Design and Implementation of a Comprehensive College Event Management System

Aby Thankachan
Department of Computer Science,
TKM College of Engineering,
Kollam, India

Muhammed Bin Shafeeq T.P.
Department of Computer Science,
TKM College of Engineering,
Kollam, India

Shanthanu S. Nair
Department of Computer Science,
TKM College of Engineering,
Kollam, India

Shijin Kumar
Department of Computer Science,
TKM College of Engineering,
Kollam, India

Sreethi S.
Department of Computer Science,
TKM College of Engineering,
Kollam, India

Shameem Ansar, PhD
Department of Computer Science,
TKM College of Engineering,
Kollam, India

ABSTRACT

This paper presents the design and implementation of a unified College Event Management System aimed at enhancing academic and cultural activities on campus. The system supports event management, emergency alerts, resource allocation, quality assurance, and peer support, serving students, club heads, faculty, and administrators. Key features include role-based access control, event registration and certification, real-time notifications, event reviews, and analytics.

Keywords

Event management system, college, events, quality assurance, academic environment, resource allocation.

1. INTRODUCTION

Modern educational institutions face increasing challenges in efficiently managing campus activities, maintaining facilities, allocating resources, ensuring safety, and fostering community engagement. Traditional approaches to event management and campus operations often involve disconnected systems that lead to communication gaps, inefficient resource allocation, and reduced engagement. These fragmented solutions typically require separate interfaces for event planning, facility maintenance, community interactions, and overall supervision, creating unnecessary complexity for users and administrators alike.

The proliferation of campus events—from academic lectures and workshops to social gatherings and club activities—demands a sophisticated management approach that transcends conventional methods. Universities and colleges typically host hundreds of events annually, each requiring coordination among multiple stakeholders, resource allocation, promotion, registration, and feedback collection. Without an integrated system, this process becomes cumbersome, error-prone, and resource-intensive. The College Event Management System addresses these challenges by providing a centralised platform that combines real-time notifications, event registration, attendance marking, event scheduling, venue allotment, certificate repository, maintenance reporting, and peer support resources. This integration enhances the overall campus experience while streamlining administrative processes and improving response times to various campus issues. The system not only facilitates event-related tasks but also creates a connected campus ecosystem where information flows seamlessly between different stakeholders, enabling more efficient operations and fostering a stronger sense of community.

By implementing this comprehensive system, educational institutions can expect significant improvements in various operational metrics, including reduced administrative overhead, faster response times to emergencies and maintenance issues, increased event participation, and enhanced community engagement. These benefits contribute to the institution's primary goals of providing a safe, productive, and enriching educational environment for all community members.

2. METHODOLOGY

The College Event Management System is built using a monolithic architecture, powered by Java Spring Boot [5] for the backend, React [4] for the frontend, and PostgreSQL for storage [6]. Additionally, cloud storage is used for managing media assets. This architectural choice was made after careful analysis of the system's functional and non-functional requirements. A monolithic approach was preferred over microservices [2] due to its simpler development and deployment process, which helps in reducing initial complexity. Since campus event management workflows are tightly coupled, a monolithic structure ensures better performance and easier maintenance, avoiding the overhead of managing multiple independent services. For the backend, Spring Boot was selected due to its enterprise-grade capabilities, robust dependency injection framework, built-in security features, and seamless database integration through Spring Data [1, 5]. It provides powerful transaction management, ensuring data consistency and reliability. Additionally, Spring Security enables JWT-based authentication and role-based access control (RBAC), ensuring secure access to sensitive information [3]. Although Spring Boot requires more server resources compared to lightweight alternatives such as Node.js. Its stability, scalability, and maintainability make it a strong choice for handling complex business logic and event-driven workflows.

On the frontend, React was chosen due to its component-based architecture, virtual DOM implementation, and strong community support [4]. The virtual DOM optimises re-rendering performance, making the UI more efficient and responsive. React's extensive ecosystem allows integration with state management libraries such as Redux, enabling efficient data flow and state synchronisation. Despite React's steeper learning curve compared to alternatives like Vue.js, its flexibility and long-term maintainability make it an ideal choice for dynamic and interactive web applications [1].

PostgreSQL was selected for its ACID compliance, strong transactional integrity, and relational capabilities, making it suitable for structured data such as user profiles, event registrations, and attendance records [6].

To handle media assets such as event photos, promotional materials, and maintenance documentation, the system utilises cloud storage solutions. This approach provides high scalability, reliability, and cost-effective storage options, reducing reliance on on-premises infrastructure. Additionally, integration with content delivery networks (CDNs) enhances global accessibility and faster content delivery. However, cloud storage comes with operational costs and potential vendor lock-in concerns, requiring careful planning to ensure long-term cost efficiency and data portability. Security is a critical aspect of the College Event Management System. To protect user data and maintain regulatory compliance, the system employs multiple layers of security mechanisms. Data encryption is implemented for both data at rest and data in transit to prevent unauthorised access. JWT-based authentication ensures secure user sessions, while role-based access control (RBAC) restricts access to sensitive functionalities based on user roles [3]. Additionally, the system undergoes regular vulnerability scanning to identify and mitigate security risks. The system is designed to comply with GDPR (General Data Protection Regulation) standards, ensuring the privacy and security of user data. Infrastructure provides high availability, minimising downtime and protecting critical information.

By integrating cutting-edge technologies and robust security measures, the College Event Management System ensures high performance, scalability, and reliability, making it a highly effective and relevant solution for campus event coordination [1]. The system's performance is driven by Spring Boot's efficient backend processing, React's optimised UI rendering, and PostgreSQL for data storage. This enables fast event registrations, real-time email updates, and seamless user interactions. Cloud storage further enhances efficiency by handling large media assets without burdening local infrastructure. Its monolithic architecture simplifies initial development while allowing for future modularisation if needed [2]. Cloud integration and optimised database strategies ensure smooth performance, even with increasing data loads. Reliability is reinforced through JWT-based authentication, role-based access control (RBAC), and data encryption, ensuring secure and compliant

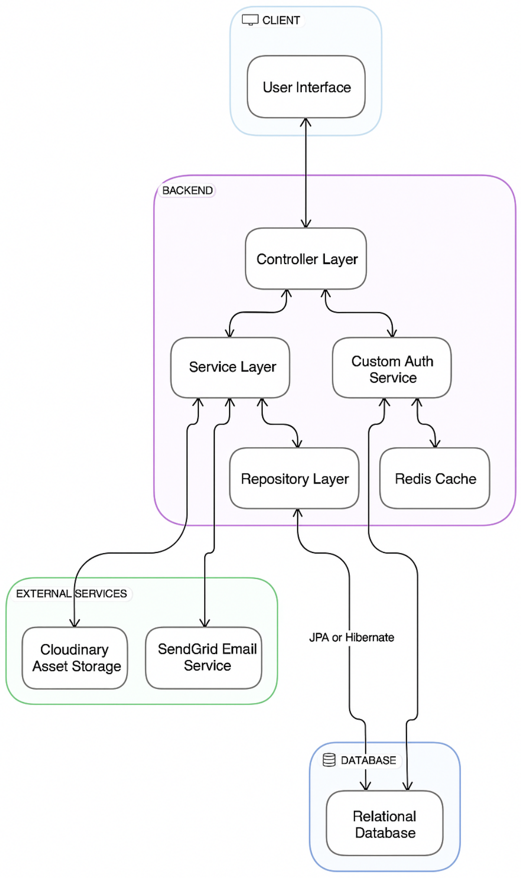

Fig. 1. System architecture.

event management. By streamlining event organisation, enhancing security, and ensuring seamless user experiences, the College Event Management System serves as a powerful, future-ready solution that significantly improves campus-wide event coordination and engagement.

To optimise performance, frequently accessed data such as user roles and sessions is cached using Redis, reducing the load on the database.

3. USER AUTHENTICATION AND AUTHORIZATION FLOW – OVERVIEW

The User Authentication and Authorisation Flow describes how a secure backend system handles login, access control, and session management using JWT tokens and Redis [3]. When a user attempts to log in, their email and password are sent to the AuthController, which delegates the validation process to the AuthService. The service loads user details from the database and, if the credentials are valid, generates a JWT access token and a refresh token. The access token is stored in the client's memory for API requests, while the refresh token is stored in a secure HttpOnly cookie. Optionally, the refresh token is also saved in Redis for added security.

For any protected API call, the client includes the access token in the request. A JWTFilter extracts and validates this token, then sets the authenticated user in the SecurityContext. Before granting access, the system checks the user's roles and permissions to ensure they are authorised to perform the requested action [3]. If the access token expires, the client can call the /auth/refresh endpoint to obtain a new one. The server validates the refresh token (checking Redis if necessary), and if it's still valid, a new access token is issued. When the user logs out, both the access and refresh tokens are blacklisted and stored in Redis to prevent any future use. Additionally, RateLimitFilter is used to monitor the number of API requests from each user, blocking excessive requests to prevent abuse. The authentication and authorisation flow is shown in Figure 2.

4. ROOM BOOKING APPROVAL SYSTEM

The Room Booking Approval System is designed to facilitate and streamline the process of reserving rooms for academic, cultural, and administrative events within the college. The system ensures that bookings are handled efficiently, reducing manual intervention while maintaining transparency and accountability at each step.

The process begins when a user—such as a student, faculty member, or event coordinator—submits a booking request through the system's interface. This request typically includes essential information such as the event title, date, time, duration, and preferred room or hall, along with any additional requirements. Upon submission, the system performs an automated conflict check by comparing the requested time slot and room with existing reservations.

If a scheduling conflict is detected, such as the room already being booked or another overlapping event, the system promptly rejects the request. A rejection email is sent to the user, clearly stating the reason for the denial. This helps avoid double bookings and allows users to modify and resubmit their requests if needed.

If no conflicts are found, the request moves to a structured, multi-level approval process. The first level of approval is handled by the Faculty Coordinator, who reviews the request for its purpose and relevance. Once approved, the request is passed on to the Room Coordinator, who verifies room suitability and availability. If approved again, it moves to the Security Department, which ensures that the event adheres to campus safety guidelines.

The final stage of approval is managed by the Office Admin, who performs an overall review to ensure that institutional policies are met and all prior approvals are in place.

When all levels of approval are successfully completed, the booking is marked as Fully Approved, and the user receives a confirmation email with booking details. However, if the request is rejected at any point in the workflow—by any of the approvers—it is immediately routed to a rejection path, and the system sends an email to notify the user of the decision and the reason behind it.

To support better request management, the system also provides a dedicated dashboard for faculty members and coordinators to view pending requests. This interface is equipped with features like pagination, sorting, and filtering, making it easy to handle large numbers of requests and quickly locate specific entries based on criteria such as date, requester name, or approval status.

Overall, the Room Booking Approval System promotes efficient event planning, prevents scheduling conflicts, and ensures that all room reservations follow a clear and accountable approval process as the event title, date, time, duration, and preferred room or hall, along with any additional requirements. Upon submission, the system performs an automated conflict check by comparing the requested time slot and room with existing reservations.

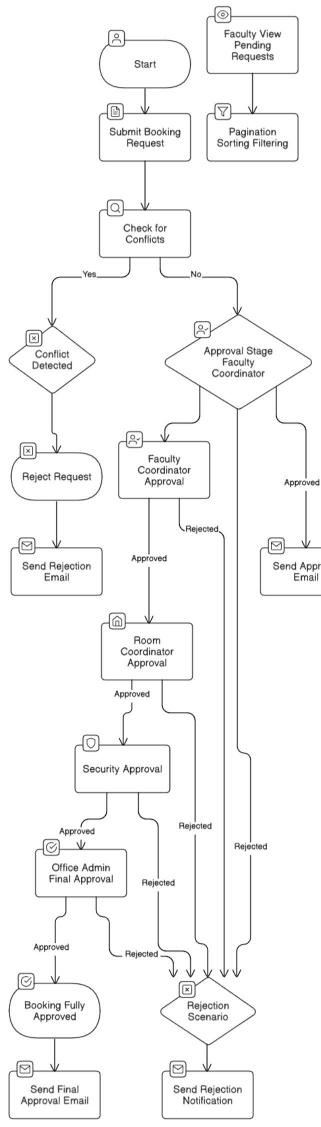

Fig. 3. Room booking approval system.

5. PERFORMANCE TESTING AND RESULTS

The proposed College Event Management System was implemented as a full-stack web application using Spring Boot for the backend, React for the frontend, and PostgreSQL as the database [1, 4, 5, 6]. Then the website is hosted on Render, NeonDB is used for hosting the PostgreSQL database.

(1) Response time test
    Objective: Measure the average response time of APIs under normal usage.
    Tool: Postman

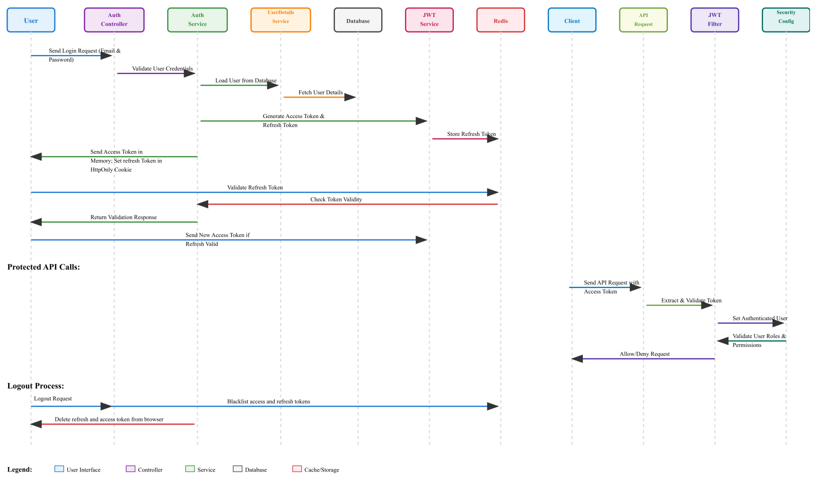

Fig. 2. Authentication and authorisation flow.

Table 1. API performance.

API Endpoint	Method	Average Response Time
/student/signup	POST	2.63 seconds
/faculty/signup	POST	1.40 seconds
/auth/login	POST	4.50 seconds
/club/create	POST	3.60 seconds

Observation: The login API showed the highest latency, potentially due to authentication overhead and searching.

(2) Concurrent user load test

Objective: Evaluate API behaviour under simultaneous access by multiple users.

Tool: Apache JMeter

Setup: 20 virtual users, 100 requests total to /auth/login

Results:

—Average Response Time: 706 ms
—Max Response Time: 4117 ms
—Error Rate: 1.0%
—Throughput: 7.57 requests/sec

Observation: The API handled moderate concurrency with some delay and a minor error rate.

(3) Spike test

Objective: Test API resilience during sudden user traffic spikes.

Tool: Apache JMeter

Setup: 200 users sent requests at once to /auth/login

Results:

—Average Response Time: 1280 ms
—Max Time: 5923 ms
—Error Rate: 3.5%
—Throughput: 15.2 requests/sec

Observation: Performance degraded with increased error rates under spike conditions, expected with free tier hosting.

(4) Data-driven API test report - /auth/login endpoint

Objective: To evaluate the performance and stability of the login API (/auth/login) under realistic usage by simulating logins from a diverse set of users using Apache JMeter with a CSV-driven data source.

Tool: Apache JMeter

Table 2. Performance results (summary report).

Metric	Value
Total Requests	10
Average Response Time	840 ms
Minimum Response Time	631 ms
Maximum Response Time	1357 ms
Standard Deviation	251.86 ms
Error Rate	0.0%
Throughput	4.82 requests/sec
Latency (Avg)	276 ms
Average Response Size	1.18 KB

Observations

—All 10 login attempts were successful (0% error rate).
—Average response time was 840 ms, which is acceptable for an authentication API.
—Response time variability (StdDev = 251.86 ms) suggests occasional delays, possibly due to backend factors like database lookup or server cold starts.
—Throughput was 5 requests/sec, which is fine for moderate usage.

This test confirms that the login API performs well under small user loads and handles varied inputs correctly.

(5) Throughput shaping test report – /auth/login

Objective: To assess the performance and stability of the /auth/login API under a controlled throughput rate using Constant Throughput Timer in Apache JMeter.

Tool: Apache JMeter

Observations

—The response times are stable even under a constant throughput.

Table 3. Throughput shaping analysis (summary report).

Metric	Value
Samples (Requests)	73
Average Response Time	793 ms
Minimum Time	516 ms
Maximum Time	1364 ms
Standard Deviation	185.62 ms
Error Rate	0.0%
Throughput	0.575 requests/sec
KB/sec	0.155
Average Bytes	0.14 KB
Latency (Avg)	276 ms

Table 4. Aggregate report analysis (percentiles).

Percentile	Response Time
50th (Median)	793 ms
90th	1028 ms
95th	1152 ms
99th	1345 ms
Min / Max	516 ms / 1364 ms

—The 95th percentile is 1152 ms, which indicates a few slower requests but nothing critical.
—Error rate is 0%, which shows solid reliability and backend resilience.
—Std deviation (186 ms) is acceptable at low traffic volume.

This test confirms that the login API performs reliably under shaped, low-frequency traffic. It handles CSV-driven user logins without error and shows slightly increasing latency at high percentiles, but nothing alarming. Also, it is suitable for light-to-moderate login traffic workloads.

Conclusion: While the APIs performed well under normal and moderate concurrent loads, performance degraded slightly under spike conditions. Given the limitations of free-tier hosting on Render, Redis, and Neon DB, the results are reasonable. For production environments, scaling and optimisation strategies (e.g., caching, connection pooling, warm starts) would be recommended [1].

6. CONCLUSION

The College Event Management System provides a centralised, secure, and scalable solution for streamlining event coordination across college campuses. By integrating multiple modules—such as user authentication, room booking approvals, real-time notifications, and role-based access control—the system addresses the core challenges faced in organising academic and cultural activities. The use of modern web technologies like Spring Boot and React, combined with a PostgreSQL database and cloud infrastructure, ensures high performance, flexibility, and ease of maintenance [1, 4, 5, 6]. Through automation, enhanced usability, and strict security protocols, the platform reduces administrative overhead, improves collaboration among stakeholders, and fosters a vibrant campus environment. Future enhancements could include AI-driven event recommendations, deeper analytics, and integration with external calendaring systems, further strengthening its impact and adaptability. Overall, the proposed system stands as a robust and future-ready tool to support and elevate campus event management in educational institutions.

7. REFERENCES

[1] G. Thombare, P. Jadhav, S. Dhere, P. Jadhav, and Dr. K. N. Honwadkar, “Event Management System using React and Spring,” *International Research Journal of Modernization in Engineering Technology and Science*, 2023.

[2] M. Fowler, “Microservices vs Monoliths,” *martinfowler.com*, 2015. [Online]. Available: https://martinfowler.com/articles/microservices.html

[3] Y. A. Marquis, “From Theory to Practice: Implementing Effective Role-Based Access Control Strategies to Mitigate Insider Risks in Diverse Organizational Contexts,” *Journal of Engineering Research and Reports*, vol. 26, no. 5, pp. 138–154, 2024.

[4] “React – A JavaScript library for building user interfaces,” ReactJS.org. [Online]. Available: https://reactjs.org/

[5] “Spring Boot Reference Documentation,” Spring.io. [Online]. Available: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/

[6] “PostgreSQL Documentation,” PostgreSQL.org. [Online]. Available: https://www.postgresql.org/docs/