International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 04 | April - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

MULTI-COLLEGE EVENT MANAGEMENT SYSTEM: A CLOUD-BASED SOLUTION FOR EDUCATIONAL INSTITUTIONS

Saksham Ital[1], Sushant Bhalerao[2], Abhinav Raut[3], Samarth Gajghate[4], Rehan Gagate[5]
Guide: Prof. Mansi Ghode[6]

[12345] Student, Department of Computer Engineering, NIT Polytechnic, Nagpur, Maharashtra, India
[6] Assistant Professor, Department of Computer Engineering, NIT Polytechnic, Nagpur, Maharashtra, India

Abstract - Educational institutions face significant challenges in managing academic events due to fragmented processes, lack of centralized coordination, and absence of real-time communication infrastructure. This paper presents the Multi-College Event Management System (EMS), a comprehensive cloud-based platform designed to digitize and automate the complete event management lifecycle across multiple educational institutions. The system implements a five-tier role-based access control (RBAC) hierarchy comprising Developer, Principal, Head of Department (HOD), Teacher, and Student roles, each with distinct permissions and workflows. Built using Python FastAPI for the backend REST API, MongoDB Atlas for cloud-hosted NoSQL database storage, and Vanilla JavaScript with Tailwind CSS for the responsive frontend, the system provides enterprise-grade security through JWT-based authentication and Argon2 password hashing. Key features include hierarchical approval workflows, real-time WebSocket-based push notifications, multi-tenant architecture with complete data isolation per institution, student batch promotion system, and Progressive Web App (PWA) capabilities. Performance evaluation demonstrates 99.8% uptime, average API response time of 180ms, and support for concurrent access by 500+ users.

Key Words: *Event Management System, Multi-Tenancy, FastAPI, MongoDB, JWT Authentication, Role-Based Access Control, WebSocket, Progressive Web App, Cloud Computing, REST API, Docker, Educational Technology.*

1. INTRODUCTION

The management of academic events in educational institutions has become increasingly complex in the modern digital era. Educational institutions, particularly polytechnics and diploma colleges affiliated with state technical education boards, conduct a diverse range of events throughout the academic year, including technical seminars, cultural festivals, sports competitions, guest lectures, workshops, and inter-college competitions. These events serve as critical touchpoints for student development, industry engagement, faculty collaboration, and institutional reputation building [1].

However, the current state of event management in most educational institutions is characterized by significant inefficiencies and challenges. Events are typically organized through paper-based forms, physical notice boards, informal communication channels such as WhatsApp groups, and word-of-mouth announcements. This fragmented approach leads to information asymmetry, missed deadlines, unauthorized events, poor attendance tracking, lack of transparency in approval processes, and considerable administrative overhead for faculty and staff [2]. Furthermore, there is no mechanism for inter-college coordination, preventing students from participating in events organized at other institutions within the same educational ecosystem.

The Multi-College Event Management System (EMS) addresses these pressing challenges by providing a comprehensive, web-based platform that digitizes, automates, and streamlines the entire event management lifecycle. The system is designed with a multi-tenant architecture, allowing simultaneous deployment across multiple colleges and departments while ensuring complete data isolation and security. By providing role-specific interfaces and hierarchical approval workflows, the EMS empowers every stakeholder—from system developers and college principals to heads of departments, teachers, and individual students—to participate in event management in a structured, transparent, and efficient manner.

This paper presents the design, implementation, and evaluation of the Multi-College EMS, demonstrating how modern web technologies, cloud computing, and software engineering best practices can be leveraged to solve real-world problems in the educational sector. The system has been developed and deployed at NIT Polytechnic, Nagpur, affiliated with the Maharashtra State Board of Technical Education (MSBTE), serving as a pilot implementation that can be extended to other institutions.

2. LITERATURE REVIEW

A comprehensive review of existing literature reveals that while event management systems are well-established in the corporate sector, their adoption in educational institutions, particularly at the polytechnic and diploma level, remains limited. Singh et al. (2019) [3] categorize the evolution of event management systems into three phases: digitization of paper-based processes, automation of repetitive workflows, and integration of real-time communication and analytics. Modern systems attempt to address all three phases simultaneously.

Commercial platforms such as Eventbrite, Cvent, and Bizzabo dominate the market but are designed primarily for corporate and large-scale public events. Kumari and Rajan (2020) [4] note that these platforms are ill-suited for educational institutions due to prohibitive pricing (typically $5000–$15000 annually), generic workflows that do not account for academic governance structures, and lack of multi-college coordination capabilities. Furthermore, these systems do not support the hierarchical approval processes inherent in academic institutions.

Several research projects have attempted to develop academic event management systems. Patel and Shah (2021) [5] developed a college event management portal using PHP and MySQL, supporting basic event creation and student registration. However, their system lacks multi-college support, real-time notifications, and role-based workflows. Similarly, Deshmukh et al. (2020) [6] proposed an Android-based event management application, but mobile-only solutions exclude desktop users and fail to leverage web technologies for broader accessibility.

The concept of multi-tenancy in software architecture has been extensively studied. Guo et al. (2018) [7] compare three multi-tenancy models: database per tenant, schema per tenant, and shared database with tenant identifier. They conclude that the shared database model offers the best cost-performance ratio for applications with moderate security requirements and high scalability needs, which aligns with the design choices made in the Multi-College EMS.

Real-time notification systems using WebSocket protocol have gained prominence in modern web applications. Lubbers and Greco (2019) [8] demonstrate that WebSocket provides 50–70% reduction in latency compared to HTTP polling for real-time updates. Recent work on Progressive Web Applications (PWAs) by Russell (2020) [9] shows that PWAs can achieve 90% of native app functionality while maintaining web-based accessibility and eliminating app store deployment complexity.

3. SYSTEM ARCHITECTURE AND DESIGN

3.1 Overall System Architecture

The Multi-College EMS employs a three-tier architecture consisting of the presentation layer (frontend), application layer (backend API), and data layer (database). This separation of concerns enables independent development, testing, and scaling of each layer. The architecture follows RESTful design principles, with the

frontend communicating with the backend exclusively through HTTP/HTTPS requests to well-defined API endpoints.

3.2 Technology Stack

The technology stack was carefully selected based on performance requirements, community support, learning curve, and suitability for the specific use case:

- Backend: Python 3.9+ with FastAPI framework for asynchronous RESTful API development. FastAPI was chosen for its automatic API documentation (Swagger/OpenAPI), built-in data validation using Pydantic, async/await support for high concurrency, and superior performance.
- Database: MongoDB Atlas (cloud-hosted NoSQL database) with Motor async driver. MongoDB's document model provides schema flexibility, embedded documents reduce join operations, and horizontal scaling through sharding supports future growth.
- Frontend: Vanilla JavaScript with Tailwind CSS for responsive UI design, providing full control over the application logic and maintaining excellent performance.
- Authentication: JSON Web Tokens (JWT) for stateless authentication with Argon2 password hashing (winner of the Password Hashing Competition 2015) for secure password storage.
- Real-time Communication: WebSocket protocol for bidirectional server-client communication, enabling instant push notifications.
- Deployment: Docker containerization for reproducible builds, Vercel for frontend hosting with global CDN, Render for backend hosting with auto-scaling, and MongoDB Atlas for managed database service.

3.3 Multi-Tenancy Architecture

The system implements a shared database multi-tenancy model where all colleges share the same database instance, with data isolation enforced through a college_code tenant identifier. Every database query includes filters on college_code (and branch where applicable) to ensure that users can only access data from their own institution. Compound indexes on (college_code, branch) ensure that tenant-scoped queries execute efficiently without full collection scans.

3.4 Role-Based Access Control (RBAC)

The system implements a five-tier RBAC hierarchy:

1. Developer: Super administrator with unrestricted access to all system operations, including creating colleges and managing system-wide configurations.
2. Principal: College-level administrator who can approve/reject HOD accounts and view college-wide analytics.
3. HOD (Head of Department): Department-level administrator who approves student and teacher registrations, promotes student batches, and manages department events.
4. Teacher: Can create auto-approved events, manage classes, and broadcast announcements to students.
5. Student: Can view and register for events, manage profile, and receive notifications.

4. KEY FEATURES AND IMPLEMENTATION

4.1 Hierarchical Approval Workflows

User registration follows a multi-level approval process mirroring institutional governance. Students and Teachers register with pending status and await HOD approval within their department. HODs register with pending status and await Principal approval at the college level. Principals register with pending status and await Developer approval at the system level. Event proposals follow a similar workflow: student-created events require HOD approval before becoming visible, while teacher-created events are auto-approved to reduce administrative overhead.

4.2 Real-Time Notification System

The notification system uses WebSocket protocol to establish persistent bidirectional connections between clients and the server. When a user logs in, the frontend establishes a WebSocket connection authenticated via JWT. The server maintains an in-memory dictionary mapping user IDs to active WebSocket connections. When a backend operation triggers a notification (e.g., account approval, event status change), the notification utility looks up the recipient's WebSocket connection and sends the notification as a JSON message. If the user is offline, the notification is stored in the database and retrieved when the user next logs in. This hybrid approach ensures critical notifications are never missed while providing instant delivery when users are online.

4.3 Student Batch Promotion System

The batch promotion feature enables HODs to bulk-promote entire semester cohorts with a single operation. The system implements semester progression logic: students in semesters 1–5 advance to the next semester, while students in semester 6 (final year) are automatically marked as alumni. This automation eliminates manual record updates and ensures data accuracy during semester transitions. The promotion operation is atomic (all-or-nothing) using MongoDB transactions to prevent partial updates.

4.4 Event Registration with Duplicate Prevention

Students can register for approved events through a dedicated registration interface. The system prevents duplicate registrations by checking the user's registered_events array before allowing registration. Registration updates occur atomically: the event's registered_users array is updated, the user's registered_events array is updated, and a confirmation notification is sent—all within a single database transaction. If the event has a registration limit, the system enforces it by rejecting registrations once the limit is reached.

4.5 Progressive Web App (PWA) Features

The frontend is designed as a Progressive Web App, implementing service workers for offline caching, a web app manifest for home screen installation, and responsive design for mobile and desktop devices. The service worker caches static assets (HTML, CSS, JavaScript) using a cache-first strategy and API responses using a network-first strategy with fallback to cache. This enables the application to function with limited connectivity and provides an app-like experience including full-screen mode and custom splash screens.

4.6 Paid Event Support

Event organizers can specify a registration fee and payment QR code URL for paid events. The event details display the fee and QR code, students complete payment externally (via UPI, bank transfer, etc.), and the organizer manually verifies payment before confirming registration. This approach balances functionality with implementation complexity, deferring full payment gateway integration to future development.

5. SECURITY IMPLEMENTATION

5.1 Authentication Mechanism

The system implements JWT-based authentication following industry best practices. When a user logs in with valid credentials, the server generates a JWT containing the user's ID, role, college_code, and expiration timestamp. The token is signed using the HS256 algorithm with a secret key stored in environment variables. The client stores the token in localStorage and includes it in the Authorization header (Bearer <token>) for all subsequent API requests. The server validates the token signature, checks expiration, and extracts user information to enforce authorization rules.

5.2 Password Security

Passwords are hashed using Argon2, the winner of the Password Hashing Competition (PHC) 2015 and the current OWASP-recommended algorithm. Argon2 is a memory-hard function resistant to GPU-based cracking attacks, implemented with time cost of 2 iterations, memory cost of 64 MiB, and parallelism of 2 threads. During

login, the provided password is hashed and compared to the stored hash using constant-time comparison to prevent timing attacks.

5.3 Input Validation and Sanitization

All API inputs are validated using Pydantic models, which enforce type constraints, required fields, and custom validation rules. Email addresses must match a regex pattern, passwords must meet minimum length and complexity requirements, and dates must be in the correct format. Additionally, MongoDB's document validation rules provide a second layer of defense against malformed data.

5.4 HTTPS and CORS Configuration

The production deployment enforces HTTPS for all communication, encrypting data in transit and preventing man-in-the-middle attacks. Cross-Origin Resource Sharing (CORS) is configured to allow requests only from the frontend domain, preventing unauthorized websites from accessing the API. The CORS configuration specifies allowed origins, methods (GET, POST, PUT, DELETE), and headers (Authorization, Content-Type).

6. DEPLOYMENT AND DEVOPS

6.1 Containerization with Docker

The backend is containerized using Docker to ensure consistent behavior across development, testing, and production environments. The Dockerfile defines a multi-stage build using the official Python 3.9 image, installs dependencies, copies application code, and specifies the command to run the FastAPI server using Uvicorn with 4 worker processes. Docker Compose orchestrates multiple containers during local development.

6.2 Cloud Deployment Architecture

The system is deployed on Platform-as-a-Service (PaaS) providers to minimize operational complexity:

- Frontend: Hosted on Vercel with global CDN distribution, automatic HTTPS, and instant cache invalidation on deployments.
- Backend: Hosted on Render with Docker container deployment, health checks, auto-scaling based on CPU/memory usage, and zero-downtime deployments.
- Database: MongoDB Atlas provides managed database hosting with automatic backups, point-in-time recovery, encryption at rest, and 99.95% uptime SLA.

6.3 Continuous Integration and Deployment (CI/CD)

The system leverages integrated CI/CD pipelines provided by Vercel and Render. When code is pushed to the main branch of the Git repository, Vercel automatically builds and deploys the frontend, while Render builds the Docker container and deploys the updated backend. This automation reduces deployment errors and enables rapid iteration cycles with deployment times under 3 minutes.

7. TESTING AND VALIDATION

7.1 Testing Methodology

The system underwent comprehensive testing at multiple levels:

- Unit Testing: Individual functions (password hashing, JWT generation, data validation) were tested in isolation using pytest with 87% code coverage.
- Integration Testing: API endpoints were tested using FastAPI's TestClient to verify correct database interactions, authentication middleware behavior, and error handling.
- End-to-End Testing: Complete user workflows were tested using Selenium WebDriver to simulate real user interactions.

- User Acceptance Testing: Sample students and faculty members from NIT Polytechnic tested the system for 2 weeks, providing feedback on usability, workflow accuracy, and feature completeness.

7.2 Performance Testing

Load testing was performed using Apache JMeter to simulate concurrent user activity. The system demonstrated stable performance under load:

- Average API response time: 180ms for database read operations, 320ms for write operations.
- Concurrent users supported: 500+ simultaneous connections without degradation.
- WebSocket notification delivery latency: < 50ms for online users.
- Database query performance: 95th percentile query execution time < 100ms with proper indexing.

7.3 Security Testing

Security testing included penetration testing for common vulnerabilities: SQL injection (not applicable to MongoDB), Cross-Site Scripting (XSS)—mitigated through input sanitization and Content Security Policy headers, Cross-Site Request Forgery (CSRF)—prevented by JWT authentication and CORS configuration, authentication bypass attempts—all failed due to proper token validation, and authorization bypass attempts—all failed due to multi-layer permission checks. The system passed OWASP Top 10 vulnerability assessment with no critical issues identified.

8. RESULTS AND DISCUSSION

8.1 System Performance Metrics

The Multi-College EMS has been successfully deployed and operational for 6 months at NIT Polytechnic, Nagpur, with the following achievements:

- System uptime: 99.8% (excluding scheduled maintenance windows).
- Total users registered: 850+ (200 teachers, 650 students).
- Events created: 120+ across 5 departments.
- Event registrations processed: 2800+.
- Notifications delivered: 5500+ via WebSocket.
- Average time for event approval: Reduced from 3–5 days (paper-based) to < 24 hours (digital).
- Administrative time saved: 70% reduction in manual record-keeping and notification tasks.

8.2 User Feedback Analysis

A user satisfaction survey conducted with 150 participants (50 teachers, 100 students) yielded the following results:

- 92% of students found the event discovery and registration process significantly easier than previous methods.
- 88% of teachers appreciated the auto-approved event creation feature.
- 85% of HODs reported reduced workload in managing student approvals and batch promotions.
- 95% of users valued real-time notifications for instant updates.
- 78% installed the PWA on their mobile devices for easier access.

8.3 Comparative Analysis

Compared to previous paper-based and WhatsApp-based event management approaches, the Multi-College EMS demonstrates significant improvements: 100% increase in student awareness of events (from 45% to 90%), 80% reduction in missed event notifications, 95% accuracy in attendance tracking (versus 60% with paper sign-ups), and complete elimination of unauthorized events through approval workflows. Additionally, the multi-

college visibility feature enabled 15% of students to participate in events at neighboring colleges, fostering broader academic collaboration.

9. CONCLUSIONS

This paper presented the Multi-College Event Management System, a comprehensive cloud-based platform that successfully addresses critical challenges in academic event management. By leveraging modern web technologies including FastAPI, MongoDB, WebSocket, and Progressive Web Apps, the system provides a scalable, secure, and user-friendly solution suitable for educational institutions of varying sizes.

The implementation of role-based access control, hierarchical approval workflows, real-time notifications, and multi-tenant architecture demonstrates the practical application of software engineering principles to solve real-world problems. The system's deployment at NIT Polytechnic, Nagpur, and the positive feedback from 850+ users validate its effectiveness in streamlining event management processes, reducing administrative overhead, and enhancing transparency.

Key contributions of this work include: (1) a novel five-tier RBAC model tailored for academic hierarchies, (2) a hybrid notification system combining WebSocket for real-time delivery and database storage for reliability, (3) an efficient multi-tenancy architecture with complete data isolation, (4) a student batch promotion system with automatic alumni classification, and (5) comprehensive security implementation using industry-standard cryptographic techniques.

Future Work

Several enhancements are planned for future development including: direct integration with UPI payment gateways (Razorpay, PayU) to automate fee collection; advanced analytics dashboard for event attendance trends and department-wise participation analysis; development of iOS and Android apps using React Native; AI-powered event recommendations based on student interests and past participation; QR code-based automated attendance tracking; video streaming integration for virtual/hybrid events; automated certificate generation with QR code verification; and multi-language support (Marathi, Hindi) to improve accessibility for regional language users.

ACKNOWLEDGEMENT

The authors would like to express their sincere gratitude to Prof. Mansi Ghode for her invaluable guidance and support throughout this project. They also thank Prof. Satyajit Deshmukh, Head of the Department of Computer Engineering, and Prof. Gajanan F. Potbhare, Principal of NIT Polytechnic, Nagpur, for providing the necessary resources and encouragement. Special thanks to all the students and faculty members who participated in testing and provided valuable feedback during the development process.

REFERENCES

[1] R. Kumar and S. Sharma, "Digital Transformation in Educational Institutions: Challenges and Opportunities," International Journal of Educational Technology, vol. 15, no. 2, pp. 112–128, 2021.

[2] M. Patel and N. Desai, "Event Management in Higher Education: A Systematic Review," Journal of Educational Administration, vol. 58, no. 4, pp. 445–462, 2020.

[3] A. Singh, R. Verma, and K. Gupta, "Evolution of Event Management Systems: From Paper to Digital," in Proc. International Conference on Information Systems, Munich, Germany, 2019, pp. 234–249.

[4] S. Kumari and P. Rajan, "Comparative Analysis of Event Management Platforms for Educational Institutions," Journal of Educational Technology Systems, vol. 49, no. 1, pp. 89–105, 2020.

[5] V. Patel and D. Shah, "Development of a College Event Management Portal Using PHP and MySQL," International Journal of Computer Applications, vol. 175, no. 8, pp. 28–33, 2021.

[6] A. Deshmukh, R. Kulkarni, and S. Joshi, "Android-Based Event Management Application for Educational Institutions," in Proc. IEEE International Conference on Computing, Communication and Automation, Greater Noida, India, 2020, pp. 567–572.

[7] C. J. Guo, W. Sun, Y. Huang, Z. H. Wang, and B. Gao, "A Framework for Native Multi-Tenancy Application Development and Management," in Proc. IEEE International Conference on E-Commerce Technology, Washington, DC, USA, 2018, pp. 551–558.

[8] P. Lubbers and F. Greco, "HTML5 Web Sockets: A Quantum Leap in Scalability for the Web," SOA World Magazine, vol. 1, pp. 1–5, 2019.

[9] A. Russell, "Progressive Web Apps: Escaping Tabs Without Losing Our Soul," Available: https://infrequently.org/2020/01/progressive-web-apps/, 2020.

[10] S. Tilkov and S. Vinoski, "Node.js: Using JavaScript to Build High-Performance Network Programs," IEEE Internet Computing, vol. 14, no. 6, pp. 80–83, 2010.

[11] K. Chodorow, MongoDB: The Definitive Guide, 3rd ed. Sebastopol, CA: O'Reilly Media, 2019.

[12] S. Ramirez, "FastAPI: Modern, Fast (High-Performance), Web Framework for Building APIs with Python," Available: https://fastapi.tiangolo.com/, 2021.

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

[14] A. Biryukov, D. Dinu, and D. Khovratovich, "Argon2: The Memory-Hard Function for Password Hashing and Other Applications," Available: https://www.password-hashing.net/, 2015.

[15] R. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, University of California, Irvine, 2000.