Integrated Smart Farming System (ISFS)

Comprehensive System Document Report

Team: Rohan, Gaurav Singh
Supervisor: Dr. N. Srinivas Naik
Department of Computer Science — Database Management Systems

October 29, 2025

Institution: Department of Computer Science
Course: Database Management Systems
Academic Year: 2025–2026

Declaration: This report documents the design, implementation, testing and evaluation of the ISFS platform developed as part of the DBMS course project.

Contents

1 Executive Summary

The Integrated Smart Farming System (ISFS) is a full-stack Oracle + Node.js + React platform that centralizes farm operations and field telemetry to enable better decisions and reduce risk. It supports:

- Farmer onboarding, JWT-based authentication, role-based access (farmer/admin).
- CRUD for farms, crops, labor, equipment, fertilizers, and sales with validation.
- Sensor ingestion (temperature, humidity, soil moisture, soil pH, light) with farmer-specific thresholds and automated alerts (in-app; optional SMS/Email).
- Oracle-powered persistence using sequences, triggers for data integrity, and views for analytics.
- Admin analytics: readings distribution, critical/warning counts, alert delivery stats, and CSV exports.

This document details requirements, architecture, database schema, APIs, security, deployment, testing, performance, governance, and research related to smart farming information systems.

2 Problem Statement

2.1 Context and Need

Small and mid-scale farms struggle with fragmented record-keeping, limited telemetry, and late feedback loops. ISFS unifies operations and sensing with a consistent data model and actionable dashboards.

2.2 Challenges Addressed

- Disconnected subsystems: operations vs. telemetry vs. analytics.
- No farmer-specific threshold configuration for alerts.
- Manual workflows increase error rates and delay responses.
- Lack of consolidated schema, views, and reports.

3 Introduction

3.1 Background

Modern agriculture increasingly relies on continuous sensing and structured data to optimize inputs and protect yields. Oracle DBMS provides industrial-grade features (sequences, constraints, triggers, views) to build robust systems.

3.2 Motivation

- Provide an end-to-end, production-grade data platform for farms.
- Ensure correctness (constraints), reliability (sequences), and visibility (views).
- Offer a usable interface for farmers and administrators.

3.3 Objectives

- Unify operations and telemetry with a single source of truth.
- Standardize alerting with thresholds per farmer and sensor type.
- Expose admin analytics and data exports for oversight and reporting.

3.4 Scope

Farmer registration, farm/crop management, labor/equipment/fertilizer tracking, sales, sensor telemetry, alerts, notifications, admin analytics, and CSV exports.

4 Literature Review and Related Work

4.1 Smart Farming Information Systems

Industrial and academic systems emphasize pipelines from field sensors to dashboards:

- Sensor suites for climate/soil monitoring; data consolidation in relational stores.
- Threshold-based alerts to mitigate stress and minimize yield loss.
- KPI dashboards: yields, input costs, monthly revenue.

4.2 Database-Centric Telemetry

- Time-stamped inserts into normalized tables; indexes on FKs and dates.
- Sequences for write-heavy ingestion with minimal contention.
- Views for aggregated analytics to accelerate frontends.

4.3 Operational Records and Integrity

- Constraints and triggers maintain integrity (e.g., farmer totals).
- Referential integrity ensures cascades (e.g., deleting a farm removes dependent rows).

4.4 Human Factors and UX

- Simple forms, low-friction flows, and clear feedback loops improve adoption.
- Role-based access reduces cognitive load and protects sensitive operations.

4.5 Research Synthesis

ISFS aligns with best practices: normalized schema, constraint-driven integrity, sequence-backed IDs, filtered endpoints, and dashboards powered by views. It contributes a cohesive integration of telemetry and operations tailored to farmer workflows.

5 Requirements

5.1 Functional Requirements

1. Farmer registration/login; admin login.

2. Manage farms (create/update/deactivate).
3. Manage crops: sowing, harvest dates, yields, status.
4. Manage labor, work logs, and costs; attendance and hours.
5. Manage equipment and maintenance; schedules and costs.
6. Manage fertilizers: type, quantity, application method, costs.
7. Manage sales: buyer, quantities, prices, totals, payment status.
8. Ingest sensor readings; validate farm ownership for farmers.
9. Configure thresholds per farmer/sensor; defaults when missing.
10. Create alerts on critical breaches; store delivery audit.
11. Admin analytics: stats, alert histories, exports.

5.2 Non-Functional Requirements

- Security: JWT, protected routes, minimal scopes.
- Reliability: robust error handling; DB constraints.
- Performance: indexing, views, lean payloads.
- Maintainability: modular services, clear naming.
- Portability: environment-based configuration.

6 Architecture and Design

6.1 Component Diagram (Narrative)

Frontend: React application with protected routes (farmer/admin), quick actions, thresholds editor, sensor reading forms, dashboards.
Backend: Node.js/Express routes for auth, operations, sensors, thresholds, alerts, analytics. Services encapsulate DB logic and notification stubs.
Database: Oracle schema with sequences, constraints, triggers, and views. Consolidated setup script.

6.2 Deployment View

Development: React dev server, Node server, Oracle DB (local or remote).
Production: Backend as a Node service behind reverse proxy; Oracle DB managed; environment variables injected at runtime.

6.3 Sequence of Operations (Typical)

1. Farmer logs in and registers farms and crops.
2. Farmer sets thresholds or uses system defaults.
3. Farmer submits readings; backend stores and classifies status.
4. Critical readings become alerts with delivery audit; notifications persisted.
5. Admin queries sensor stats and alert histories; exports data for reporting.

7 Data Model and Schema

7.1 Entities

FARMER, ADMIN, FARM, CROP, LABOUR, LABOURWORK, EQUIPMENT, EQUIPMENT_MAINTENAN FERTILIZER, SALES, WEATHER_DATA, WEATHER_ALERT, ALERT_PREFERENCES, SENSOR_DATA, SENSOR_THRESHOLDS, SENSOR_ALERTS.

7.2 Constraints and Integrity

- PKs via sequences; FKs with cascades for dependent rows.
- CHECK constraints enforce allowed statuses.
- UNIQUE constraints for phone and emails.

7.3 Triggers

TRG_FARM_INSERT/UPDATE/DELETE maintain farmer's `TOTAL_FARMS` and `TOTAL_AREA` to keep dashboards consistent without heavy queries.

7.4 Views

- **FARMER_DASHBOARD**: totals across farms, crops, yields, and revenue.
- **FARM_PERFORMANCE**: farm-level KPIs including efficiency metrics.
- **CROP_ANALYTICS**: aggregate crop yields and price trends.
- **MONTHLY_REVENUE**: monthly revenue summaries.

7.5 Selected Data Dictionary (Extended)

Table	Column	Description
FARMER	farmer_id (PK)	Sequence-backed unique ID
	phone (UNQ)	Contact, unique
	total_farms, total_area	Derived via triggers
FARM	farm_id (PK)	Unique farm ID
	farmer_id (FK)	Owner link
	soil_type, soil_ph	Optional soil descriptors
CROP	crop_id (PK)	Unique crop ID
	farm_id (FK)	Parent farm link
	expected/actual_harvest_date	Schedules
SENSOR_DATA	sensor_id (PK)	Unique reading ID
	sensor_type	TEMPERATURE/HUMIDITY/...
	sensor_value, unit	Input value and unit

	status	NORMAL/WARNING/CRITICAL
SENSOR_THRESHOLDS	threshold_id (PK) critical_min/max	Per farmer and sensor type Critical bounds used for status
SENSOR_ALERTS	alert_id (PK) sms/email/notif flags	Persistent audit of events Delivery metadata

8 Sequences and Trigger Logic

8.1 Sequences

Core entity sequences plus telemetry sequences: SENSOR_DATA_SEQ, SENSOR_THRESHOLD_SEQ, SENSOR_ALERT_SEQ.

8.2 Trigger Behavior

On farm insert/update/delete, update farmer totals atomically; ensures analytics views remain accurate with near-zero maintenance burden.

9 API Design

9.1 Auth

- `POST /api/auth/login` → { token, farmerId, name }
- `POST /api/admin/login` → { token }

9.2 Sensor Endpoints (Farmer)

- `POST /api/sensors/reading` (farmId, sensorType, value, unit?, notes?)
- `GET /api/sensors/readings` (filters: farmId, sensorType, dates, status)
- `GET /api/sensors/thresholds` (optionally by sensorType)
- `PUT /api/sensors/thresholds` (sensorType, criticalMin?, criticalMax?, useDefaults?)

9.3 Sensor Endpoints (Admin)

- `POST /api/admin/sensors/reading`
- `GET /api/admin/sensors/alerts`
- `GET /api/admin/sensors/stats`
- `GET /api/admin/sensors/export`

9.4 Operations (Examples)

- `/api/farms`: create/update/list; ownership checks.
- `/api/crops`: lifecycle management.
- `/api/labours`, `/api/labour-work`: work logs and costs.

- `/api/equipment`: inventory and maintenance.
- `/api/fertilizers`: applications and costs.
- `/api/sales`: revenue tracking and buyer records.

10 Security Model

10.1 Authentication & Authorization

JWT tokens, role claims, and middleware for route protection. Farmers restricted to their resources; admins access analytics and global tools.

10.2 Input Validation

Strict validation on farmId ownership, sensor types, date ranges, and numeric bounds to protect the DBMS and ensure data quality.

10.3 Secrets Handling

Secrets never in VCS; environment variables and rotation policies if exposure suspected.

11 Deployment and Operations

11.1 Database Setup

Execute consolidated script once:

```
@ISFS_backend/database/final_setup.sql
```

11.2 Backend

```
cd ISFS_backend
cp .env.example .env   # Set Oracle and JWT values
npm i
npm run dev
# http://localhost:5000/api
```

11.3 Frontend

```
cd ISFS_frontend
npm i
npm start
```

12 Testing and Validation

12.1 Strategy

- **Unit Tests**: services and utility functions.
- **API Tests**: Postman collections for success and negative cases.
- **DB Tests**: constraints, triggers, sequences, and views cross-checked.

12.2 Representative Tests

1. Threshold Upsert: create defaults, override min/max, assert persisted values.
2. Critical Alerts: submit out-of-range readings, assert alert row and flags.
3. Ownership Checks: ensure farmer cannot submit reading for others' farms.

12.3 Results Summary

- Alerting works for high/low critical ranges across sensor types.
- Admin stats align with aggregate queries; CSV exports open correctly.

13 Performance Engineering

13.1 Indexes

FKs on farmId and farmerId; compound indexes for (farmId, recorded_date) and distributions for `sensorType`, minimizing scan time.

13.2 Views and Aggregations

Pre-aggregated views reduce compute on the client; live recompute on modest volumes keeps data fresh.

14 Governance, Ethics, and Compliance

14.1 Data Governance

- Data ownership retained by farmers; admins access aggregates and alert logs.
- Minimal retention for sensitive contact information.

14.2 Ethical Use

- System designed for agronomic support, not enforcement or punitive actions.
- Notifications aim to inform and reduce risk; opt-out mechanisms respected.

14.3 Compliance

- Secrets and credentials handled via environment management.
- Auditability through alert and notification logs.

15 Project Management

15.1 Work Breakdown Structure

- WBS-1: Requirements and schema design.
- WBS-2: Backend routes and services.
- WBS-3: Frontend dashboards and forms.
- WBS-4: Sensor ingestion and thresholds/alerts.
- WBS-5: Admin analytics and exports.
- WBS-6: Testing, tuning, documentation.

15.2 Risk Register

ID	Risk	Mitigation
R1	Sequence drift	Align on registration; helper procedure
R2	Secret leakage	Env variables; rotate on suspicion
R3	Data quality	Validation, constraints, typed fields
R4	Performance	Indexing and views; lean payloads

15.3 Timeline

- Weeks 1–2: Requirements, schema.
- Weeks 3–5: Backend core, operations CRUD.
- Weeks 6–7: Sensors, thresholds, alerts.
- Weeks 8–9: Admin analytics, exports.
- Week 10: Testing, polish, documentation.

16 User Guides

16.1 Farmer

1. Login; create farms; add crops.
2. Set or confirm thresholds; submit sensor readings.
3. Review notifications; respond to alerts.

16.2 Admin

1. Login; open dashboards.
2. Inspect stats and filters; export datasets as CSV.

17 Extended Research Notes

17.1 Telemetry and Agronomic Decision Support

Multi-parameter monitoring (e.g., temperature with humidity; soil moisture with pH) improves signal over single-parameter alerts. Thresholds are most useful when configurable per farm and season.

17.2 Relational Design in Agriculture

Normalized models with enforced constraints reduce inconsistencies; sequences simplify ID generation under concurrent inserts. Views accelerate dashboards without duplicating data.

17.3 Notification Practices

In-app notifications serve as low-latency, zero-cost defaults. Optional channels help in limited-connectivity contexts; auditing delivery improves trust.

18 Detailed Use-Case Scenarios

18.1 UC-01: Farmer Submits Sensor Readings

Primary Actor: Farmer
Goal: Record a reading and trigger alerts if out of bounds.
Preconditions: Farmer is authenticated; owns at least one farm.
Main Flow:

1. Farmer navigates to "Submit Reading".
2. Selects farm, sensor type (TEMPERATURE/HUMIDITY/SOIL_MOISTURE/SOIL_PH/LIGHT), enters value and unit.
3. Submits; backend creates `SENSOR_DATA` row and evaluates thresholds.
4. If CRITICAL, backend writes `SENSOR_ALERTS` with delivery flags.
5. Farmer sees confirmation and any alert surfaced in notifications.

Alternate: If farmId not owned, 403. If sensorType unknown, 400.

18.2 UC-02: Admin Reviews Alerts

Actor: Admin
Goal: Review historical alerts and export data.
Flow:

1. Admin opens "Alerts" and sets filters.
2. Backend joins `SENSOR_ALERTS`, `FARM`, `FARMER`.
3. Results include delivery flags and statuses.
4. Admin exports CSV as needed.

18.3 UC-03: Farmer Manages Thresholds

Actor: Farmer
Goal: Configure thresholds for a sensor type.
Flow:

1. Farmer opens "Thresholds".
2. Chooses sensor type; uses defaults or sets custom criticalMin/criticalMax.
3. System upserts a row in `SENSOR_THRESHOLDS`.

19 Cost–Benefit and Impact Analysis

19.1 Operational Costs

- Database hosting (Oracle instance or managed service).
- Backend hosting (Node.js), reverse proxy.
- SMS/Email providers (optional channels).
- Maintenance (patches, backups/restores).

19.2 Tangible Benefits

- Risk reduction via early warnings.
- Time savings from consolidated records and dashboards.
- Decision support from KPIs and views.

19.3 Intangible Benefits

- Data hygiene through constraints and validation.
- Audit trails improve trust and accountability.

19.4 ROI Illustration (Hypothetical)

Item	**Annual**	**Notes**
Hosting	Moderate	Oracle + Node host sized to data rate
Alerts	Variable	Based on SMS/Email usage; in-app is free
Labor Savings	Significant	Fewer hours on manual collation
Yield Protection	Significant	Fewer stress events via early warning

20 Extended Test Plan Matrix

20.1 Functional Test Matrix

Feature	**Result**	**Procedure**
Auth (Farmer)	Pass	Valid credentials → token; invalid → 401

Auth (Admin)	Pass	Valid admin credentials → token; invalid → 401
Farm CRUD	Pass	Create/Update/Delete with ownership checks
Sensor Ingest	Pass	Submit reading; create row in `SENSOR_DATA`
Thresholds Upsert	Pass	PUT → row updated/inserted
Alerts	Pass	Critical reading → row in `SENSOR_ALERTS`
Admin Alerts List	Pass	Filters by type/farm/date
CSV Export	Pass	File opens in spreadsheet

20.2 Negative Tests

- Submit reading on unowned farm → 403.
- Unknown sensor type → 400.
- Thresholds min > max → 400.
- Duplicate phone → 409.

21 Operations Runbook (SRE View)

21.1 Routine Tasks

- Backups; test restores quarterly.
- DB stats for query planning.
- Log review: ingestion, alerts, delivery outcomes.
- Capacity: monitor SENSOR_DATA growth; index health.

21.2 Incident Response

- DB errors (5xx): check pool, retry policy, locks.
- Sequence drift: realign via registration logic or helper proc.
- Performance: add indexes, review slow queries, verify view usage.

21.3 Change Management

- Migration scripts; never mutate prod manually.
- Version APIs; document breaks; deprecation windows.

22 Maintenance and Upgrade Plan

22.1 Schema Evolution

- Prefer additive changes.
- For breaking changes, use phased rollout.

22.2 Release Strategy

- Feature branches; code review; CI tests.
- Dev → staging → prod.

22.3 Backup/Restore Drills

Quarterly drills ensure RTO/RPO; validate `SENSOR_DATA`, `SENSOR_ALERTS`, and references.

23 Glossary and Abbreviations

Term	**Definition**
CRUD	Create, Read, Update, Delete operations
JWT	JSON Web Token, used for stateless authentication
FK	Foreign Key (referential integrity)
PK	Primary Key (unique identifier)
KPI	Key Performance Indicator
CSV	Comma-Separated Values (export format)

24 Research Addendum (Topic-Related)

24.1 Telemetry and Decision Support

Multi-parameter monitoring and farmer-specific calibration are emphasized across studies. Combining environmental metrics with crop phenology reduces false positives.

24.2 Database Design for Agricultural Systems

Normalized schemas with strong constraints reduce inconsistency; sequence-backed IDs are reliable for concurrent writes. Views aid dashboards with minimal duplication.

24.3 Alerting and Human-in-the-Loop

Alert fatigue is mitigated by sensible defaults, personal calibration, and audit trails that support end-of-season reviews.

25 Appendices

25.1 Default Thresholds Reference

Sensor Type	Critical Min	Critical Max	Unit
SOIL_MOISTURE	20	80	%
SOIL_PH	5.0	8.5	pH
TEMPERATURE	5	40	°C
HUMIDITY	30	95	%
LIGHT	1000	100000	lux

25.2 Sample API Calls

Submit Reading

```
POST /api/sensors/reading
Authorization: Bearer <token>
{
  "farmId": 1,
  "sensorType": "TEMPERATURE",
  "value": 45,
  "unit": " C ",
  "notes": "critical test"
}
```

Threshold Upsert

```
PUT /api/sensors/thresholds
Authorization: Bearer <token>
{
  "sensorType": "SOIL_MOISTURE",
  "criticalMin": 25,
  "criticalMax": 60
}
```

Admin Alerts

```
GET /api/admin/sensors/alerts?sensorType=TEMPERATURE&startDate
    =2025-10-01
Authorization: Bearer <adminToken>
```

25.3 CSV Export

```
GET /api/admin/sensors/export?farmId=1&sensorType=TEMPERATURE&startDate
    =2025-10-01
```

25.4 Indexing Cheat Sheet

- `FARM(farmer_id)`, `CROP(farm_id)`, `SENSOR_DATA(farm_id, recorded_date)`
- `SENSOR_DATA(sensor_type, status)` for filters
- `SENSOR_ALERTS(farm_id, created_date)` for admin queries

26 References

1. Oracle (2024). *oracledb Node.js Driver and SQL Documentation.* https://oracle.github.io/node-oracledb/
2. React Team (2024). *React Documentation.* https://react.dev/
3. FAO/ITU (2023). *E-agriculture in Action: Drones, Data and Emerging Technologies in Agri-food.* (general background)
4. Academic and industry whitepapers on smart farming telemetry, DBMS constraint design, and alerting best practices (2019–2025).