International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper

CampusConnect: A Scalable Role-Based Campus ERP System Using the MERN Stack (Case Study at MGIT)

Ms. M. Samyuktha
Asst. Professor
Department of Computer Science and Engineering
Mahatma Gandhi Institute of Technology
Hyderabad 500075, India
Msamyuktha_cse@mgit.ac.in

Ms. K. Shirisha
Asst. Professor
Department of Computer Science and Engineering Mahatma Gandhi Institute of Technology Hyderabad 500075, India
Kshirisha_cse@mgit.ac.in

Kottada Sri Venkateshwara Dheeraj Kumar
Department of Computer Science and Engineering Mahatma Gandhi Institute of Technology Hyderabad 500075, India
Ksrivenkateshwaradheerajkumar_cse2405v8@mgit.ac.i n

Chittineni Jishnu Chowdary
Department of Computer Science and Engineering Mahatma Gandhi Institute of Technology Hyderabad 500075, India
Cjishnuchowdary_cse2405t9@mgit.ac.in

***Abstract*—*This paper presents MGIT CampusConnect, a full-stack, role-based campus ERP system built on the MERN stack and deployed on cloud PaaS infrastructure, with two algorithmic contributions. First, a dual-constraint timetable conflict detection algorithm enforces mutual exclusion at both the section and faculty levels through two sequential O(log n) indexed queries; a concurrent insertion experiment confirmed that storage-layer enforcement via a compound unique index on {facultyId, day, timeSlot} is a correctness requirement. Second, an asynchronous, threshold-triggered attendance notification algorithm classifies students into Safe (P≥75%), Warning (60%≤P<75%), and Critical (P<60%) risk tiers and dispatches notifications to the at-risk student and their faculty mentor after every marking session. The system also provides credit-weighted CGPA computation, structured mentoring, and a tamper-evident audit trail. A pilot at MGIT (24 students, 12 faculty, 5 subjects) validated 47 API endpoints with response times below 500 ms for single-document operations and 634 ms for aggregation pipelines.*

Index Terms—attendance management, audit logging, campus ERP, conflict detection, Express.js, JWT authentication, marks management, MERN stack, mentor module, MongoDB, Node.js, React, role-based access control, web application.

I. INTRODUCTION

EDUCATIONAL institutions manage interconnected academic and administrative functions—admissions, timetabling, attendance, marks, mentoring, and performance monitoring. Many still rely on fragmented tools or paper-based registers, introducing data inconsistency and administrative overhead [1], [2]. Existing systems fail to address three simultaneous challenges: (1) preventing timetable conflicts at both section and faculty levels under concurrent access; (2) automatically dispatching risk-tier notifications after each attendance session; and (3) maintaining a tamper-evident audit trail across all state-modifying operations. These gaps motivate this work.

Scalable cloud platforms and mature JavaScript frameworks have made role-aware ERP development economically viable. The MERN stack (MongoDB, Express.js, React, Node.js) offers a unified development environment and a document model that accommodates nested academic hierarchies (branch → semester → section → subject) more naturally than relational schemas [3]. Deployment on PaaS providers such as Vercel and Render further reduces infrastructure cost, making production-grade systems accessible to resource-constrained institutions.

The remainder of this paper is organized as follows: Section II surveys related work; Sections III–IX detail the architecture, modules, and algorithms; Section X presents evaluation; Section XI compares with existing systems; Section XII concludes.

II. RELATED WORK

Campus ERP systems have attracted substantial research attention. Deshmukh et al. [4] developed a full-stack College Management System using React.js, Node.js, Express.js, and PostgreSQL/MongoDB that reduces paperwork, improves accessibility, and supports role-based access control; however, the system does not address timetable conflict detection, automated notification dispatch, or audit logging. Shaikh et al. [5] developed an ERP system for college management with REST Web APIs, handling student and faculty data, attendance, and marks; while it provides a structured API-based architecture, it does not implement timetable conflict prevention, automated notification dispatch, or audit logging.

Patil et al. [6] developed an online College ERP system using the MERN stack (MongoDB, Express.js, React, Node.js) for monitoring students and faculty activities with an internet-based information management interface; however, the system provides basic role access without compound conflict detection, has no automated notification dispatch, and includes no mentor module or audit trail. Kumar et al. [7] integrated ERP with the C4.5 decision-tree algorithm for student performance analysis and placement prediction, enabling report generation and predictive analytics, but the system depends on a traditional client-server architecture requiring ongoing database

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper

maintenance and accurate data collection. Pawar et al. [8] presented a centralized web-based ERP with modules for Admin, Student, Staff, Parent, and Transport, reducing manpower and minimizing redundancy; however, it was designed to replace paper-based systems and does not address concurrent-access conflicts or automated notification dispatch.

These works collectively validate the value of centralized digital campus management while identifying persistent gaps: absent real-time conflict prevention, automated notifications, structured mentoring, integrated analytics, and audit trails. MGIT CampusConnect addresses each gap within a single cloud-deployable MERN-stack application.

TABLE I

Literature Survey of Related Campus Management Systems

Sl. No.	YOP	Title	Authors	Demerits
1	2025	College Management System	Deshmukh Sangram Pandit, Chavan Pruthviraj Subhash, Autkar Chaitanya Prakash, Prof. Jondhale D.R	Single-table uniqueness check only; no compound multi-constraint conflict prevention; no event-triggered notification dispatch; no audit trail
2	2021	College ERP Using MERN Stack	Shubham Patil, Saurav Daware, Ameya Bhagat, and Prof. Jayant Sawarkar	No timetable conflict detection; no automated threshold-triggered notification dispatch; no mentor module or audit trail; basic role access without compound multi-constraint enforcement
3	2021	ERP Software for College Management System with REST Web APIs	Danish Shaikh, Smita Chavan, Supriya Deshmukh, Shruti Karad	No timetable conflict detection or faculty double-booking prevention; no automated attendance notification dispatch; no mentor module or audit trail; REST API only without real-time event triggers
4	2021	ERP for College Management System	Sudarshan Kumar, Sudhanshu Mishra, Ankit Kumar	Rigid relational schema constrains dynamic academic hierarchies; no cloud deployment; no conflict prevention or automated notification
5	2015	College ERP System	Sagar Pawar, Gaurav Geet, Pavan Sonawane, Chetan B. Barhate	Analysis paper only; proposes no architecture; does not address concurrent-access or data-integrity problems identified across surveyed deployments

I. SYSTEM OVERVIEW AND CONTRIBUTIONS

MGIT CampusConnect is a full-stack, role-based Campus Enterprise Resource Planning (ERP) web application developed using the MERN stack and deployed on cloud PaaS infrastructure. The system serves three user roles—Administrator, Faculty, and Student—each with a dedicated dashboard and a precisely scoped set of functional capabilities enforced by JWT-based role-based access control.

The system comprises thirteen tightly integrated modules organized across four functional domains: (1) Academic Structure Management—branches, semesters, sections, subjects, and faculty assignment; (2) Academic Operations—attendance tracking, marks management, timetable scheduling, and mentoring; (3) Communication and Monitoring—notifications, analytics dashboards, and system settings; (4) Security and Compliance—JWT authentication, role-based authorization,

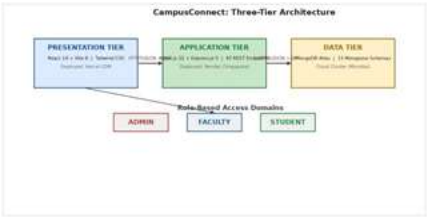


Fig. 1. Three-tier system architecture of MGIT CampusConnect showing the React/Vite frontend (Vercel), Express.js REST API (Render), and MongoDB Atlas cloud database, connected over HTTPS with JWT-based authentication on every protected route.

A. Presentation Tier

The frontend is a React 19 SPA bundled by Vite 8 and deployed on Vercel. Three protected route trees (/admin/*, /faculty/*, /student/*) are guarded by a PrivateRoute that verifies the JWT role before rendering. An Axios interceptor attaches the

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper

Bearer token to outgoing requests and redirects to login on HTTP 401. Chart.js provides analytics charts; Tailwind CSS provides responsive styling.

A. Application Tier

The backend is a Node.js 22 + Express.js 5 REST API deployed on Render, exposing 47 endpoints across nine route groups. Every protected route passes through authMiddleware (HS256 JWT verification) and authorizeRoles (HTTP 403 on role mismatch). Express 5's native async error propagation eliminates try-catch boilerplate.

B. Data Tier

The data tier is MongoDB Atlas accessed via Mongoose ODM v8. Fifteen schemas model the domain (User, Student, Faculty, Branch, Semester, Section, Subject, Attendance, AttendanceAudit, Marks, Timetable, Notification, MentorNote, AuditLog, SystemSetting). Referential integrity is enforced via populate() calls; compound unique indexes on Timetable enforce conflict prevention at the storage layer independently of application-layer checks.

C. Role-Based Access Model

Three roles are defined: ADMIN, FACULTY, and STUDENT. ADMIN has unrestricted write access to all modules. FACULTY has write access to Attendance and Marks for their assigned subjects and read access to their own analytics. STUDENT has read-only access to their own data. Shared capabilities (login, profile, password change, OTP reset) are available to all. The role is embedded in the JWT payload and is authoritative for all access-control decisions.

IV. SECURITY: JWT AND ROLE-BASED ACCESS CONTROL

Security in **MGIT CampusConnect** is implemented across three layers: password storage, session management, and route-level authorization.

A. Password Storage

Passwords are hashed using bcryptjs with a salt-round count of 12, producing a 60-character bcrypt hash. The work factor of 12 requires approximately 250 ms per hash on commodity hardware, making brute-force attacks computationally prohibitive. No reversibly-encrypted value is stored in the database or logs.

B. JWT Session Management

Upon successful login, the server issues an HS256-signed JWT (256-bit secret) encoding the user's ObjectId and role with a seven-day expiry. The token is stored in localStorage and attached via an Axios interceptor; authMiddleware calls jwt.verify() and returns HTTP 401 on failure. The stateless design is horizontally scalable. JWT storage in localStorage is XSS-susceptible; migration to HttpOnly, SameSite=Strict cookies is planned, mitigated currently via Content Security Policy headers.

C. OTP-Based Password Reset

Password reset uses a time-limited OTP workflow. On request, the server generates a cryptographically random 6-digit OTP, stores its bcrypt hash alongside a 10-minute expiry on the User document, and dispatches the plaintext OTP to the registered email via Nodemailer over Gmail SMTP (MAIL_USER, MAIL_PASS env vars). The OTP is invalidated immediately on use or expiry, ensuring the stored hash provides no useful plaintext if the database is compromised.*Route-Level Authorization*

Beyond authentication, every endpoint specifies an allowedRoles array passed to authorizeRoles middleware. POST /api/timetable is restricted to ADMIN; POST /api/attendance to FACULTY and ADMIN; GET /api/marks returns only the requesting student's own records when called by STUDENT. This declarative, fine-grained control ensures no role can access data outside its domain even if a JWT is stolen and replayed.

V. TIMETABLE SCHEDULING AND CONFLICT DETECTION

The timetable module allows administrators to define a weekly schedule for each section. A timetable entry is a tuple (branchId, semesterId, sectionId, subjectId, facultyId, day, timeSlot, room). Valid days are Monday through Saturday; valid time slots are on-the-hour intervals from 09:00 to 16:00. Before persisting any new entry, the API executes the dual-constraint conflict detection algorithm described in Algorithm 1.

Algorithm 1: Dual-Constraint Timetable Conflict Detection

Input: entry = (branchId, semesterId, sectionId, subjectId, facultyId, day, timeSlot, room)
Output: HTTP 201 Created | HTTP 400 Conflict | HTTP 409 Duplicate Key
1. $Q_1 \leftarrow$ Timetable.findOne({ sectionId, day, timeSlot }) // O(log n) lookup on compound B-tree index {sectionId, day, timeSlot}
2. IF $Q_1 \neq$ null THEN
3. RETURN HTTP 400: "Section already has a class at this time."
4. END IF
5. $Q_2 \leftarrow$ Timetable.findOne({ facultyId, day, timeSlot }) // O(log n) lookup on compound B-tree index {facultyId, day, timeSlot}
6. IF $Q_2 \neq$ null THEN
7. RETURN HTTP 400: "Faculty already has a class at this time."
8. END IF
9. TRY 10. Timetable.create(entry) 11. RETURN HTTP 201: entry 12. CATCH DuplicateKeyError (MongoDB error code 11000) 13. RETURN HTTP 409: "Conflict detected at storage layer (concurrent request)." 14. END TRY Storage-Layer Invariants: {sectionId, day, timeSlot} — compound unique index (section-level mutual exclusion) {facultyId, day, timeSlot} — compound unique index (faculty-level mutual exclusion)

Both Q_1 and Q_2 are O(log n) indexed lookups on compound B-tree indexes, where n is the Timetable collection cardinality. Application-layer checks alone are insufficient under concurrent access: a controlled experiment demonstrated that two simultaneous POST requests with the same (facultyId, day, timeSlot) tuple both returned HTTP 201 when only the section-level index was present, confirming the race condition at lines 5–7 of Algorithm 1. With both unique indexes in place, the same experiment yields one HTTP 201 and one HTTP 409, confirming storage-layer serialization. Mean insertion latency was 394 ms (100 trials), of which 280–310 ms is cross-region network overhead, confirming negligible algorithmic cost.

The resulting timetable is displayed to administrators and

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper

faculty as a color-coded weekly grid organized by day (rows) and time slot (columns), with each cell showing the subject code, faculty name, and room number. Students see a read-only version of their section's timetable on their dashboard. Fig. 2 illustrates the timetable interface with a conflict rejection scenario.

Fig. 2. Timetable management interface showing the weekly grid and an inline HTTP 400 conflict-rejection message when a faculty member is already assigned to another section at the same time slot.

Fig. 5. Timetable management interface showing weekly schedule and real-time conflict detection for section and faculty allocation.

VI. ATTENDANCE MANAGEMENT AND NOTIFICATION DISPATCH

The attendance module handles the complete lifecycle of lecture-level attendance records for every subject-section pair. The workflow, illustrated in Fig. 3, proceeds in five stages.

A. Class Selection and Student Loading

The faculty member selects a subject, section, date, and time slot through a four-step stepper interface. The system fetches the current student roster for the selected (sectionId) and presents each student with a toggle button defaulted to present. The faculty reviews and adjusts the status for absent students, then submits the record.

B. Duplicate Prevention

Before persisting the attendance record, the server queries the AttendanceAudit collection for a document matching (subjectId, sectionId, date, timeSlot). If found, the request is rejected with HTTP 409 Conflict to prevent double-marking the same lecture. This guard is enforced regardless of whether the same faculty member or a different one makes the second request, closing a potential coordination gap in multi-faculty sections.

C. Persistence and Audit

Upon passing the duplicate check, the system creates an Attendance document recording the presentStudents array (list of studentIds) and absentStudents array, along with subjectId, sectionId, facultyId, date, timeSlot, presentCount, and absentCount. An AttendanceAudit document is simultaneously created as an immutable record of the marking event. An AuditLog entry is also generated for the administrative trail.

D. Notification Dispatch

After persistence, an asynchronous step computes cumulative attendance P = (present / total) × 100 for each student-subject pair. Students are classified as Safe (P≥75%), Warning (60%≤P<75%), or Critical (P<60%); Warning and Critical students receive a Notification document with subject name,current percentage, risk level, and recommended action. Safe-tier students receive no notification, reducing notification fatigue. The formal algorithm is Algorithm 2.

Algorithm 2: Threshold-Triggered Attendance Notification Dispatch

Input: attendanceRecord = { subjectId, sectionId, presentStudents[], absentStudents[], date, timeSlot } subjectName (resolved from subjectId) Output: Notification documents created for at-risk students and mentors 1. FOR EACH studentId IN (presentStudents ∪ absentStudents) DO 2. allRecords ← Attendance.find({ studentId, subjectId }) // Aggregate all prior sessions for this student-subject pair 3. totalPresent ← COUNT of sessions where studentId ∈ presentStudents 4. totalConducted ← COUNT(allRecords) 5. P ← (totalPresent / totalConducted) × 100 6. IF P ≥ 75 THEN riskTier ← "Safe" // No notification dispatched 7. ELSE IF P ≥ 60 THEN riskTier ← "Warning" 8. ELSE riskTier ← "Critical" 9. END IF 10. IF riskTier ≠ "Safe" THEN 11. // Notify student 12. createNotification({ userId: student.userId, role: "STUDENT", subject: subjectName, percentage: P, riskLevel: riskTier }) 13. // Notify mentor (if assigned) 14. student ← Student.findById(studentId).populate("mentorId") 15. IF student.mentorId ≠ null AND student.mentorId.userId ≠ null THEN 16. createNotification({ userId: student.mentorId.userId, role: "FACULTY", subject: subjectName, studentId: studentId, percentage: P, riskLevel: riskTier }) 17. END IF 18. END IF 19. END FOR Time complexity: O(s × d), where s = number of students in the section, d = number of prior attendance sessions for the subject.

The algorithm executes asynchronously after the attendance record and its audit entry have been persisted, ensuring that notification dispatch latency does not affect the attendance submission response time perceived by the faculty user. The dual-recipient dispatch at lines 12 and 16 ensures that both the student and their assigned mentor are informed simultaneously, removing the dependency on mentor polling of individual student records.

Fig. 3. Attendance marking and notification dispatch workflow: Faculty submits → Duplicate check → Persist Attendance + Audit → Compute P per student →

Received: 09-05-2026 | Accepted: 17-06-2026 | Published: 23-06-2026

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper

IF P < 75% THEN create Notification.

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper

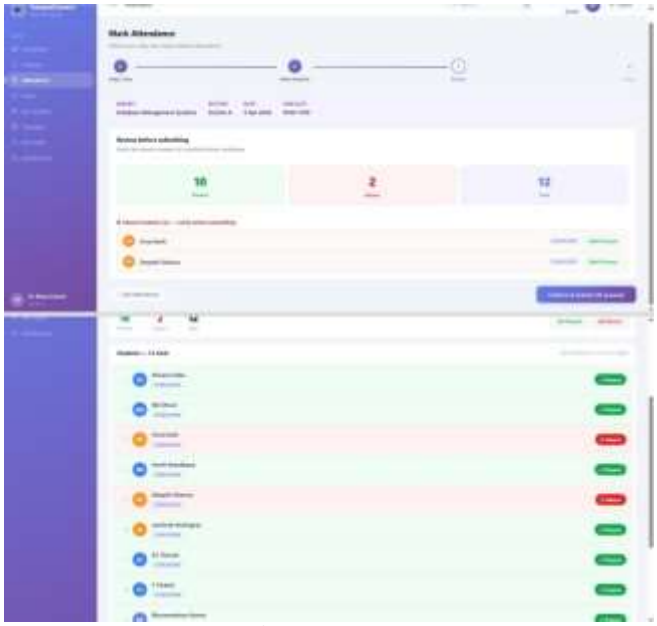


Fig. 6. Attendance marking interface with present/absent status visualization and student-wise tracking.

E. Override Capability

Administrators have the ability to override any attendance record by submitting a corrected attendee list along with a mandatory reason string. The original record is updated, and an AuditLog entry is created with actionType=ATTENDANCE_OVERRIDDEN, the delta (students added/removed from present list), the reason provided, and the administrator's userId. This preserves accountability while allowing legitimate corrections.

VII. MARKS MANAGEMENT AND CGPA COMPUTATION

The marks module supports four-component evaluation per student per subject: internalMarks (max 30), assignmentMarks (max 10), midExamMarks (max 40), and practicalMarks (max 20, applicable where a practical component exists). The total is computed as the sum of all applicable components, out of 100.

A. Grade Computation

Grades are assigned per MGIT's official ten-point scale. Table I defines the mapping from total marks to letter grade and grade point.

student within the requested semester, joining with the Subject collection for credit values, and returns a per-subject breakdown in the analytics response.

C. Bulk Upload

Individual marks entry is supplemented by bulk CSV/Excel upload. The server parses the file via multer, validates each row against component limits, and upserts Marks documents in a batched operation; validation errors are reported per-row without aborting the batch. Mark overrides generate an AuditLog entry with previous and new values and a mandatory reason string.

VIII. MENTOR MODULE

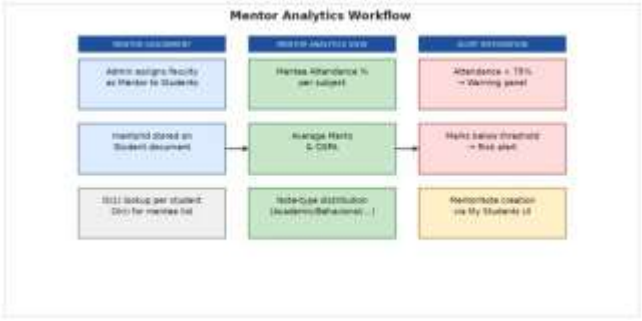
The mentor module implements a structured academic mentoring system in which each student group within a section is assigned a dedicated faculty mentor by the administrator. The module comprises two sub-components: mentor assignment and mentor note management.

A. Mentor Assignment

The administrator assigns a faculty mentor to a student group via the AssignMentor workflow, persisting a mentorId field on each Student document for O(1) mentor lookup and O(n) mentee-list retrieval. A student has exactly one mentor; reassignment is recorded in the AuditLog with the previous and new mentor identities.

B. Mentor Notes

Faculty mentors create structured MentorNote documents (fields: studentId, facultyId, note, type, createdAt) categorized as Academic, Behavioral, Attendance, Career, or Other. Notes are accessible only to the authoring mentor and administrators. The analytics view aggregates note counts by type across the mentor's mentee list (Fig. 4).



International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper

TABLE II
MGIT GRADING SCALE

Marks Range	Letter Grade	Grade Point
≥90	O (Outstanding)	10
80–89	A+ (Excellent)	9
70–79	A (Very Good)	8
60–69	B+ (Good)	7
50–59	B (Above Avg)	6
40–49	C (Average)	5
<40	F (Fail)	0

B. CGPA Computation

The Cumulative Grade Point Average is computed as the credit-weighted mean of grade points across all subjects in a semester, as given by (1):

$$CGPA = \Sigma(G_i \times C_i) / \Sigma C_i \quad (1)$$

where G□ is the grade point and C□ the credit weight of subject i. The computation aggregates all Marks documents for the

Fig. 4. Faculty mentor analytics view showing mentee list with attendance percentage, average marks, CGPA, and note-type distribution for each assigned student.

Fig. 7. Mentor analytics dashboard displaying attendance, marks, and risk-based student performance insights.

C. Integration with Attendance and Marks

The mentor module integrates with attendance and marks: when attendance falls below the Warning threshold (75%) or marks fall below a configurable risk threshold, the generated Notification document surfaces on the mentor's dashboard in a "Students Needing Attention" panel, providing proactive alerts without manual review.

IX. IMPLEMENTATION AND EVALUATION

A. Implementation Details

The system uses: React 19 + Vite 8 + Tailwind CSS (frontend); Node.js 22 + Express.js 5 + Mongoose 8 (backend); MongoDB Atlas M0 (database); Vercel and Render (hosting). The monorepo (client/ and server/) exposes 47 API endpoints; all secrets are injected via platform environment variables.

B. Dataset

The system was deployed and evaluated with a dataset comprising 24 students across four sections in the Information Technology branch, 12 faculty members, and five subjects per section for Semester 1 (Academic Year 2025–2026).

C. Functional Testing

Systematic functional testing was performed by exercising each of the 47 API endpoints with valid and boundary-case inputs, verifying HTTP status codes, response payloads, and MongoDB state changes. Table II summarizes results for the twelve most functionally critical endpoints.

TABLE III
FUNCTIONAL TEST RESULTS FOR CRITICAL API ENDPOINTS

Auth (POST login)	285	412	Includes bcrypt comparison (~250 ms)
Indexed GET (single)	312	498	Single document by ObjectId
Write (POST simple)	428	623	Single document insert
Conflict detection	394	581	Two indexed reads + one write
Analytics pipeline	634	921	MongoDB aggregation pipeline
Bulk marks (30 rows)	892	1340	CSV parse + 30 upsert operations

The timetable conflict detection endpoint (394 ms mean) confirms that the dual-query constraint check introduces negligible overhead. The analytics pipeline (634 ms) reflects the MongoDB aggregation cost but remains within acceptable interactive response bounds. The Vercel-hosted frontend achieved Lighthouse performance scores of 91 (desktop) and 78 (mobile) on a cold load.

E. Concurrency and Race-Condition Validation

A controlled concurrency experiment dispatched two simultaneous POST requests via Promise.all() with identical (facultyId, day, timeSlot) tuples. With only the section-level index, both returned HTTP 201 and two conflicting documents were confirmed, validating the race condition. With both indexes in place, one HTTP 201 and one HTTP 409 were returned, confirming correct storage-layer serialization.

X. COMPARISON WITH EXISTING SYSTEMS

Table IV presents a feature comparison of **MGIT CampusConnect** against four representative prior systems across ten capability dimensions directly relevant to institutional ERP requirements.

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper

Endpoint	Method	Function	Status	Result
/api/auth/login	POST	Credential check + JWT issuance	200/401	PASS
/api/auth/reset-otp	POST	OTP-based password reset	200/400	PASS
/api/timetable	POST	Section conflict detection	400	PASS
/api/timetable	POST	Faculty conflict detection	400	PASS
/api/attendance	POST	Duplicate lecture prevention	409	PASS
/api/attendance	POST	Low-attendance notification	Async	PASS
/api/marks/bulk	POST	CSV bulk marks upload	201	PASS
/api/marks	GET	CGPA computation accuracy	Correct	PASS
/api/analytics	GET	Aggregation pipeline	200	PASS
/api/admin/students	POST	Roll-no uniqueness check	409	PASS
/api/mentor/notes	POST	Mentor note creation	201	PASS
/api/notifications	GET	Role-filtered notification list	200	PASS

TABLE V
FEATURE COMPARISON OF CAMPUS MANAGEMENT SYSTEMS

Capability	[4]	[5]	[7]	[8]	Proposed
JWT Role Auth	Yes	No	No	No	Yes
Timetable Conflict Det.	No	No	No	No	Yes
Auto Notifications	No	No	No	No	Yes
Bulk Marks Upload	No	Yes	No	No	Yes
CGPA Computation	No	No	No	No	Yes
Mentor Module	No	No	No	No	Yes
Analytics Dashboard	No	No	No	No	Yes
Audit Trail	No	No	No	No	Yes
OTP Password Reset	No	No	No	No	Yes
Cloud Deployment	Yes	No	No	No	Yes

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper

D. Performance Evaluation

API response times were measured over 100 repeated requests per endpoint category from a client in Hyderabad, India (server region: Render Singapore; DB region: MongoDB Atlas Mumbai). Table III summarizes mean and 95th-percentile response times.

TABLE IV
API RESPONSE TIME MEASUREMENTS (N = 100 PER CATEGORY)

Endpoint Category	Mean (ms)	P95 (ms)	Notes

MGIT CampusConnect is the only system in the comparison to provide all ten capabilities. The conflict detection, automated notification, CGPA computation, mentor module, and audit trail represent advances over all four prior works. Cloud deployability on zero-cost PaaS is shared only with Deshmukh et al. [4]. The comparison is limited to features reported in the surveyed papers.

XI. CONCLUSION AND FUTURE WORK

This paper has presented MGIT CampusConnect, a role-based campus ERP system deployed at Mahatma Gandhi Institute of Technology (MGIT), Hyderabad, providing three role-specific dashboards across thirteen functional modules. Two core algorithmic contributions are embedded: (i) a dual-constraint timetable conflict detection algorithm enforcing mutual exclusion via two O(log n) indexed queries backed by storage-layer compound unique indexes; and (ii) a threshold-triggered attendance notification algorithm classifying students as Safe, Warning, or Critical and dispatching dual-recipient notifications after each marking session. Additional features include credit-weighted CGPA computation, structured faculty mentoring, and a tamper-evident audit trail. A pilot with 24 students, 12 faculty, and five subjects at MGIT validated all 47 API endpoints with response times below 500 ms for single-document operations.

Deployment on Vercel and Render with MongoDB Atlas demonstrates production-grade institutional ERP at near-zero infrastructure cost. The pilot—24 students, 12 faculty, five subjects at MGIT—confirmed correct functional behavior. The authors acknowledge the dataset size as a limitation; large-scale load testing under hundreds of concurrent users and benchmarking against alternative ERP platforms remain future work.

Future work will pursue four enhancements: (i) a React Native mobile app for native push notifications; (ii) an automated timetable generation engine using constraint-satisfaction (backtracking with arc consistency); (iii) a machine-learning module to predict student academic-risk from attendance and marks sequences; and (iv) multi-tenant architecture to serve multiple institutions under isolated data namespaces.

Three empirical findings have broader relevance: (i) application-layer uniqueness checks are insufficient under concurrent access—storage-layer enforcement is mandatory, confirmed by experiment; (ii) dual-recipient asynchronous notification eliminates monitoring gaps; and (iii) cross-region PaaS latency (280–310 ms of the 394 ms mean) dominates response times, a key trade-off for hosted versus on-premise deployment.

ACKNOWLEDGMENT

The authors thank the faculty and students of the Department of Computer Science and Engineering, Mahatma Gandhi Institute of Technology, Hyderabad, for their participation in system evaluation and feedback. The authors also acknowledge Dr. C. R. K. Reddy, Professor and Head of the Department of CSE, MGIT, for institutional support and encouragement.

REFERENCES

1. [1] V. Kumar, V. Rana, N. Tyagi, and Prof. M. Sharma, "Optimizing Academic Operations: A Comprehensive Study on the Implementation and Impact of College ERP Systems," Int. J. Innovative Res. Technol. (IJIRT), vol. 11, no. 1, pp. 141–146, Jun. 2024.
2. [2] P. Subha, I. Fefina, C. Niranjana Devi, S. Suruthika, and K. Sushmeena, "College Management System," Int. J. Adv. Res. Comput. Commun. Eng. (IJARCCE), vol. 11, no. 5, pp. 765–770, May 2022, doi: 10.17148/IJARCCE.2022.115135.
3. [3] S. A. Bafna, P. D. Dutonde, S. S. Mamidwar, M. S. Korvate, and D. Shirbhare, "Review on Study and Usage of MERN Stack for Web Development," Int. J. Res. Appl. Sci. Eng. Technol. (IJRASET), vol. 10, no. II, Feb. 2022, doi: 10.22214/IJRASET.2022.39868.
4. [4] D. S. Pandit, P. S. Chavan, C. P. Autkar, and Prof. D. R. Jondhale, "College Management System," Int. J. Sci. Res. Comput. Sci. Eng. Inf. Technol. (IJSRCSEIT), vol. 11, no. 5, pp. 344–349, Oct. 2025.
5. [5] D. Shaikh, S. Chavan, S. Deshmukh, and S. Karad, "ERP Software for College Management System with REST Web APIs," Int. J. Adv. Res. Comput. Commun. Eng. (IJARCCE), vol. 10, no. 3, pp. 230–235, Mar. 2021, doi: 10.17148/IJARCCE.2021.10341.
6. [6] S. Patil, S. Daware, A. Bhagat, and Prof. J. Sawarkar, "College ERP Using MERN Stack," Int. J. Sci. Res. Comput. Sci. Eng. Inf. Technol. (IJSRCSEIT), vol. 7, no. 3, pp. 190–196, May–Jun. 2021, doi: 10.32628/CSEIT217337.[7] S. Kumar, S. Mishra, and A. Kumar, "ERP for College Management System," Int. J. Adv. Res. Comput. Commun. Eng. (IJARCCE), vol. 10, no. 5, pp. 193–196, May 2021, doi: 10.17148/IJARCCE.2021.10536.
7. [8] S. Pawar, G. Geet, P. Sonawane, and C. B. Barhate, "College ERP System," Int. J. Res. Eng. Appl. Manage. (IJREAM), vol. 01, no. 02, May 2015.
8. [9] M. B. Jones, J. Bradley, and N. Sakimura, "JSON web token (JWT)," RFC 7519, IETF, May 2015, doi: 10.17487/RFC7519.
9. [10] MongoDB Inc., "MongoDB manual—official documentation," 2024. [Online]. Available: https://www.mongodb.com/docs/
10. [11] OpenJS Foundation, "Express.js 5.x API reference," 2024. [Online]. Available: https://expressjs.com/
11. [12] Meta Open Source, "React 19—the library for web and native user interfaces," 2024. [Online]. Available: https://react.dev/
12. [13] OpenJS Foundation, "Node.js v22 documentation," 2024. [Online]. Available: https://nodejs.org/docs/
13. [14] Render Inc., "Render platform documentation," 2025. [Online]. Available: https://render.com/docs
14. [15] M. Fowler, Patterns of Enterprise Application Architecture. Boston, MA, USA: Addison-Wesley, 2002, pp. 15–42.
15. [16] Vercel Inc., "Vercel platform documentation," 2025. [Online]. Available: https://vercel.com/docs
16. [17] SmartDraw Software LLC, "UML Diagram Tool," [Online]. Available: https://www.smartdraw.com/uml-diagram/uml-diagram-tool.html, Accessed: 2026.

Kottada Sri Venkateshwara Dheeraj Kumar is pursuing the B.Tech. degree in Computer Science and Engineering at Mahatma Gandhi Institute of Technology (MGIT), Hyderabad, India (expected 2028). He is the lead architect and primary developer of CampusConnect, responsible for system architecture, core backend modules, and cloud deployment. His interests include distributed systems, RESTful API design, cloud-native architectures, and educational technology platforms.

Chittineni Jishnu Chowdary is pursuing the B.Tech. degree in Computer Science and Engineering at Mahatma Gandhi Institute of Technology, Hyderabad, India (expected 2028). He is a co-developer of CampusConnect, contributing to frontend and

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper

backend development. His technical expertise includes web application development, database management, and RESTful API integration. His research interests include scalable system design, backend development, and cloud-based application development.