing the administrator's interactions.

The administrator is the single actor in the current version. The use cases are registering a customer, adding a daily entry, updating payment status, viewing the dashboard, filtering records and deleting a customer.

C. Data Model

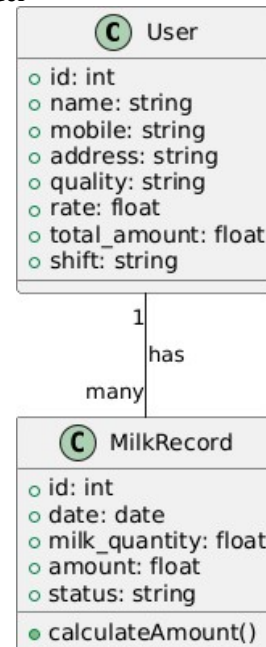


Fig. 3. Class diagram of the data model.

The data model has two entities in a one-to-many relationship, shown in Fig. 3. A User record holds the customer's name, mobile number, address, milk quality, rate, total amount and shift. Each MilkRecord holds a date, a milk quantity, an amount and a payment status, and belongs to one User. The amount is produced by the calculateAmount operation from the

Mahendra Kumar Guru *et. al.*, /International Journal of Engineering & Science Research

quantity on the record and the rate on the parent customer, so the rate is stored once and a change to it does not require every past entry to be revisited.

D. Interaction and Workflow

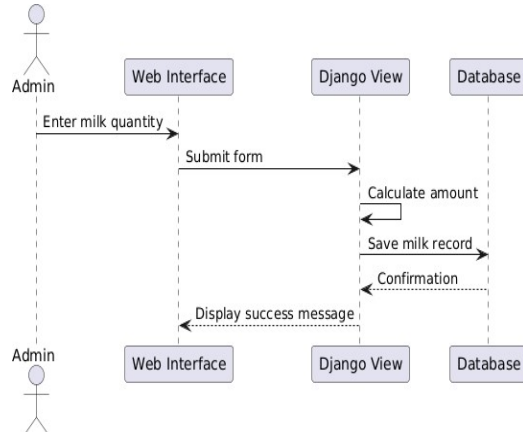


Fig. 4. Sequence diagram for adding a daily milk entry.

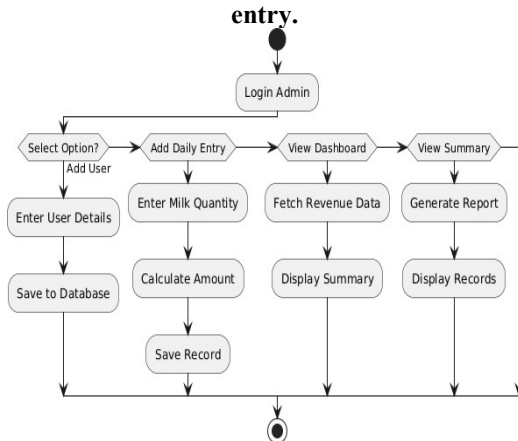


Fig. 5. Activity diagram of the overall workflow.

Figs. 4 and 5 show the same operation from two angles. The sequence diagram traces a daily entry from the form submission through the view, the billing calculation and the database write to the response. The activity diagram places that operation in the wider workflow, from customer registration through daily entry and payment update to summary generation.

E. Input and Output Design

Input is collected through Bootstrap-styled web forms in three groups. Customer registration takes name, mobile number, address, milk quality, rate per litre and shift. Daily entry takes the date and the milk quantity in litres, computes the amount, and takes the payment status. Filter input takes a customer, a shift or a date. Every form applies required-field checks, numeric validation on the quantity, a non-negative constraint, a restricted set of values for payment status, and Django's cross-site request forgery protection.

Output is presented in three forms. The dashboard shows total customers, total revenue, amount received,

amount pending and the most recent transactions. The customer list shows personal and milk information with the shift. The payment summary shows date-wise totals for a selected customer.

V. Implementation

The application is written in Python and runs on Django. The two models of Fig. 3 are declared as Django model classes, and the framework generates the database schema and the migrations from them. The billing calculation is implemented as a method on the milk-record model, so the amount is derived consistently wherever a record is created or edited rather than being recomputed in each view.

The views handle the request cycle: validating the submitted form, invoking the model, and selecting the template. The dashboard view aggregates over the milk records to produce the totals, which keeps the running balances derived from the underlying entries rather than stored separately and allowed to drift. The filter views apply the selected customer, shift or date as query constraints.

Templates are plain Django templates extending a common base, with Bootstrap supplying the grid and the components. The dashboard cards, the customer cards and the summary tables are all rendered from the same base layout, which is what keeps the interface consistent across screens. The application is platform-independent and has been run on both Windows and Linux.

VI. Results

The working screens of the deployed platform are shown below. Customer mobile numbers and addresses visible in the original screenshots have been masked, since they belong to real people.

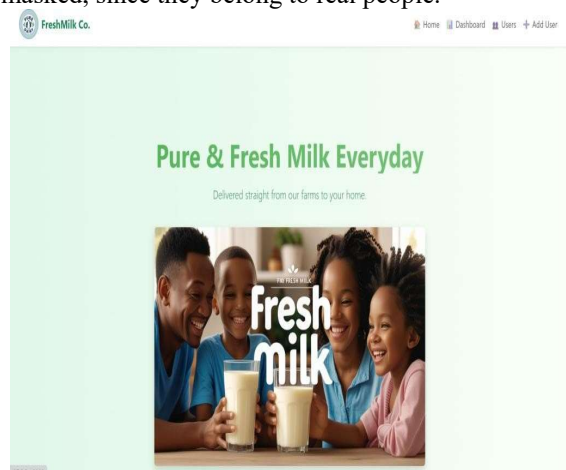


Fig. 6. Home screen with navigation to the dashboard, customer list and registration form.

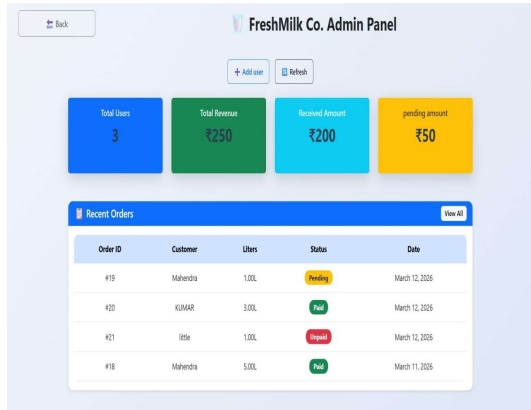


Fig. 7. Dashboard showing totals and recent transactions.

The dashboard of Fig. 7 gives the position of the business in one view: the number of registered customers, the total revenue, the amount received and the amount still pending, with the most recent orders listed below and each one carrying its payment status. This is the screen that replaces the manual month-end exercise, and it is current at every moment rather than once a month.

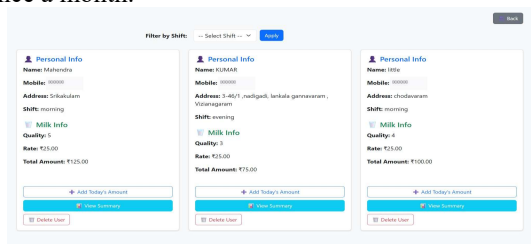


Fig. 8. Customer list with shift filter, personal and milk information per customer.

Fig. 8 shows the customer list. Each card carries the customer's details, the milk quality, the rate and the running total, with buttons to add the day's quantity, view that customer's summary or remove the record. The shift filter at the top restricts the list to morning or evening customers, which matches how the rounds are actually organised.

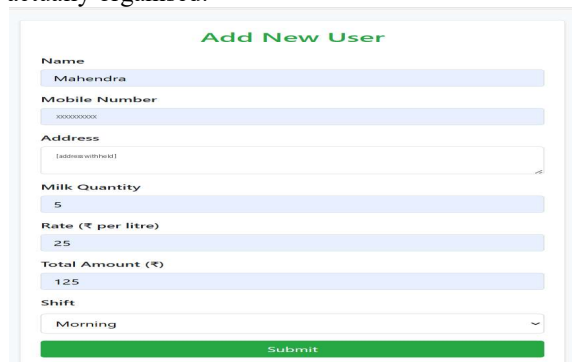


Fig. 9. New customer registration form.

The registration form of Fig. 9 collects the customer record once. The rate and shift entered here are what

the daily entries are subsequently computed and filtered against.

A. Testing

The system was tested module by module and then as a whole. Each module, including registration, daily entry, payment update, dashboard summary and report generation, was verified individually. Form validation was checked for required fields, numeric constraints and non-negative values. Database operations were checked for correct creation, retrieval, update and deletion. Integration testing confirmed that the templates and the views interact correctly and that a payment status change is reflected in the summary totals. Security testing confirmed that cross-site request forgery protection and input validation are in effect. The ten functional test cases are listed in Table II.

TABLE II. Functional Test Cases And Results

Test scenario	Input	Expected output	Result
Add new customer	Valid details	Record saved	Pass
Add with empty field	Name missing	Error shown	Pass
Add daily entry	5 L at Rs. 40	Amount Rs. 200	Pass
Negative quantity	-5	Validation error	Pass
Update payment status	Set to Paid	Status updated	Pass
View dashboard	Existing records	Correct totals	Pass
Filter by shift	Morning	Morning only	Pass
Delete customer	Valid ID	Record removed	Pass
View customer summary	Existing customer	Date-wise records	Pass
Invalid payment status	Random text	Error handled	Pass

All ten cases passed. The testing was functional rather than load-based: the system has not been exercised with a large customer base or with concurrent users, and the performance observed is that of a single administrator working against a small dataset. Establishing behaviour at the scale of a few hundred

customers with a year of daily records is the obvious next step before the platform is relied on in production.

VII. CONCLUSION

A web-based dairy management platform has been built in Python and Django to replace the handwritten register used by small and medium dairies. The system manages customer records, records daily milk quantities, computes bills automatically from quantity and rate, tracks payment status, and reports revenue, receipts and outstanding amounts on a dashboard that is current at all times. Django provides the database handling and the standard web protections; Bootstrap provides an interface that works on a phone as well as a desktop.

The platform removes the arithmetic errors and the record loss that characterise manual bookkeeping, makes outstanding payments visible rather than buried, and reduces month-end settlement from an exercise in re-totalling to a query. It requires no instrumentation and no licence, which is what makes it appropriate for the scale of business it targets. Ten functional test cases covering the core operations all passed.

Several extensions follow naturally. Sensors could supply milk quantity and animal health data directly, writing into the database the platform already provides. A mobile application would let the round be recorded as it happens rather than afterwards. Hosting the system in the cloud would add remote access and offsite backup. Online payment integration would close the loop between the invoice and the settlement. Notifications would cover payment reminders and delivery changes. Role separation between administrator, staff and farmer would allow more than one person to use the system safely. And once a season of data has accumulated, charts of production and revenue trends would make the dashboard predictive as well as descriptive.

References

- [1] S. Wolfert, L. Ge, C. Verdouw and M. J. Bogaardt, "Big data in smart farming – a review," *Agricultural Systems*, vol. 153, pp. 69–80, 2017.
- [2] A. Kamilaris and F. X. Prenafeta-Boldú, "Deep learning in agriculture: a survey," *Computers and Electronics in Agriculture*, vol. 147, pp. 70–90, 2018.
- [3] D. Berckmans, "Precision livestock farming technologies for welfare management," *Animal Frontiers*, vol. 7, no. 1, pp. 18–23, 2017.
- [4] D. Berckmans, "Precision livestock farming technologies for welfare management in intensive livestock systems," *Revue Scientifique et*

Technique (OIE), vol. 33, no. 1, pp. 189–196, 2014.

- [5] S. Neethirajan, "The role of sensors and IoT in dairy farming," *Sensing and Bio-Sensing Research*, vol. 29, Art. no. 100367, 2020.
- [6] R. Elmasri and S. Navathe, *Fundamentals of Database Systems*, 7th ed. Pearson, 2024.
- [7] Django Software Foundation, *Django Documentation*, version 5.x, 2024. [Online]. Available: <https://docs.djangoproject.com>
- [8] Python Software Foundation, *The Python Language Reference*. [Online]. Available: <https://docs.python.org>
- [9] Bootstrap Team, *Bootstrap Documentation*, version 5. [Online]. Available: <https://getbootstrap.com/docs>
- [10] R. Buyya, J. Broberg and A. Goscinski, *Cloud Computing: Principles and Paradigms*. Wiley, 2011.