International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 10 | Oct - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

CampusCore: An Integrated Cloud-Connected Smart Campus Platform with RFID Attendance and a Tamper-Evident Audit Trail

Mohammed Taha Shariff[1], Naushin[2], Nafisa[3], Sanjana S D[4]

[1]Student, Department of Computer Science & Engineering, Kalpataru Institute of Technology, Tiptur, Karnataka, India
[2]Student, Department of Computer Science & Engineering, Kalpataru Institute of Technology, Tiptur, Karnataka, India
[3]Student, Department of Computer Science & Engineering, Kalpataru Institute of Technology, Tiptur, Karnataka, India
[4]Student, Department of Computer Science & Engineering, Kalpataru Institute of Technology, Tiptur, Karnataka, India

---

Abstract - Colleges that still depend on manual roll-calls, paper registers and scattered messaging groups suffer from proxy attendance, delayed visibility, fragmented communication and weak audit trails. This paper presents CampusCore, an integrated, cloud-connected design for college management and smart campus intelligence, aimed at institutions of 500 to 5,000 students. The platform has four layers: an IoT edge (an ESP8266 hub with an RC522 RFID reader, relay, buzzer and LCD), a Node.js and Express backend with Socket.io, Node-cron, JWT and bcrypt, a PostgreSQL and Redis data tier with a hash-chained audit trail, and a React single-page application with role-based dashboards. A rule engine converts card taps into per-period attendance using timetable and grace-period rules, while an assessment module, class feed, digital library and automated guardian e-mail alerts operate under six-role access control. We present the requirements, architecture, workflow, schema, API, security controls, fault-tolerance behaviour and an 89-case test plan. CampusCore is a design-stage contribution: no hardware is deployed, the blockchain trail is conceptual, and the 150 ms response, 60 s alert and 500 scans/s figures are design targets, not measured results.

Key Words: *smart campus, RFID attendance, Internet of Things, role-based access control, audit chain, Node.js.*

1. INTRODUCTION

A college runs several operations in parallel: attendance, internal assessment, announcements, library circulation and communication with parents. In many institutions these run on separate tools, ranging from paper registers to spreadsheets and informal messaging groups. National policy encourages greater use of technology in education [7], yet the daily mechanics of attendance and record keeping often remain manual.

Four problems recur. First, manual roll-calls consume class time and allow proxy responses. Second, attendance and marks are consolidated late, so faculty, students and parents see the position only after the damage is done. Third, communication is spread across channels that leave no structured record. Fourth, registers can be altered or lost, so there is little basis for trusting a record after the fact.

Radio-frequency identification (RFID) is a natural basis for attendance capture because a tag responds to a reader and converts a tap into a timestamped event [1], [2]. However, capturing a tap is only the first step. The value to an institution lies in what happens next: validating the tap against the timetable, updating dashboards, alerting guardians and keeping a record that is hard to alter silently. This paper therefore treats RFID capture as the entry point of an integrated platform rather than a stand-alone gadget.

Contributions. (i) A four-layer architecture that separates IoT edge, backend logic, data and cache, and presentation. (ii) An attendance pipeline with signed reader payloads and a rule engine that applies timetable and grace-period logic per subject and per period. (iii) An academic hub (assessment and class feed), a digital library and an event-driven guardian alert system, all under six-role access control. (iv) A hash-chained audit design with a path to a permissioned ledger. (v) A validation plan of 89 test cases that keeps design targets clearly separate from measurements.

Status of the work. CampusCore is at the design stage. The requirements, schema, interfaces and wireframes are complete, but the RFID hardware has not been procured, the backend and frontend have not been built, and the audit chain is conceptual. Every quantitative figure below that concerns time or capacity is a design target. The rest of the paper is organised as follows: Section 2 reviews related work, Section 3 gives requirements, Sections 4 to 9 present the design, Section 10 gives implementation status and the evaluation plan, and Sections 11 and 12 discuss limitations and conclude.

2. RELATED WORK

2.1 RFID-based attendance

In RFID attendance, each student carries a card with a unique identifier and a reader at the classroom entrance turns a tap into a record. High-frequency (HF) cards at 13.56 MHz are powered by inductive coupling and read over a few centimetres [2], which suits a deliberate tap at a door and limits accidental reads; the RC522 module used here targets this band [3]. Web-server based RFID attendance systems have long been described, in which readers feed a server that records attendance semi-automatically [4]. Such work establishes feasibility of the capture step, but typically stops at attendance records.

2.2 IoT and smart classroom platforms

Recent work extends attendance in other directions. Wang and Pang combine IoT deployment, dynamic QR codes and deep learning to manage attendance at campus activities [5]. Ainebyona et al. pair camera-based attendance with facial emotion classification and a cloud dashboard to monitor engagement [6]. These studies show the breadth of the smart classroom space, but each concentrates on one capability: capture method or analytics. None of the sources we reviewed describes one documented architecture that joins card-based capture, timetable rules, role-based dashboards, guardian alerts, library services and a tamper-evident audit record.

2.3 Attendance capture alternatives

Table -1 compares common capture approaches in qualitative terms. The comparison reflects general engineering considerations, not measurements. RFID is chosen because it is fast, inexpensive and works with existing ID cards; its main

weakness, card lending, motivates the QR and biometric options reserved for future work.

Table -1: Qualitative comparison of attendance capture approaches

Approach	Strengths	Weaknesses
Manual roll-call	No equipment	Slow; proxy responses; transcription errors
Paper register	Simple; low cost	Late consolidation; easy to alter or lose
QR code scan	Low cost; uses phones	Codes can be shared; needs phone
RFID card tap	Fast; low cost; reuses ID cards	Cards can be lent; hardware upkeep
Biometric / face	Strong identity binding	Higher cost; privacy concerns

Table -2 positions the sources against CampusCore. The entries summarise what each source abstract describes; a dash means that the capability is not described there, not that the study lacks it.

Table -2: Related work against CampusCore capabilities

Work	Capture	Analytics	Integrated modules	Audit trail
Ref. [4]	RFID, web server	–	–	–
Ref. [5]	Dynamic QR codes	Deep learning on logs	–	–
Ref. [6]	Camera, face	Emotion classes	Cloud dashboard	–
CampusCore	RFID card tap	Rule engine; alerts	Six modules	Hash chain (design)

2.4 Engineering foundations

The design relies on established building blocks rather than new primitives: REST for stateless interfaces [8], role-based access control (RBAC) [9], JSON Web Tokens (JWT) for session claims [10], HMAC for message authentication [11], TLS 1.3 for transport protection [12], and adaptive salted password hashing with bcrypt [13]. For record integrity, linking each entry to a hash of its predecessor follows the classic idea of cryptographically time-stamped documents [14]. For a distributed deployment, Hyperledger Fabric is a permissioned ledger platform suited to known participants [15], [16].

2.5 Research gap

Three gaps motivate this work. Capabilities such as attendance, assessment, communication, library services and audit logging are usually treated separately, while a college needs them in one system. Design guidance sized for mid-sized colleges of 500 to 5,000 students with modest hardware cost is thin. Finally, targets are often stated without a validation plan. CampusCore addresses these by specifying one integrated architecture with traceable requirements and an explicit test plan. It does not claim to outperform any individual study.

3. REQUIREMENTS

Six roles interact with the platform: Administrator, Lecturer, Student, Class Representative (CR), Librarian and Parent. Their needs were consolidated into functional and non-functional requirements. The principal functional requirements are: (a) record attendance by card tap and apply timetable and grace-period rules; (b) show live attendance to authorised roles; (c) let lecturers enter internal assessment, quiz and practical marks, with automatic grading and later locking; (d) publish structured class posts; (e) issue and return library books with due-date tracking and fines; (f) send guardian alerts for absence, low attendance and repeated absence; and (g) record every attendance, administrative and alert event in a tamper-evident log.

Table -3 summarises what each role needs from the platform. The administrator needs an institution-wide live view and configuration control; the lecturer needs class-level attendance, marks entry and posting; the student and parent need read-only personal views and, for parents, timely alerts.

Table -3: Stakeholders and principal needs

Role	Principal needs
Administrator	Institution-wide live attendance, analytics, configuration, broadcast and full audit access
Lecturer	Class-level real-time attendance; marks entry; class feed posting; timetable management
Student / CR	Personal attendance and marks (read-only); class feed; CRs may post to the feed
Librarian	Catalogue management; issue, return and fine handling
Parent	Ward summary, alert e-mails and read-only marks through a token-authenticated portal

The central use case, recording attendance by card tap, is specified in Table -4. Its alternate flows matter as much as the main flow: an unregistered card or a room with no scheduled class leads to denial and a logged event, and a late scan is marked Absent.

Table -4: Use case UC-S1: record attendance by card tap

Item	Description
Actor	Student, with the reader and backend as supporting components
Precondition	Student enrolled with a registered card; a class is scheduled in the room
Main flow	Tap; reader extracts UID; edge sends HMAC-protected payload over HTTPS; backend checks timetable and 10-minute grace period; record written and audit block appended; relay, buzzer and LCD show result; dashboards update live
Alternate flows	Unregistered card or no class: denied, logged. Late scan: Absent. Network down: configured local policy
Postcondition	Attendance and audit entries exist; alert conditions evaluated

Table -5 lists the quantitative non-functional targets. All are targets that remain to be verified.

Table -5: Principal quantitative non-functional targets

ID	Requirement	Status
NFR-09	Card scan to relay response below 150 ms	Target; not measured
NFR-10	Alert dispatch below 60 s	Target; not measured
NFR-11	500 concurrent scan events per second	Target; not measured
NFR-13	Serve 500 to 5,000 students without redesign	Design requirement
NFR-14	Stateless services for horizontal scaling	Design requirement
NFR-15	Load simulation of 100,000 concurrent users	Planned

4. SYSTEM ARCHITECTURE

Six design objectives guided the architecture: separation of concerns, a short path from tap to door response, fault isolation, role-differentiated access, auditability of every attendance and administrative event, and horizontal scalability. The platform realises them as four layers (Fig -1). Adjacent layers talk through REST interfaces, with WebSockets carrying live updates to the browser.

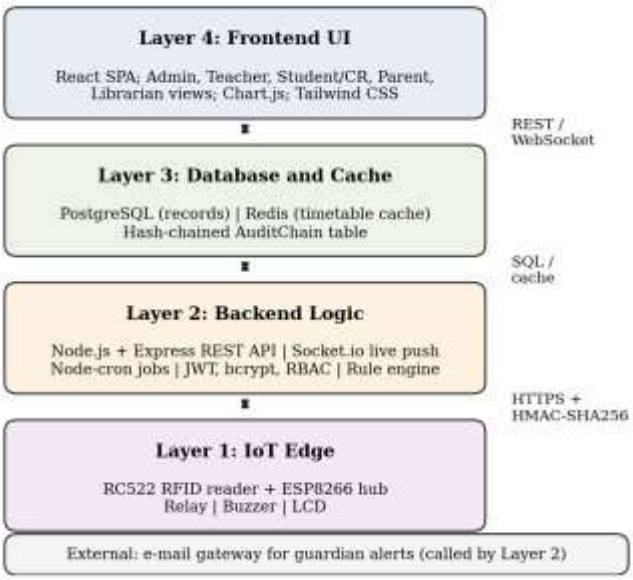


Fig -1: Four-layer CampusCore architecture

Layer 1, IoT edge. An ESP8266 hub reads the RC522 reader, signs the scan payload, and drives the relay, buzzer and LCD. **Layer 2, backend.** A Node.js and Express service validates requests, applies the rule engine, serves REST and Socket.io, and runs Node-cron jobs for period closing and alert evaluation. **Layer 3, data.** PostgreSQL stores records, Redis caches the timetable, and an AuditChain table holds hash-linked entries. **Layer 4, frontend.** A React single-page application reconfigures its layout from the authenticated role.

The platform is more than an attendance recorder. Table -6 shows the six subsystems that share one cloud infrastructure and one identity model; the attendance subsystem is the principal source of events and feeds stakeholder visibility through alerts, administrative analytics through aggregation, and the audit chain through append operations.

Table -6: Subsystems and data flow

Subsystem	Data	Flow	Output
Attendance (RFID)	Scan logs	Instant sync to cloud	Live records, parent alerts
Academic performance	Marks, progress	Calculation engine	Dashboards, auto grades
Communication	Announcements	Targeted delivery	Class feeds, notifications
Library	Borrowing records	Inventory management	Catalogue, availability
Stakeholder visibility	Attendance, alerts	Secure portal	Parent dashboards, e-mails
Administration	System-wide data	Role-based dashboards	Reports, audit logs

Adjacent layers exchange data only through the interfaces in Table -7. The edge never accesses the database directly and the frontend never talks to the edge. This keeps each layer replaceable: a future biometric reader could replace the RC522 without changes to the dashboards, provided it sends the same authenticated scan request.

Table -7: Interfaces between layers and their protection

Boundary	Interface	Protection (design)
Layer 1 to 2	POST /api/attendance/scan over HTTPS	TLS 1.3; HMAC-SHA256 payload
Layer 2 to 3	Parameterised SQL; cache lookups; audit append	Input validation; AES-256 at rest
Layer 2 to 4	REST and Socket.io channel	JWT; RBAC; rate limiting
Layer 2 to e-mail	E-mail API	Server-side credentials; status recorded

A scan flows through four stages: the edge captures, the backend decides, the data tier records, and the frontend and alert layer inform. Each stage can scale independently, and alert evaluation runs only after the attendance record is safely written, so a messaging failure can never block or undo attendance.

5. ATTENDANCE WORKFLOW AND RULE ENGINE

5.1 Edge processing

The edge does only what must be local: extract the card UID, convert it to a canonical upper-case hexadecimal form, discard invalid or partial reads, debounce repeated reads of one tap, build and sign the payload, and actuate the relay, buzzer and LCD. Business decisions, including the grace-period rule, are deliberately not made on the device, so the door action always follows backend authorisation. The firmware is a non-blocking loop that alternates between polling the reader and servicing the network, and each unit holds its own key so that a compromised unit can be revoked alone.

Table -8 lists the edge components and Table -9 the firmware state machine. The behaviour during a network fault, for example whether a cached allow list is used, is an institutional policy decision left configurable.

Table -8: Edge layer components

Component	Role	Function
RC522 reader	Input	Reads 13.56 MHz cards and extracts UIDs
ESP8266	Hub	Wi-Fi; prepares, signs and sends payload; drives outputs
Relay module	Action	Controls door access on backend authorisation
Buzzer	Audio feedback	Allowed or denied tones
LCD	Visual feedback	Student name, status, timestamp

Table -9: Edge firmware states (design)

State	Entry condition	Action and next state
Idle	Power-up or after response	Poll reader; on card go to Read
Read	Card in field	Extract UID, discard invalid reads; go to Send
Send	Valid UID	Build payload, HMAC, POST; go to Await
Await	Request sent	Wait within budget; go to Actuate or Fault
Actuate	Decision received	Drive relay, buzzer, LCD; go to Idle
Fault	Timeout or network error	Apply local policy; fault tone; go to Idle

5.2 Signed scan payload

The reader sends the payload to POST /api/attendance/scan over HTTPS. An HMAC over the UID alone would not stop replay, so the signed content also binds the room, time and a per-message value (Table -10). The backend rejects a request with an invalid signature, a timestamp outside the allowed window, or a nonce it has already seen.

Table -10: Proposed scan payload fields

Field	Content	Security role
uid	Canonical card identifier	Identity claim
reader_id	Classroom reader identifier	Binds scan to room and key
timestamp	Time at the reader	Rejects stale messages
nonce	Per-message value	Rejects repeated messages
signature	HMAC-SHA256 over the fields	Authenticates sender

5.3 Processing steps

The backend executes the following steps for each scan. (1) Verify signature, timestamp window and nonce. (2) Look up the student by UID and check enrolment; if unknown or inactive, answer DENY. (3) Read the timetable for the reader's room from Redis, falling back to PostgreSQL on a miss. (4) If a session matches, mark Present within 10 minutes of the class start, otherwise mark Absent (late); if a record already exists for this student and period, treat the tap as a repeat. (5) Write the attendance record in one transaction and append an audit block. (6) Push a live update over Socket.io. (7) Return the decision to the reader.

Fig -2 shows the message sequence for a successful scan. Alert evaluation is deliberately placed after the record is written.

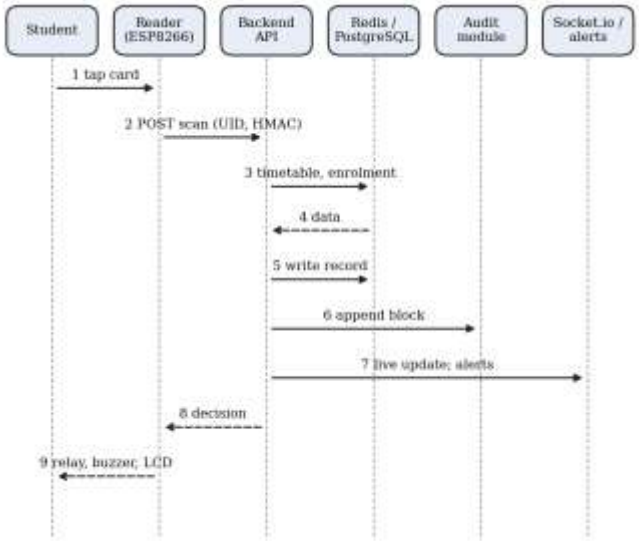


Fig -2: Sequence of a successful scan

5.4 Latency budget

The 150 ms response is allocated as shown in Fig -3: 10 ms to capture, 20 ms to prepare and send, 50 ms to arrive and validate, and 70 ms for the decision, database and audit writes, and the return signal. These are cumulative design targets. The last two intervals carry the most risk and would be examined first on a bench prototype. Whether the allocation is realistic is exactly what the prototype is intended to establish.

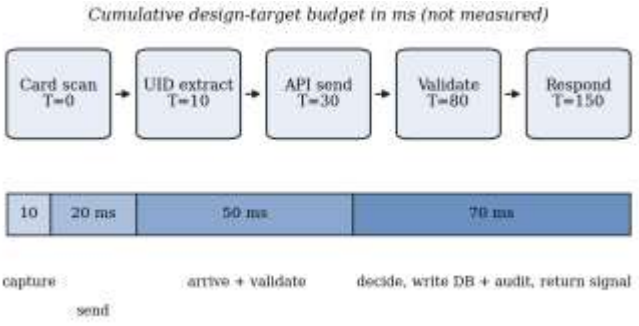


Fig -3: RFID pipeline with cumulative design-target budget

5.5 Rule engine

The rule engine (Fig -4) turns scans into decisions. A registered scan within the 10-minute grace period marks the student Present; a later scan marks Absent (late). A Node-cron job runs when each period ends and marks every student with no scan as Absent. Attendance is tracked per subject and per period rather than as a daily entry:

$$\text{Attendance \%} = (\text{periods attended} / \text{periods conducted}) \times 100 \quad (1)$$

Period-level tracking gives granular records and lets the alert logic detect a pattern in a single subject. Table -11 lists the outcomes.

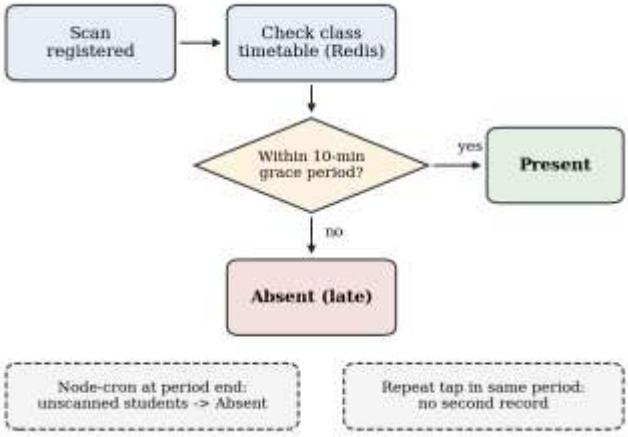


Fig -4: Attendance rule engine with grace period and end-of-period job

Table -11: Rule engine outcomes

Condition	Result	Recorded as
Scan in class, within 10 min of start	Present	Record, method RFID
Scan after the grace period	Absent (late)	Record flagged late
No scheduled class in room	Rule applied	Scan log entry
No scan by end of period	Absent	Created by period job
Repeat tap in same period	Repeat	No second record

6. ACADEMIC HUB, LIBRARY AND GUARDIAN ALERTS

Assessment module. Lecturers enter internal assessment, quiz and practical marks. A calculation engine produces averages and top scores and generates grades automatically. After a fixed duration the records are locked, and any later change needs an audited administrator action. Students and parents view results read-only.

Class feed. Lecturers and CRs publish structured posts such as assignments, announcements and polls. The server stores them and delivers real-time notifications to searchable student dashboards, replacing informal messaging groups with a record of what was communicated and when.

Digital library. The library reuses the student's RFID identity. A scan identifies the student and the book, a digital issue creates a catalogue record, due dates are tracked automatically, a reminder is sent before the deadline, and on return the system calculates any late fee.

Guardian alerts. The alert path is event-driven (Table -12). A trigger such as an absence, a drop below the configured threshold (default 75%) or repeated absences is detected; a Node-cron job acts as the logic gate against the student profile; and the system sends an e-mail alert or weekly summary to the parent. Each dispatch is stored with its delivery status and appended to the audit trail, and a failed delivery is flagged for retry. Guardians can also review full history in a token-authenticated portal. Because the logic gate is a scheduled job, the worst-case delay includes the job interval, so the interval must be short enough to meet the 60 s target for urgent triggers.

Table -12: Guardian alert triggers (channel: e-mail to parent)

Trigger	Condition	Channel
Absence	Student marked absent for a period	E-mail
Low attendance	Overall below threshold (default 75%)	E-mail
Repeated absence	Repeated or consecutive absences	E-mail
Weekly summary	End of week	E-mail

Four role-differentiated dashboards present these services (Table -13). Access control is enforced both at the API and in the frontend.

Table -13: Dashboard feature matrix

Feature	Admin	Teacher	Student	Parent
Attendance	Institution live aggregate	Class real-time	Personal history	Ward summary, alerts
Marks	Trends, export	Entry; class analytics	Read-only	Read-only
Communication	Broadcast; configure alerts	Post to class feed	Read feed (CRs post)	Alert e-mails
Configuration	Full system	Class, timetable	Profile only	Summary preferences
Audit access	Full log	Class-level logs	None	None

7. DATA MODEL, API AND ACCESS CONTROL

Schema. The core relational schema is summarised in Table -14. Types and constraints are indicative proposals. Indexes on rfid_uid and on (student_id, timestamp) keep scan-time lookups and per-student history queries bounded.

Table -14: Core database tables (indicative)

Table	Principal fields
Students	student_id (PK); rfid_uid (unique, indexed); name; course_id; parent_contact (protected); enrollment_status
Attendance	att_id (PK); student_id, class_id (FK); timestamp; method; status (present, absent, late, denied)
Classes	class_id (PK); course_name; faculty_id (FK); room; schedule; timetable_json (cached in Redis)
Notifications	notif_id (PK); recipient; trigger_event; sent_at; delivery_status (queued, delivered, failed)
AuditChain	block_id (PK); prev_hash; event_hash; timestamp; event_type; actor
Users	user_id (PK); role; email; password_hash (bcrypt); created_at; last_login
Supporting	Marks; FeedPosts; Books; LibraryTransactions

Relationships are implemented by foreign keys (Fig -5): Attendance refers to Students and Classes, Classes to a faculty user, and each AuditChain block to its predecessor through prev_hash. Marks refer to a student and a class, feed posts to a class and an author, and library transactions to a student and a book.

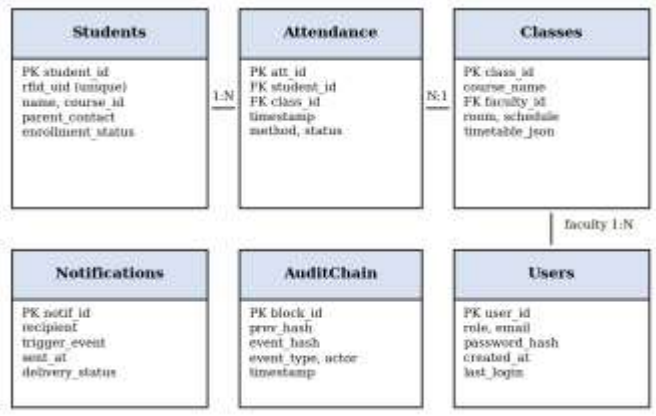


Fig -5: Entity relationship view of the core tables

API. The API follows REST conventions over HTTPS with TLS 1.3 and JSON bodies. Each request carries all information needed to process it, so any instance can serve any request, which is what allows instances to be added without session replication. Users present JWT bearer tokens and reader devices present signed payloads. Edge devices may call only the scan endpoint (Table -15).

Table -15: Core API endpoints

Endpoint	Verb	Purpose
/api/attendance/scan	POST	Process an RFID scan (reader)
/api/students/{id}/attendance	GET	Scoped attendance records
/api/notifications/dispatch	POST	Trigger alert dispatch (jobs, admin)
/api/admin/analytics	GET	System-wide analytics (admin)
/api/admin/config	PUT	Update configuration (admin)

Every request passes through the same ordered chain of checks (Table -16) before reaching business logic. Cheap checks such as rate limiting come first so that abusive traffic is rejected early.

Table -16: Ordered request-processing chain

Order	Stage	Purpose
1	TLS 1.3 termination	Encrypt traffic in transit
2	Rate limiting	Reject flooding clients early
3	Authentication (JWT or device HMAC)	Establish who or what is calling
4	Authorisation (role and scope)	Confirm the caller may use the route
5	Input validation	Reject malformed or hostile input
6	Business handler	Run operation with parameterised queries
7	Audit and response	Append audit entry; return result

Access control. Users authenticate against bcrypt hashes, administrators complete a second factor, and the backend issues a JWT whose claims include the role. The role is never taken from the client. Enforcement happens at the API and again in the frontend, which reconfigures its layout from the role. Table -17 gives the access matrix over six modules.

Table -17: Role-based access matrix (F = full, R = read-only, – = none)

Role	Config	Att. logs	Assess.	Feed	Library	Reports
Admin	F	F	F	F	F	F
Lecturer	–	R	F	F	R	–
Student	–	R	R	R	R	–
Class Rep	–	R	R	F	R	–
Librarian	–	–	–	–	F	–
Parent	–	R	R	–	–	R

8. SECURITY AND AUDIT DESIGN

Security is treated as a property of every layer. Table -18 maps each considered threat to the control specified in the design.

Table -18: Threats and design controls

Threat	Control
Interception in transit	TLS 1.3 on all network communication
Replay of scan payloads	HMAC-SHA256 over uid, reader, time, nonce
Disclosure of stored data	AES-256 encryption at rest
Password database theft	bcrypt salted hashes; no plaintext
Unauthorised dashboard access	JWT with RBAC; MFA for administrators
SQL injection and XSS	Validation; parameterised queries
Denial of service	API rate limiting
Record tampering	Append-only hash-chained audit log

Audit chain. Each audit block stores the hash of the block before it (Fig -6), so changing any earlier block invalidates every later one. Attendance records, administrative actions,

assessment-lock overrides and alert dispatches append blocks. Verification walks the chain from the first block, recomputes each hash and compares it with the stored value; a mismatch identifies the first altered block. The first implementation step is a hash-chained AuditChain table in PostgreSQL. Hyperledger Fabric [15], [16] is proposed for the distributed version because a campus ledger involves known participants and does not need a public blockchain. The ledger is conceptual, and whether every attendance record reaches the chain without loss cannot be assessed until it is built and tested. A chain also proves only that a record was not changed after it was written; it does not prove that the original tap was made by the card owner.

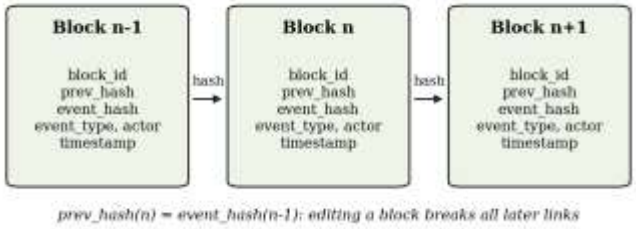


Fig -6: Structure of the audit chain

9. SCALABILITY, FAULT TOLERANCE AND DEPLOYMENT

Scalability rests on five choices: stateless API services, container orchestration that adds capacity at peaks such as class change-over, Redis caching of timetables, indexed lookups, and event-driven alerting that keeps messaging off the scan path. The services are packaged as Docker containers and orchestrated with Kubernetes on AWS or GCP, with a GitHub Actions pipeline to build, test and deploy images.

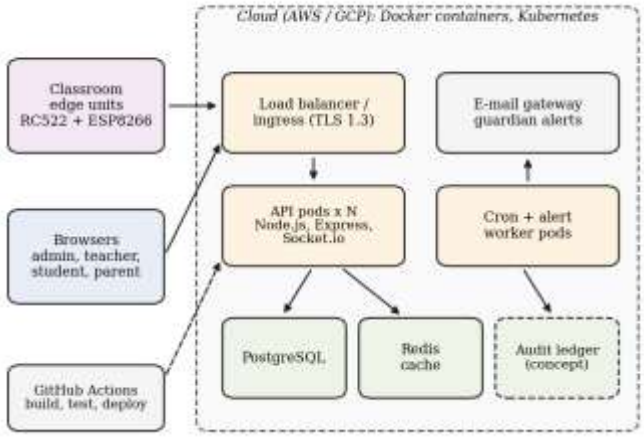


Fig -7: Proposed deployment architecture

Fig -7 shows the proposed deployment: edge units and browsers reach the API pods through a TLS-terminating ingress; worker pods run period-closing and alert jobs; and the data tier sits behind the API. Table -19 gives the rationale for the principal technology choices.

Table -19: Technology selection and rationale

Area	Choice	Rationale
Edge	ESP8266, RC522, relay, buzzer, LCD	Low cost; integrated Wi-Fi; common parts
Backend	Node.js, Express, Socket.io, Node-cron	Non-blocking I/O suits scan bursts; live push; scheduled jobs
Security	JWT, bcrypt	Stateless authentication; safe password storage
Data	PostgreSQL, Redis	ACID storage; fast timetable access
Audit	Hyperledger Fabric	Permissioned ledger for known participants
Frontend	React, Chart.js, Tailwind CSS	Fast SPA; responsive design
Operations	Docker, Kubernetes, GitHub Actions	Portability, scaling, automated delivery

Two theoretical checks give a sense of scale, with the caveat that their inputs are assumptions. By Little's law [17], the mean number of requests in the system equals arrival rate times time in system; at 500 scans per second and an assumed 120 ms server-side time this is 500 × 0.12 = 60 requests in flight, a modest load for an event-loop server whose requests mostly wait on I/O. For storage, 5,000 students taking six sessions a day over 200 teaching days add about 6,000,000 attendance rows a year, which indexed PostgreSQL tables should handle but which a load test must confirm.

Table -20 lists the failure scenarios considered and the designed response to each.

Table -20: Failure scenarios and designed responses

Failure	Designed response
Network loss at reader	Local policy; scans retried on recovery
API instance failure	Orchestrator restart; load balancer reroutes
Redis unavailable	Fall back to PostgreSQL; rebuild cache
E-mail gateway failure	Record failed status; retry; weekly summary
Scheduled job missed	Re-run on restart; period close idempotent
Audit write failure	Queue and retry; no record silently dropped

10. IMPLEMENTATION STATUS AND EVALUATION PLAN

Table -21 states exactly what exists today. The design, schema, specifications and wireframes are complete; everything else is proposed.

Table -21: Implementation status by component

Component	Technology	Status
Edge unit	ESP8266, RC522, relay, buzzer, LCD	Designed; not procured
Backend	Node.js, Express, JWT, bcrypt	Specified; not built
Real-time, jobs	Socket.io, Node-cron	Designed; not built
Database, cache	PostgreSQL, Redis	Schema designed
Frontend	React, Chart.js, Tailwind CSS	Wireframes only
Alerts	E-mail gateway	Architecture only
Audit chain	Hash chain; Hyperledger Fabric	Conceptual
DevOps	Docker, Kubernetes, GitHub Actions	Planned

Test plan. Testing is layered: unit tests protect individual rules, integration tests exercise layer boundaries, system tests walk through user scenarios, and non-functional tests cover performance, load, scalability and reliability. Test cases come from equivalence partitioning, boundary-value analysis (for example, scans just inside and outside the 10-minute grace period, or attendance just above and below 75%), decision tables (role by module) and scenarios. Simulated readers let most testing begin before hardware exists; latency needs a bench prototype. The plan contains 89 proposed cases: 8 unit, 6 integration, 6 system, 7 API, 6 database, 6 authentication, 8

authorisation, 8 security, 7 RFID workflow, 6 notification, 4 interface, 4 performance, 4 load, 4 scalability and 5 reliability cases. None has been executed. Table -23 lists the evidence that would confirm each principal target.

Table -22: Selected proposed non-functional test cases

ID	Test	Expected result
PERF-01	End-to-end capture latency on bench prototype	Distribution of times; target below 150 ms
PERF-02	Per-stage timestamps	Each stage compared with its allocation
PERF-03	Alert latency, trigger to gateway acceptance	Target below 60 s
LOAD-01	Ramp to 500 scan events per second	Error rate and latency recorded at each step
LOAD-02	Class change-over burst across many rooms	No lost records; latency recorded
SCAL-01	Increase API instances under load	Throughput rises with instance count
SCAL-03	Simulate 100,000 concurrent users	Behaviour recorded; target not assumed met
REL-02	Stop Redis	Fallback to database; correct decisions
REL-04	Restart scheduler during period closing	No duplicate or missing absences
REL-05	Compare attendance records with audit blocks	Every record has a block

For the latency tests we propose reporting the full distribution of capture times, for example the median and a high percentile, rather than a single average, because a door that is usually fast but occasionally slow still frustrates users. Per-stage timestamps (PERF-02) would show which interval of the budget is exceeded.

Table -23: Targets and the evidence that would confirm them

Target	Evidence required
Response below 150 ms	Bench-prototype timings (PERF, RF cases)
Alert below 60 s	Gateway acceptance times (PERF, NT cases)
100,000 simulated users	Load simulation (SCAL case)
All records in audit chain	Record-to-block comparison (REL, IT cases)
No cross-role exposure	Role-by-module tests (RBAC, UI cases)

11. DISCUSSION AND LIMITATIONS

Table -24 records the principal project risks identified during planning and the mitigation planned for each.

Table -24: Risk register (L = likelihood, I = impact)

ID	Risk	L	I	Mitigation
R1	150 ms target missed on real hardware	M	H	Early bench prototype; Redis cache; small payloads
R2	Card lending undermines integrity	M	M	Log analysis; QR and biometric options later
R3	Intermittent connectivity	M	H	Configured local policy at edge; retry
R4	E-mail delivery failures	M	M	Status tracking; retry; weekly summary
R5	Ledger integration harder than expected	M	M	Start with hash-chained table; migrate later
R6	API or edge security weakness	L	H	TLS, HMAC, bcrypt, rate limits; security tests
R7	Privacy concerns over student and parent data	M	H	RBAC; encryption; data minimisation

The evidence available supports one conclusion: the design is complete, internally consistent, and built from components whose roles are well established. It does not support performance claims. There is no hardware, so the 150 ms figure is unverified; there is no backend implementation, so throughput is theoretical; no load simulation has run; the blockchain is conceptual; and there is no user study, so usability and acceptance are unknown.

Three design limitations deserve emphasis. Card lending remains possible with RFID alone; QR or biometric checks are future options and bring cost and privacy trade-offs. E-mail delivery to an inbox depends on external systems the platform cannot guarantee, which is why a weekly summary provides a second route. Finally, a cloud design assumes dependable connectivity at the campus, which the reader-side retry policy only partly mitigates. We regard these as the first things to examine in a pilot.

The platform also holds personal data about students and parents, including parent contact details, marks and attendance history. The design limits exposure through role-based access, encryption at rest and token-authenticated portals, but an institution adopting it would still need its own data-protection policy covering retention, consent and access review. The literature results discussed in Section 2 come from other systems and settings and indicate direction only; they should not be read as predictions for CampusCore.

12. CONCLUSION AND FUTURE WORK

CampusCore specifies an integrated, four-layer platform that links RFID attendance, a timetable-aware rule engine, live role-based dashboards, an academic hub, digital library services, event-driven guardian alerts and a tamper-evident audit design, together with requirements, schema, API, security controls and an 89-case test plan. Its main value is a single, traceable blueprint sized for 500 to 5,000 students.

Future work proceeds in stages: build a bench prototype to measure capture latency; implement the backend and dashboards and run the planned tests; then add the Hyperledger Fabric ledger. Longer-term extensions are a QR-based backup, a mobile application with push notifications, AI-based analytics for performance prediction and dropout-risk analysis (for which classification methods such as the fusion approach in [18] and sampling-based learning on large data [19] may be relevant), biometric or facial recognition, and multi-institution deployment.

ACKNOWLEDGEMENT

The authors express their sincere gratitude to their guide, Dr. Yathish Aradya B C, Associate Professor, Department of Computer Science and Engineering, Kalpataru Institute of Technology, Tiptur, for his valuable guidance and constant support, and to the Department of Computer Science and Engineering, Kalpataru Institute of Technology, Tiptur, for providing the necessary facilities and support throughout this work.

REFERENCES

[1] K. Ashton, “That ‘Internet of Things’ thing,” RFID Journal, vol. 22, no. 7, pp. 97–114, 2009.

[2] K. Finkenzeller, RFID Handbook: Fundamentals and Applications in Contactless Smart Cards, Radio Frequency Identification and Near-Field Communication, 3rd ed. Chichester, U.K.: Wiley, 2010.

[3] NXP Semiconductors, “MFRC522: Standard performance MIFARE and NTAG frontend,” Data sheet.

[4] A. A. Mohammed and U. Jyothi Kameswari, “Web server based student attendance system using RFID technology,” Int. J. Eng. Trends Technol., vol. 4, no. 5, May 2013.

[5] Y. Wang and J. Pang, “College student activity attendance management system design based on Internet of Things and deep learning technology,” in Proc. SPIE 12989, 3rd Int. Conf. Intelligent Traffic Systems and Smart City (ITSSC 2023), Art. 1298914, 2024, doi: 10.1117/12.3023911.

[6] K. Ainebyona, A. M. Oguti, J. Walusimbi, and R. Kobusingye, “Integrating attendance tracking and emotion detection for enhanced student engagement in smart classrooms,” arXiv:2601.08049, 2026.

[7] Government of India, Ministry of Education, "National Education Policy 2020," New Delhi, India, 2020.

[8] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, Univ. California, Irvine, CA, USA, 2000.

[9] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," IEEE Computer, vol. 29, no. 2, pp. 38–47, 1996.

[10] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," IETF RFC 7519, 2015.

[11] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-hashing for message authentication," IETF RFC 2104, 1997.

[12] E. Rescorla, "The Transport Layer Security (TLS) protocol version 1.3," IETF RFC 8446, 2018.

[13] N. Provos and D. Mazières, "A future-adaptable password scheme," in Proc. USENIX Annu. Tech. Conf., 1999.

[14] S. Haber and W. S. Stornetta, "How to time-stamp a digital document," J. Cryptology, vol. 3, no. 2, pp. 99–111, 1991.

[15] E. Androulaki et al., "Hyperledger Fabric: A distributed operating system for permissioned blockchains," in Proc. 13th EuroSys Conf., Art. 30, 2018.

[16] Hyperledger Foundation, "Hyperledger Fabric documentation," Linux Foundation, 2023. [Online]. Available: https://hyperledger-fabric.readthedocs.io

[17] J. D. C. Little, "A proof for the queuing formula L = λW," Operations Research, vol. 9, no. 3, pp. 383–387, 1961.

[18] Y. Aradhya B. C., "A fusion classification prototypical for eye state recognition in stroke patients using electroencephalogram (EEG) data," Int. J. Intell. Syst. Appl. Eng. (IJISAE). [Online]. Available: https://ijisae.org/index.php/IJISAE/article/view/3080

[19] Y. Aradhya B. C. and Y. P. Gowramma, "Progressive sampling algorithm with Rademacher averages for optimized learning of big data: A novel approach," Int. J. Comput. Appl., vol. 175, no. 15, pp. 37–40, Aug. 2020, doi: 10.5120/ijca2020920652.