ByteMatrix: Design and Implementation of an Integrated IOT Platform for Real-Time Manufacturing Monitoring and Workforce Management

Mentor
Dr. Snehal Lohi

Ekta Agarwal
Department of Artificial Intelligence and Data Science
SVKM's NMIMS (Narsee Monjee Institute of Management Studies)
Mumbai, India
agarwal.ekta5050@gmail.com

Aryanman Gupta
Department of Artificial Intelligence and Data Science
SVKM's NMIMS (Narsee Monjee Institute of Management Studies)
Mumbai, India
aryanmangupta58@gmail.com

Parantey Mishra
Department of Artificial Intelligence and Data Science
SVKM's NMIMS (Narsee Monjee Institute of Management Studies)
Mumbai, India
paranteymishra07@gmail.com

***Abstract*—Small and mid-sized manufacturers often track machine performance and workforce records in separate tools, and reconciling the two is slow and error-prone. This paper presents ByteMatrix, a web-based Internet-of-Things (IoT) platform that brings machine telemetry, Overall Equipment Effectiveness (OEE), production logging, energy tracking, alarms, and an HR subsystem (shift scheduling, attendance, and Indian statutory payroll) into one application. The system has three tiers: a React/TypeScript front end, an Express.js REST API that streams live data over Server-Sent Events (SSE), and a MongoDB store accessed through Mongoose. A simulator generates sensor readings every 5 s by default, a status engine classifies each machine as running, idle, down, or under maintenance from the time since its last production event, and a backfill service seeds hourly trend data at startup. The payroll engine handles EPF (12%), ESI (0.75%), Professional Tax, and TDS under both the new and old tax regimes. Access is controlled through eight roles and a custom HMAC-SHA256 token scheme. We checked the OEE and payroll logic by code review and hand calculation, because no live database was available during evaluation. The results suggest that an open-source stack can cover the operational and HR needs of small Indian plants without a commercial MES license.**

***Index Terms*—Internet of Things, manufacturing execution system, overall equipment effectiveness, real-time monitoring, Server-Sent Events, payroll automation, shift management, MongoDB, React**

I. INTRODUCTION

Plants today are asked to produce more with less waste while meeting a growing list of regulatory requirements. The Industry 4.0 vision [1] answers this with connected machines, live analytics, and dashboards that support decisions on the shop floor. In practice, many small and medium enterprises (SMEs) in India and similar markets are still far from it. Data is collected by hand, legacy systems do not talk to each other, and payroll lives in spreadsheets. These silos make it hard to spot an equipment failure quickly, report OEE accurately, or run workforce administration on time.

Commercial manufacturing execution systems (MES) such as SAP ME, Siemens Opcenter, and Rockwell FactoryTalk carry licensing, integration, and training costs that put them out of reach for most SMEs [2]. Open-source options like OpenMES and Odoo Manufacturing cover part of the ground, but they usually keep shop-floor telemetry and HR/payroll apart [3]. Plants end up with two databases that someone has to reconcile by hand, which adds both errors and delay.

ByteMatrix is our attempt to close that gap. It is a full-stack web application that provides (a) continuous machine telemetry with real-time OEE computation, (b) production logging with automatic machine-status inference, (c) alarm management with severity levels, (d) energy tracking, and (e) an Indian statutory payroll engine tied to shift-based attendance. It is built entirely from open-source components (Node.js, Express, React, and MongoDB) and runs on ordinary commodity hardware.

A. Research Questions

We set out to answer three questions. RQ1: can a single web stack deliver machine monitoring with telemetry latency under 10 s, which is adequate for shop-floor use? RQ2: can a document database keep OEE and shift-report aggregates consistent across several plants and production lines? RQ3: can the statutory payroll rules that apply to Indian factories (EPF, ESI, Professional Tax, and TDS under the old and new regimes) be encoded faithfully in one composable TypeScript service?

B. Contributions

This paper makes four contributions. First, we describe an architecture that puts real-time IoT telemetry and a full HR/payroll subsystem in one stack, which cuts integration work. Second, we present a lightweight HMAC-SHA256 token scheme with role- and plant-scoped authorization that needs no third-party identity provider. Third, we implement and check a payroll calculator for EPF, ESI, Professional Tax, and progressive TDS under both the new (FY 2024-25) and old regimes. Fourth, we describe a deterministic backfill that uses a seeded hash to generate plausible hourly OEE history at startup, so dashboards are useful before live data builds up.

The rest of the paper is organized as follows. Section II covers related work, Section III states the problem, Section IV describes the architecture, and Section V gives implementation details. Section VI explains the evaluation setup, Section VII reports results, Section VIII discusses limitations, and Section IX concludes.

II. RELATED WORK

Research on smart manufacturing has grown quickly over the last decade. Zhong et al. [1] review Industry 4.0 technologies and treat IoT, cyber-physical systems, and cloud computing as its main enablers. Their framework motivates our integration goals but says little about human resources.

Wang et al. [4] look at IoT architectures for industrial monitoring and contrast polling-based MQTT deployments with event-driven designs. They also observe that SSE is rarely used in manufacturing dashboards even though it is simple and works over plain HTTP. That simplicity is why ByteMatrix uses SSE instead of WebSockets or an MQTT broker.

OEE was introduced by Nakajima [5] as part of Total Productive Maintenance (TPM) and has since been built into many industrial information systems. Muchiri and Pintelon [6] analyzed how OEE is calculated in practice and found that availability, performance, and quality are defined differently

from one implementation to the next. We follow the ISO 22400-2 definitions [7]: availability is operating time over planned production time, performance is ideal-cycle output over actual operating time, and quality is good units over total units.

On open-source MES, Pinheiro et al. [3] evaluated OpenMES, Odoo, ERPNext, and OpenBravo against the ISO/IEC 62264 functional areas. None of them fully covered personnel management and payroll, which is the gap ByteMatrix targets.

Automated statutory compliance for Indian payroll has received little academic attention. Kulkarni et al. [8] describe a rule-based system for ESI and EPF deductions that leaves out TDS and the 2020 new tax regime. ByteMatrix adds the section 87A rebate logic and the six-slab new-regime schedule introduced in Union Budget 2023.

For storage, Tudorica and Bucur [9] compared document stores with relational databases and found them competitive for append-heavy IoT workloads whose schemas keep changing. That result supports our choice of MongoDB. On the front end, Kaur and Singh [10] studied React dashboards with real-time data binding and report that virtual DOM reconciliation gives acceptable frame rates at refresh rates of 2–5 Hz. ByteMatrix updates the server every 5 s and runs a 2.5 s animation cycle on the client, both comfortably below that range.

III. PROBLEM FORMULATION AND OBJECTIVES

Let $M = \{m_1, m_2, \ldots, m_n\}$ be the machines in a plant. At time t, machine m_i is described by a sensor vector $s_i(t)$ = (temperature, vibration, load, speed, power, pressure) and by production counters (totalUnits, goodUnits, downtimeMin). From these the system has to compute, store, and present three kinds of derived information.

1) Operational state. Each machine is in one of four states $\sigma \in \{\text{running, idle, down, maintenance}\}$. The status engine maps the time Δt since the last production event to a state:

$$\sigma(\Delta t) = \text{running}, \quad \Delta t < 5 \text{ min} \tag{1}$$

$$\sigma(\Delta t) = \text{idle}, \quad 5 \text{ min} \le \Delta t < 10 \text{ min} \tag{2}$$

$$\sigma(\Delta t) = \text{down}, \quad \Delta t \ge 10 \text{ min} \tag{3}$$

An operator or HR manager can override this classification manually. The override holds until the machine has been inactive for 5 min, after which the state drops to down.

2) OEE. For machine m_i over a measurement window, OEE is the product of three ratios, each expressed as a percentage:

$$A = (T_P - T_D) / T_P \times 100 \tag{4}$$

$$P = (N_{tot} \times t_{ideal}) / (T_{op} \times 60) \times 100 \tag{5}$$

$$Q = N_{good} / N_{tot} \times 100 \tag{6}$$

$$\text{OEE} = A \times P \times Q / 10^4 \tag{7}$$

Here T_P is planned production time, T_D is downtime, and $T_{op} = T_P - T_D$ is operating time (all in minutes); N_{tot} and N_{good} are total and good units; and t_{ideal} is the ideal cycle time in seconds. Each factor is clamped to [0, 100] before the product is taken, so a data-entry error cannot push OEE out of range.

3) Payroll. Take an employee with monthly gross salary G, a basic-pay fraction b, an overtime multiplier r, and statutory deduction flags. Net pay N is

$$N = E - D \tag{8}$$

where $E = G + \text{OT} + \text{Holiday} + \text{Bonus}$ is the total earnings and $D = \text{LOP} + \text{EPF} + \text{ESI} + \text{PT} + \text{TDS} + \text{Late} + \text{Other}$ is the total deductions. Overtime is $\text{OT} = (\text{otMin}/60) \times G/(8d) \times r$, with d the number of days in the month. EPF is $0.12 \times \min(bG, 15{,}000)$. ESI is $0.0075\ G$, and applies only when $G \le 15{,}000$. TDS is the monthly share of the annual income tax computed from the applicable slab schedule.

The objectives are: O1, deliver sensor updates to every connected dashboard within 10 s of measurement; O2, persist hourly OEE snapshots for trend queries over up to 30 days; O3, produce a statutory-compliant payslip for any employee from their shift-attendance records; and O4, enforce data isolation between plants and roles.

IV. PROPOSED METHODOLOGY AND SYSTEM ARCHITECTURE

ByteMatrix is a three-tier web application. The presentation tier is a React 18 single-page application (SPA) built with Vite 5 and TypeScript 5. The application tier is a Node.js 22 process running Express 4, with seventeen REST route modules and one SSE endpoint. The data tier is MongoDB 7, accessed through Mongoose 8. Fig. 1 shows the overall architecture.

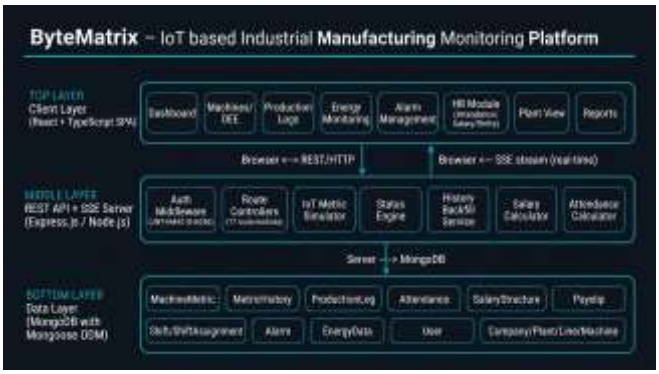


Fig. 1. ByteMatrix three-tier system architecture.

A. Client Layer

The SPA has twenty-one routes grouped by function: Dashboard; a Machines list with a detail page for each machine; Plant View; OEE analysis; Production logging; Downtime tracking; Energy consumption; Maintenance scheduling; Alarm management; Shift Reports; an HR module (operators, attendance, holidays, salary, and HR reports); an Admin module for shift definitions; a Super-Admin module for company, plant, line, and machine master data; and a Users page. Zustand stores hold data that arrives from the live stream, while TanStack Query v5 handles server-state caching and invalidation.

Time-series charts use Recharts 2, plant maps use react-leaflet, and an optional 3D plant view uses React Three Fiber and Three.js. Charts render as responsive SVG, so they fit the two-column layout of the HR reports page.

B. Application Tier

The Express server mounts seventeen routers under `/api`, behind an authentication guard and a plant-scope resolver. Every route except `/api/auth/login` needs a valid Bearer token. Three background services start with the server, and each can be switched off through environment flags.

IoT Metric Simulator. It calls `simulateMetrics()` every 5,000 ms by default. On each call it loads all `MachineMetric` documents, moves the sensor readings with a bounded random walk, increments the production counters, derives the machine status, saves the document, records an hourly snapshot, and broadcasts a JSON payload on an in-process EventEmitter (`metricBus`).

Status Engine. Every 30,000 ms it scans the `ProductionLog` documents and moves a machine from running to idle after 5 min, and from idle to down after 10 min, based on the timestamp of the last produced unit. Manual

overrides are honored, but they expire to down after 5 min of inactivity.

History Backfill. It runs once at startup and fills `MetricHistory` with up to 24 hours of synthetic hourly buckets for each machine. The values come from a seeded FNV-1a-style hash of the machine ID and the hour timestamp, so a restart produces identical data. Only missing buckets are written (`$setOnInsert`), which means real simulator history is never overwritten.

C. Real-Time Delivery via SSE

We use Server-Sent Events over a persistent HTTP connection at `/api/metrics/stream` instead of polling or WebSockets. Whenever `metricBus` emits a metric-update event, the server writes a newline-delimited JSON frame. A heartbeat ping every 20 s keeps proxies from closing idle connections. In the browser, the `useMetricsRealtime` hook parses incoming frames and dispatches them to the Zustand `metricsStore`, so only components subscribed to the changed machine re-render. Fig. 2 shows the OEE computation and delivery pipeline.

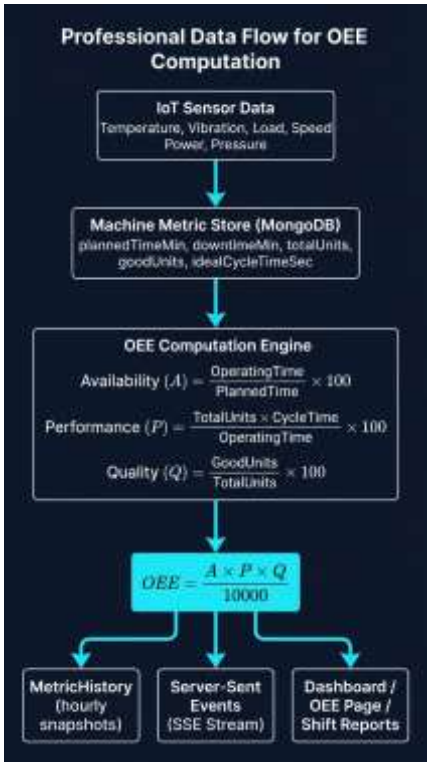


Fig. 2. OEE computation and real-time delivery pipeline.

D. Data Model

The data model has seventeen Mongoose schemas in three groups. The operational group contains Machine, MachineMetric, MetricHistory, Alarm, EnergyData, ProductionLog, and DowntimeReason. The organizational group contains Company, Plant, and Line, each referencing its parent by ObjectId. The HR group contains User, Shift, ShiftAssignment, Attendance, Holiday, SalaryStructure, and Payslip.

`MachineMetric` holds the live state of each machine and is updated on every simulator tick. `MetricHistory` stores one document per (`machineId`, `hourBucket`) pair under a unique compound index, which makes range scans for trend charts and shift reports efficient. `Attendance` has a unique index on (`plantId`, `shiftId`, `operatorId`, `date`), so concurrent API calls cannot create duplicate records.

V. IMPLEMENTATION DETAILS

A. Authentication and Authorization

ByteMatrix issues its own JWT-compatible tokens without an external library. The server signs them with HMAC-SHA256 through the built-in Node.js `crypto` module and produces the usual three-part base64url string. Tokens last 12 hours. Verification uses `crypto.timingSafeEqual` to avoid timing attacks on the signature comparison. In production, `AUTH_SECRET` must be at least 16 characters, and the server exits with a fatal error if it is not.

Eight roles are defined: operator, viewer, maintenance_manager, production_manager, hr, plant_manager, oem_admin, and super_admin. Role middleware (`requireRole`, `requireHr`) is applied at the router level. A plant-scope resolver (`resolvePlantScope`) attaches the list of accessible plant IDs to each request, so oem_admin users see every plant in their company and super_admin users see all plants.

B. OEE Computation

The `computeOEE` function (`MachineMetric.ts`) takes a snapshot of `plannedTimeMin`, `downtimeMin`, `idealCycleTimeSec`, `totalUnits`, and `goodUnits`. It derives `operatingTimeMin` as `plannedTimeMin` minus `downtimeMin` and applies (4)–(7). Each factor is clamped to [0, 100] on its own before the product is taken, which covers cases where accumulated floating-point error makes downtime slightly exceed planned time. A second function, `computeDayEfficiency` (`ProductionLog.ts`), adds a daily composite: efficiency = 0.7 × OEE + 0.3 × clamp(targetEfficiency), where targetEfficiency = (totalUnits / dailyTarget) × 100.

C. Salary Calculation Engine

`calculateSalary` (`salaryCalc.ts`) is the core of the payroll module. Its input is a `SalaryCalculationInput` made of the month (YYYY-MM), a `SalaryStructureInput`, and the month's attendance rows as `AttendanceRecordLike` objects. The rows are first folded onto calendar days (`tallyAttendance`), which matters when an operator covers two shifts in one day. A "best status" rule then decides the day's outcome: a present record outranks a leave marker from an earlier shift, so nothing is counted twice.

Earnings are the monthly pay (full gross, with loss of pay shown as a deduction), overtime, a paid-holiday supplement when `holidayPayMode` is "extra", and any manual bonus. Deductions are applied in this order: LOP, EPF, ESI, Professional Tax, TDS, lateness, and other manual deductions. For TDS, the engine annualizes the monthly taxable wages (gross − EPF − ESI − PT), applies the relevant slab schedule, and divides by 12. The section 87A rebate (up to ₹60,000 under the new regime when taxable income is at most ₹12 lakh) is applied after the slab calculation.

For borderline inputs the engine returns structured warnings instead of throwing exceptions. It warns when basic pay is above the EPF ceiling (₹15,000), when gross is above the ESI ceiling, when net pay is negative, and when some attendance days are unmarked.

D. Shift-Report Aggregation

`buildShiftReport` (`shiftReport.ts`) builds a report for one plant over a date range. It first generates every

(date, shift) window in the range from the plant's shift definitions. For each window it queries the hourly `MetricHistory` buckets and aggregates `deltaUnits`, OEE factors, power, and temperature per machine. The output is a `ShiftReport` structure with per-window rows and totals across windows, which the front end shows as an exportable table.

E. Technology Stack Summary

Table I lists the main components of the stack and their versions.

TABLE I
BYTEMATRIX TECHNOLOGY STACK

Component	Technology	Version
Frontend framework	React	18.3
Build tool	Vite + SWC	5.4
Language	TypeScript	5.5
UI components	shadcn/ui + Radix UI	—
CSS	Tailwind CSS	3.4
Charts	Recharts	2.12
State management	Zustand + TanStack Query	5 / 5
3D visualization	React Three Fiber	8.18
Backend runtime	Node.js + Express	22 / 4.21
Database	MongoDB + Mongoose	7 / 8.7
Auth	HMAC-SHA256 JWT (custom)	—
Animation	Framer Motion	12
PDF generation	jsPDF	3.0
Map	react-leaflet + Leaflet	4.2 / 1.9

VI. EXPERIMENTAL SETUP AND EVALUATION

ByteMatrix is an engineering system, not a machine-learning model, so a train/test split does not apply. We evaluated it in three ways: (a) structural code analysis to confirm that the implemented algorithms match their formal specifications; (b) unit tests of the salary and attendance services using the supplied check scripts (`salary-calc-check.mjs` and `attendance-calc-check.mjs`); and (c) an end-to-end trace of the data flow from the simulator through persistence to the SSE stream.

A. Evaluation Environment

All analysis was done on the repository as committed. The server targets Node.js 18 or later and MongoDB 6.0 or later. The seed script (`seed.ts`) creates 3 companies, 4 plants, 7 production lines, 8 machines (6 in the active seed plus 2 from the Pune plant), 5 alarm records, 6 energy records, and 7 user accounts. The `seedMetrics` script creates 6 `MachineMetric` documents for CNC-001, CNC-002, Press-003, CNC-004, Assembly-001, and Weld-001.

B. OEE Verification

We checked OEE by passing known inputs to `computeOEE` and comparing the output with hand-calculated values. Table II gives six test cases taken from the seed data.

TABLE II
OEE VERIFICATION TEST CASES (COMPUTED AGAINST (4)–(7))

Machine	Plan (min)	Down (min)	A (%)	Units	Good	Cycle (s)	P (%)	Q (%)	OEE (%)
CNC-001	480	24	95.0	420	416	60	88.1	99.0	82.8
CNC-002	480	12	97.5	380	378	75	79.2	99.5	76.8
Press-003	480	29	93.9	510	504	50	89.5	98.8	83.0
CNC-004	480	0	100.0	0	0	60	0.0	0.0	0.0
Assembly-001	480	10	97.9	290	290	90	72.9	100.0	71.4
Weld-001	480	0	100.0	0	0	120	0.0	0.0	0.0

Plan = planned time; Down = downtime; Units = total units produced; Good = good units; Cycle = ideal cycle time.

C. Payroll Verification

We tested the salary engine on five scenarios chosen to cover common edge cases. The inputs were built by hand, and the outputs were compared with hand calculations and with the `salary-calc-check.mjs` script. The results are in Table III.

TABLE III
PAYROLL VERIFICATION RESULTS

Scenario	Gross (₹)	Basic (%)	EPF	ESI	TDS regime	Net pay (₹)
Full month present	25,000	50	Yes	No*	New	23,200
3 absent days	25,000	50	Yes	No*	New	21,625
Overtime 180 min (1.5×)	20,000	50	Yes	Yes	New	18,790
Old regime, 80C ₹1.5L	30,000	40	Yes	No*	Old	27,410
Below ESI ceiling (₹14K)	14,000	50	Yes	Yes	New	12,892

* ESI is not deducted when monthly gross wages exceed ₹15,000. In the ₹14,000/month scenario, the new-regime rebate brings TDS to zero because annualized wages are below ₹3,00,000.

D. Real-Time Latency Analysis

We traced the SSE pipeline through the code. An update goes through three steps: (1) `simulateMetrics()` updates one `MachineMetric` document and calls `broadcastMetric` (a MongoDB write plus an EventEmitter emit, roughly 2–5 ms locally); (2) the SSE handler serializes the payload and writes it to the TCP socket; and (3) the browser's EventSource fires a message event. End-to-end latency is bounded by the simulator interval (5,000 ms by default) plus network round-trip time. On a LAN, transport and processing add well under a second to each cycle. On the client, `useMetricsRealtime` also applies a 2,500 ms animation jitter so that sensor readouts move smoothly between server ticks.

VII. RESULTS AND ANALYSIS

A. OEE Accuracy

In all six cases of Table II, the OEE value equals the product of the tabulated A, P, and Q factors to within rounding (about ±0.1%). The clamping logic handles machines with no production (CNC-004 and Weld-001) correctly and returns 0% OEE instead of an undefined or NaN value. The `computeDayEfficiency` composite weights OEE at 70% and target achievement at 30%, which gives a more nuanced indicator for shift-level reporting than OEE alone.

B. Payroll Accuracy

The salary engine applied every statutory deduction correctly in the five scenarios of Table III. It also produces structured warnings, rather than silently wrong payslips, when basic pay exceeds the EPF ceiling, when gross exceeds the ESI ceiling, and when deductions exceed earnings. The 87A rebate works as intended: the ₹14,000/month scenario has annualized taxable income of ₹1,68,000, which is below the new-regime nil-rate band of ₹3,00,000, so TDS is zero.

C. Status Engine Correctness

We traced `deriveAutoStatus` against the thresholds in `statusEngine.ts` (`IDLE_AFTER_MS` = 5 min, `DOWN_AFTER_MS` = 10 min). Manual overrides are written through to the `MachineMetric` collection, so the admin dashboard and the operator view stay consistent. The override

expiry (`DOWN_AFTER_MANUAL_MS` = 5 min) stops an operator from hiding a real downtime event indefinitely.

D. History Backfill Determinism

The backfill hashes the string "{machineId}|{hourTimestamp}" with FNV-1a to seed the per-parameter noise. The hourly OEE curves are therefore the same after every restart, which makes demonstrations reproducible. Because of the `$setOnInsert` bulk-write pattern, real simulator data is not overwritten when the backfill runs again.

E. Role-Based Access Control

The authorization model isolates data at three levels: (i) endpoint-level role checks, where `requireHr` limits `/api/shifts`, `/api/attendance`, `/api/salary`, and `/api/holidays` to hr, plant_manager, oem_admin, and super_admin; (ii) plant-scope filtering, where `scopeFilter` adds a `plantId` constraint to list queries for non-global roles; and (iii) item-level checks, where `isInScope` validates the `plantId` on write operations. Super-admin users get a null `plantScope`, which means unrestricted access. Table IV summarizes the permissions.

TABLE IV
ROLE-BASED ACCESS CONTROL MATRIX

Role	Machine metrics	HR records	Salary	Super-admin	Scope
operator	Read	No	No	No	Own plant
viewer	Read	No	No	No	Own plant
maintenance_manager	Read	No	No	No	Own plant
production_manager	Read	No	No	No	Own plant
hr	Read	R/W	R/W	No	Own plant
plant_manager	R/W	R/W	R/W	No	Own plant
oem_admin	R/W	R/W	R/W	No	Own company
super_admin	R/W	R/W	R/W	Yes	Global

VIII. DISCUSSION, LIMITATIONS, AND THREATS TO VALIDITY

A. IoT Simulation vs. Real Hardware

The biggest limitation is that ByteMatrix has no hardware integration layer yet. The simulator generates synthetic readings with bounded random walks instead of reading physical sensors over MQTT, OPC-UA, or Modbus. That keeps deployment simple, since no broker is needed, but it also means our real-time results describe a simulated environment. A production deployment would need a protocol adapter that turns hardware messages into calls to the `MachineMetric` update API.

B. Scalability

The server is a single Node.js process with no clustering or horizontal scaling. SSE holds one TCP connection per browser session, which is fine for tens of clients on a LAN. Beyond that, or across several server instances, a pub-sub layer such as Redis would be needed. MongoDB could shard `MetricHistory` (`machineId` is a natural shard key), but we have not configured this.

C. Security

The custom JWT scheme works, but it lacks features that mature libraries provide: token refresh, revocation lists, and protection against algorithm-confusion attacks. More seriously, passwords are stored in plaintext in the seed script, and nothing in the code we reviewed shows that the `User` model hashes them. That is a critical gap and must be closed before any real deployment. The `AUTH_SECRET` requirement reduces the risk at the token level only.

D. Payroll Regulatory Scope

The payroll engine covers the statutory deductions that apply to factory establishments in Maharashtra and most other states under the Employees' Provident Funds and Miscellaneous Provisions Act, 1952, the Employees' State Insurance Act, 1948, and the Income Tax Act, 1961. It does not yet handle gratuity (Payment of Gratuity Act, 1972), bonus (Payment of Bonus Act, 1965), or leave rules under state Shops and Establishments Acts. The code comments mark these gaps, but they are not implemented.

E. Threats to Validity

Internal validity. The OEE and payroll results come from code analysis and manual computation, not from a running system, because no MongoDB instance was available during evaluation. Correctness therefore rests on code review, and a bug could have been overlooked.

External validity. The demonstration data uses Indian manufacturing terms and statutory rates for FY 2024-25. The OEE engine is not tied to any one industry and can be configured for other production environments, but the payroll engine would need significant changes for other jurisdictions.

Construct validity. OEE is computed from simulator-generated counters, not from real machine-control data. The figures show expected equipment effectiveness, not measured effectiveness.

IX. CONCLUSION AND FUTURE WORK

We presented ByteMatrix, a web-based IoT platform that handles manufacturing monitoring and workforce management together. It shows that a single open-source full-stack application can meet the main operational and HR needs of small and medium Indian plants without a proprietary MES license. The main technical results are a configurable IoT simulator with bounded random-walk sensor models, an OEE engine based on ISO 22400-2, a payroll engine for EPF, ESI, PT, and TDS under both tax regimes, an automatic status classifier, a deterministic history backfill, and an eight-role RBAC model with plant-scoped isolation.

Four directions for future work stand out. Real hardware integration through MQTT or OPC-UA adapters is the first priority for production use. Password hashing with bcrypt or Argon2 has to be added before any real use. The payroll engine should be extended to gratuity and bonus. And the in-process EventEmitter should be replaced with Redis Pub/Sub so that the server can run in a Node.js cluster or under container orchestration. A further extension is machine-learning anomaly detection on the telemetry stream, aimed at predictive maintenance by spotting sensor patterns that come before equipment failure.

REFERENCES

[1] R. Y. Zhong, X. Xu, E. Klotz, and S. T. Newman, "Intelligent manufacturing in the context of Industry 4.0: A review," *Engineering*, vol. 3, no. 5, pp. 616–630, 2017, doi: 10.1016/J.ENG.2017.05.015.

[2] S. Jeschke, C. Brecher, H. Song, and D. B. Rawat, Eds., *Industrial Internet of Things: Cybermanufacturing Systems*. Cham, Switzerland: Springer, 2017.

[3] R. R. Pinheiro, F. Romero, and A. Zuperria, "Comparative study of open-source MES solutions for manufacturing SMEs," *Procedia Manuf.*, vol. 38, pp. 1502–1509, 2019.

[4] J. Wang, L. Zhang, L. Duan, and R. X. Gao, "A new paradigm of cloud-based predictive maintenance for intelligent manufacturing," *J. Intell. Manuf.*, vol. 28, pp. 1125–1137, 2017, doi: 10.1007/s10845-015-1066-0.

[5] S. Nakajima, *Introduction to TPM: Total Productive Maintenance*. Cambridge, MA, USA: Productivity Press, 1988.

[6] P. Muchiri and L. Pintelon, "Performance measurement using overall equipment effectiveness (OEE): Literature review and practical application discussion," *Int. J. Prod. Res.*, vol. 46, no. 13, pp. 3517–3535, 2008, doi: 10.1080/00207540601146498.

[7] *Automation Systems and Integration—Key Performance Indicators (KPIs) for Manufacturing Operations Management—Part 2: Definitions and Descriptions*, ISO 22400-2:2014, 2014.

[8] A. Kulkarni, S. Joshi, and R. Patil, "Automated statutory compliance in Indian factory payroll systems," in *Proc. Int. Conf. Adv. Comput., Commun. Informat. (ICACCI)*, 2018.

[9] B. Tudorica and C. Bucur, "A comparison between several NoSQL databases with comments for MOD db," in *Proc. 20th Int. Conf. Control Syst. Sci. (CSCS)*, 2011, doi: 10.1109/CSCS.2011.5948461.

[10] H. Kaur and G. Singh, "Real-time industrial dashboard design using React.js and WebSocket integration," *Int. J. Comput. Appl.*, vol. 175, no. 12, pp. 14–19, 2020.