

UniEvent Hub: A Centralized Web Platform for Inter-College Event Discovery and Participation

Durgasri R¹, Ridha N², Rithika N C³, Sridharshini V⁴ and Mrs. M. Minu Sunitha Mary⁵

^{1,2,3,4}UG Students, ⁵ Associate Professor, Department of Computer Science and Engineering

(Artificial Intelligence and Data Science),

School of Engineering, Avinashilingam Institute for Home Science and Higher Education for Women,

Coimbatore 641108, Tamil Nadu, India

ABSTRACT

College and university campuses across India run a steady stream of technical, cultural, and sports events, yet a student's chance of hearing about an event that suits them still depends heavily on informal channels — a friend studying at another institution, a notice pinned on a departmental board, or a staff member mentioning it in passing. Because there is no single place where such information is collected, many students discover an event only after registration has closed, or miss it altogether. This paper presents **UniEvent Hub**, a centralized web-based platform that brings colleges and students onto one interface for posting and discovering academic events. The system offers two distinct access modes: a faculty/institution login through which a staff member can publish and manage the events happening on their campus, and a student login through which a student signs in with their institutional details and is shown events that match their stream and interests. A follow-and-notify mechanism lets a student subscribe to the colleges they care about and be alerted whenever those colleges post something new, removing the need to check multiple sources manually. The front end is built with HTML5, CSS3, and JavaScript following responsive UI/UX practices, while MongoDB is used on the back end to store user profiles, event records, and follower relationships as flexible, document-oriented data rather than rigid relational rows. A working prototype covering registration, login, event listing, and a browsable event feed was built and tested, and this paper walks through the problem, the design decisions, the architecture, the implementation, and the results, along with the directions in which the project can still grow.

Key Words—Event Management System, Centralized Platform, Role-Based Access Control, MongoDB, Web Application, UI/UX, Student Engagement, Notification System

1. INTRODUCTION

Almost every academic calendar in an engineering college is packed with hackathons, technical symposiums, cultural festivals, and workshops, many of which are open to students from other institutions. In principle, an inter-college event is a chance for a student to step outside their own campus, meet people from other colleges, and compete or collaborate on something new. In practice, that opportunity is frequently lost simply because the information never reaches the student in time. A poster shared on one college's Instagram page rarely reaches a student enrolled elsewhere unless a mutual friend

forwards it, and even then the details — deadline, venue, registration link — often arrive incomplete or outdated.

The motivation behind **UniEvent Hub** grew out of this everyday experience. A student today typically learns about a competition through one of three channels: browsing social media and event-aggregator websites, hearing about it from friends studying at other colleges, or being informed by a staff member within their own department. None of these channels is reliable on its own, and together they still leave gaps — a student whose friend circle happens not to include anyone from a particular college, or whose department does not actively circulate notices, simply never finds out. UniEvent Hub is

designed to close this gap by giving colleges a direct channel to publish events and giving students a direct, searchable channel to discover them, instead of relying on word of mouth.

The system is built around two login pathways. A faculty login allows a staff member to add and update the events taking place at their institution — essentially turning every participating college into a publisher of its own event calendar. A student login lets a student register using their college email and department details, after which the platform can surface events relevant to their stream and notify them when a college they follow posts something new. Framing the platform this way turns event discovery from a passive, word-of-mouth activity into an active subscription: a student decides once which institutions matter to them, and the system takes over the job of watching those institutions on their behalf.

The contribution of this work is not a single novel algorithm but a complete, working system that combines several ideas that are usually found in isolation in the systems surveyed in Section II. In summary, this paper contributes: (i) a two-role authentication model that treats colleges as first-class publishers rather than passive listings; (ii) a follow-and-notify mechanism that keeps a student informed of new events from chosen institutions without repeated manual searching; (iii) a document-oriented data model built on MongoDB that accommodates the uneven, evolving way different colleges describe their events; and (iv) a working front-end prototype, tested with realistic data drawn from real regional colleges and event names, that demonstrates the approach end to end.

The rest of this paper is organized as follows: Section II reviews related work in the area of event management systems. Section III restates the problem being addressed. Section IV analyzes the functional and non-functional requirements the system must satisfy. Section V describes the proposed system and its architecture. Section VI covers the development methodology. Section VII discusses implementation details. Section VIII describes the testing carried out on the prototype. Section IX presents the results. Section X compares UniEvent Hub against the systems surveyed earlier. Section XI notes the limitations of the current prototype, Section XII discusses the platform's broader societal and educational impact, and Section XIII concludes the paper with directions for future work.

II. RELATED WORK

Web-based event management has been explored in several student projects and research papers, each addressing a slightly different slice of the problem. Shivarkar et al. describe FestFlow, a college event management system built with Django, HTML, CSS, and MySQL that uses role-based access to route events through creation, approval, registration, and feedback stages [1]. Their work shows that even a single-institution system benefits substantially from separating the concerns of event organizers and event attendees, which is a design principle carried forward in the present work but extended across multiple institutions rather than one.

Namratha et al. take a different angle, studying how social media monitoring tools can be used to gauge audience engagement and sentiment around an event, and how that data feeds back into marketing decisions for future events [2]. Their findings reinforce the idea that visibility is one of the biggest determinants of an event's turnout — a theme central to the motivation of UniEvent Hub, though the present system tries to guarantee visibility structurally, through a follow-and-notify mechanism, rather than relying on the reach of a social media post.

Kamal presents an Event Management System built on PHP, MySQL, HTML, and CSS following a waterfall development model, with modules for event listing, registration, and admin management [3]. This project, like several others surveyed during the literature review, is scoped to a single organizing body rather than a network of colleges, and does not include a subscription or notification layer that keeps a user informed after the initial visit.

Beyond the three systems studied closely, the broader literature on event and community platforms echoes the same pattern. Systems described in student mini-project reports and conference-style papers on the topic (see references [4]–[7]) tend to reproduce the same core recipe — a relational database, a set of CRUD forms for creating and listing events, and a single login type for the end user — with only minor variation in the technology stack used. None of the surveyed works treats the source of an event (the college) and the consumer of that event (the student) as two separate account types that log in through different pathways and interact with the system in

fundamentally different ways, which is the starting point UniEvent Hub takes.

Table I summarizes how the surveyed systems compare with the proposed platform along four dimensions that matter most for the problem being addressed here: whether the system supports more than one publishing institution at a time, whether it distinguishes an organizer role from an attendee role at login, whether a user can subscribe to future updates from a specific publisher, and whether the underlying data store is schema-flexible enough to absorb variation in how different organizers describe an event.

TABLE I

COMPARISON OF SURVEYED EVENT MANAGEMENT SYSTEMS

System	Multi-Institution	Role-Based Login	Follow / Subscribe	Flexible Data Model
FestFlow [1]	No	Yes	No	No (SQL)
Social-Media Analytics [2]	N/A	No	No	N/A
Kamal's EMS [3]	No	Partial	No	No (SQL)
UniEvent Hub (proposed)	Yes	Yes	Yes	Yes (MongoDB)

Taken together, the surveyed systems confirm that the building blocks of an event platform — authentication, event creation, registration, and administration — are well understood individually. What is largely missing from the systems reviewed is a genuinely multi-institution model in which many colleges publish into a shared space that students browse and subscribe to, combined with a notification layer that pushes new events to interested students instead of requiring them to keep checking back. UniEvent Hub is positioned to fill that specific gap.

III. PROBLEM STATEMENT

A student's ability to take part in an inter-college event currently depends on one of three unreliable channels: stumbling across it on an online platform, hearing about it from a friend at another college, or being told by a staff member in their own circle. There is no single, dependable source a student can check to see every event they might be eligible for. Event information stays scattered across individual college websites, social media pages, and notice boards, so awareness is inconsistent from one student to the next and often arrives too late for the student to register. There is also no structured mechanism through which a student can subscribe to a college's future events, so even a student who does discover one relevant event has no way of automatically hearing about the next one from the same institution.

IV. REQUIREMENT ANALYSIS

Before settling on the architecture described in Section V, the functional and non-functional requirements of the platform were listed out so that design decisions — which login roles to support, which database to use, which notifications to prioritize — could be checked against a concrete target rather than intuition alone.

A. Functional Requirements

The system shall:

- Allow an institution to register an account and publish, edit, and remove its own event listings.
- Allow a student to register an account using their college, department, and year of passing.
- Allow a student to browse all published events and filter them by category (technical, cultural, sports, workshop, networking).
- Allow a student to follow or unfollow any registered institution.
- Notify a student when an institution they follow publishes a new event.
- Allow a student to reach an event's registration link directly from the feed.
- Allow an administrator to view, moderate, and remove accounts or event listings that violate platform guidelines.

B. Non-Functional Requirements

Usability was treated as a first-order requirement given that the primary users are students accessing the platform between classes, often on a mobile device, so the interface needed to remain legible and operable on small screens without a separate mobile build. Performance requirements were kept modest for a prototype of this scale — event listings and login should respond within a couple of seconds under normal load — but the choice of a document database was also partly forward-looking, since MongoDB scales horizontally more easily than a single relational instance if the number of participating colleges grows well beyond the pilot set used in testing. Reliability requires that a student's follow relationships and an institution's posted events persist correctly across sessions, and security requires that passwords are never stored or transmitted in plain text and that a student account cannot modify another institution's events.

C. Hardware and Software Requirements

TABLE II

DEVELOPMENT ENVIRONMENT

Category	Requirement
Processor	Dual-core 2.0 GHz or higher
RAM	4 GB minimum (8 GB recommended)
Operating System	Windows 10/11, or any Linux distribution
Front-end	HTML5, CSS3, JavaScript (ES6)
Back-end runtime	Node.js / JavaScript server logic
Database	MongoDB (NoSQL, document-oriented)
Browser	Chrome, Firefox, or Edge (latest)
Version control	Git

D. Assumptions and Constraints

The design assumes that each participating institution designates at least one staff member responsible for keeping its event listings current, since the platform has no way to independently verify that a posted event is genuine beyond the fact that it was submitted through a registered institutional

account. It is also assumed that students access the platform primarily through a web browser rather than a dedicated app, which is why responsive design rather than a native mobile build was prioritized for this stage of the project (Section VII-A). A practical constraint on the current prototype is that it was developed and tested by a four-member student team over a single academic semester, which shaped the decision to scope the admin module narrowly (Section V-C) and to defer features such as payment integration to future work (Section XIII) rather than attempt them within the same build.

V. PROPOSED SYSTEM

UniEvent Hub addresses the problem above by giving colleges and students a shared, structured space instead of leaving discovery to chance. Two role-based login modules sit at the front of the system: a faculty/institution login through which a college can register, publish event details, and edit or remove them as plans change, and a student login through which a student registers with their college and department information, browses events filtered by category, and follows the institutions whose events interest them. A notification layer watches for new posts from followed institutions and surfaces them to the student, closing the loop that word-of-mouth discovery leaves open.

A. System Architecture

The architecture separates the system into three layers. The presentation layer is the browser-rendered interface built with HTML5, CSS3, and JavaScript, through which both students and institutions interact with the platform. The application layer, implemented in JavaScript, receives requests from the interface, applies the necessary validation and business logic — checking credentials at login, or attaching a timestamp to a new event post — and translates them into database operations. The data layer is a MongoDB instance holding three principal collections: users (covering both student and institution accounts), events (each event document carrying its title, host college, category, date, and registration link), and followers (mapping which students follow which institutions). Because MongoDB stores data as JSON-like documents rather than fixed relational tables, an event document can carry optional fields — a poster image, a set of tags, a capacity limit — without requiring a schema migration, which suits a platform

where different colleges may want to describe their events slightly differently.

B. Advantages of the Proposed Architecture

Splitting the system into three layers in this way keeps each concern independently replaceable. The presentation layer could be swapped for a native mobile client (as proposed in Section XIII) without touching the application logic, because both would talk to the same application-layer functions. The application layer isolates every rule that must hold regardless of which client is asking — that a student cannot edit another institution's event, that a password is always hashed before storage — so those rules are enforced once, centrally, rather than being re-implemented in every screen that happens to need them. And because the data layer is reached only through the application layer's data-access functions rather than being queried directly from the front end, the underlying database technology could in principle be changed later without requiring a rewrite of the user interface. This separation is a conventional three-tier design, but it is worth stating explicitly here because student mini-projects — including the ones surveyed in Section II — frequently blur the presentation and application layers together, embedding business logic directly inside page-rendering code, which becomes harder to test and maintain as the project grows.

Fig. 1 shows the top-level data flow of the system. A student's browser sends view, follow, and notification requests to the UniEvent Hub system, which in turn draws on the user and event collections to answer them and forwards relevant activity to the college or university that owns the event.

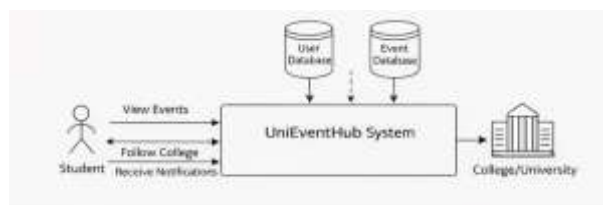


Fig. 1. Top-level data flow between a student, the UniEvent Hub system, and a college/university.

Fig. 2 expands this into the three internal processes that the application layer performs — user authentication, event management, and the follow-and-notification process — each of which reads from and writes to the underlying user, event, and follower collections before a response reaches the student.

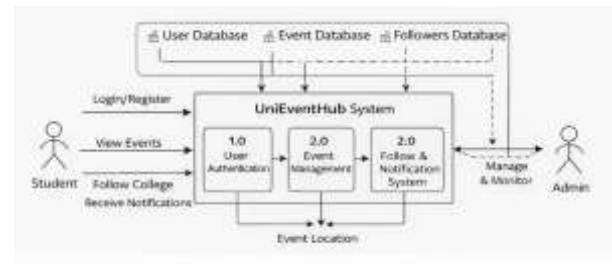


Fig. 2. Internal processes of the UniEvent Hub system: authentication, event management, and follow/notification.

Zooming further into the event management process, Fig. 5 shows how an institution's request to add, edit, or delete an event travels through the corresponding sub-process before being stored back into the event database, and how the location associated with an event is resolved and attached to the record.

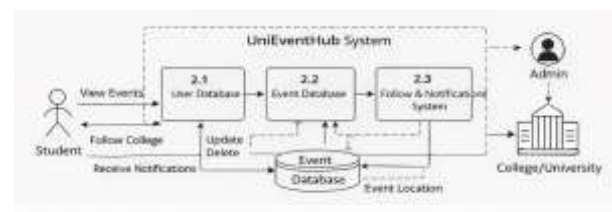


Fig. 5. Event management sub-process: add, edit, and delete operations on the event database.

C. Database Design

The data layer uses MongoDB, and the schema is organized around three collections rather than the dozen or so normalized tables a relational design of the same system would typically require. The Users collection holds one document per account, tagged with a role field (student or institution) so that a single collection can serve both without duplicating shared fields such as name, email, and password hash; student-only fields (department, year of passing) and institution-only fields (campus address, category of institution) simply do not appear on documents where they are not relevant, which a relational schema would otherwise handle through nullable columns or a separate table. The Events collection holds one document per posted event, referencing the hosting institution's user ID and carrying the event's title, category, date, description, and registration link; because MongoDB does not enforce a rigid column set, an institution that wants to attach an extra field particular to its event — a dress code, a team-size limit — can do so without a schema migration. The Followers collection maps student IDs to the institution IDs they follow, and is the collection consulted whenever a new event is posted to determine who should be notified.

Table III lists the principal fields carried by each collection. Indexing was applied on the role field in Users, the hostInstitution and category fields in Events, and a compound index on (studentId, institutionId) in Followers, so that the three queries the system runs most often — logging in, listing events by category, and checking who follows a given institution — stay fast as the collections grow.

TABLE III
PRINCIPAL DATABASE COLLECTIONS

Collection	Key Fields
Users	userId, role, name, email, passwordHash, college, department, yearOfPassing
Events	eventId, hostInstitutionId, title, category, date, description, registrationLink
Followers	studentId, institutionId, followedOn

D. Functional Modules

The system is organized into five cooperating modules:

Module I :User Interface and Authentication Renders the homepage, login, and registration screens, and validates credentials so that students, colleges, and the admin each land on a dashboard scoped to their role.

Module II :Event Management Lets an institution add, edit, or remove event records — name, date, description, category, and registration link — which are then listed for students to browse.

Module III :Follow and Notification Lets a student follow or unfollow a college and maintains that relationship in the followers collection; when a followed college posts a new event, the student is notified.

Module IV : Database and Integration Connects the front end to MongoDB, keeping user, event, and follower data consistent and retrievable in real time.

Module V : Admin Control Gives an administrator oversight of accounts and posted events, including the ability to remove content that violates platform guidelines.

These five modules were deliberately kept loosely coupled: the Follow and Notification module, for instance, only needs to know a student's ID and an institution's ID, not the internal details of how the Event Management module structures a listing, which allowed different members of the team to build and test modules in parallel once the database schema in Section V-B was agreed on. The Admin Control module was kept intentionally minimal for this prototype — covering account and listing moderation rather than a full analytics suite — since the priority for the current stage of the project was proving out the student- and institution-facing experience first.

VI. METHODOLOGY

Development proceeded in three broad phases that mirror the module breakdown above. The first phase focused on the user interface and the two login pathways, establishing the visual identity of the platform and building the forms through which a student or a college registers and signs in. Particular attention was paid to UI/UX at this stage — form fields were grouped logically, validation messages were kept close to the field they referred to, and the layout was kept responsive so that the same interface holds up on both desktop and mobile screens, since students are as likely to check the platform on a phone between classes as on a laptop.

The second phase implemented event management and the follow system: the forms and views through which a college posts an event, and the browsing and filtering interface through which a student explores events by category — technical fests, cultural fests, sports meets, workshops, and networking sessions — and follows the colleges they are interested in.

The third phase connected the interface to MongoDB and built the notification pathway. Student, event, and follower data are persisted as documents rather than relational rows, which kept the data model flexible while the event schema was still evolving during development. Whenever an institution publishes a new event, the platform checks the followers collection for that institution and queues a notification for each following student, so that the update reaches them without requiring a repeat visit to the site.

Testing was carried out iteratively alongside development. Each module — registration, login, event posting, browsing, and following — was exercised with realistic sample data (drawn from actual regional colleges and event names) to

confirm that the interface behaved correctly and that data written by one role was visible in the correct form to the other role. Section VIII discusses this testing in more detail.

The notification pathway, being the least conventional part of the system, is worth walking through as pseudocode rather than prose alone. When an institution submits a new event, the application layer executes roughly the following sequence:

```
Function on New Event Posted (event): save event to Events  
collection  
Followers = query Followers where Institution Id == event.  
Host Institution Id   for each follower in followers: create  
Notification (follower, studentId, event.id)           return  
confirmation to institution dashboard
```

This keeps the write path for posting an event and the fan-out to followers within a single request, which was acceptable at the scale exercised during testing; the future-work discussion in Section XIII notes where this would need to become an asynchronous queue if the number of followers per institution grew large.

VII. IMPLEMENTATION

A. Front-End Implementation

The interface is built with semantic HTML5 for structure, CSS3 for styling, and vanilla JavaScript for interactivity, deliberately avoiding a heavier front-end framework so that the project stayed approachable to a team building its first full web application. CSS Flexbox and Grid were used to keep the login, dashboard, and event-feed layouts responsive across screen sizes, and a consistent color palette and card-based layout were applied across screens so the platform reads as a single coherent product rather than a set of independently styled pages. Client-side JavaScript handles form validation — checking that a college email follows the expected institutional domain pattern, that a password and its confirmation match, and that required fields are not left blank — before a request is ever sent to the server, which keeps obviously invalid submissions from reaching the back end at all.

The student dashboard, shown later in Fig. 6, brings together the elements a returning student needs on one screen: a sidebar for navigating between Home, Explore, Colleges, Map, Notifications, and Messages; a summary of liked events,

followed events, and messages at the top; and a scrollable feed beneath it that mixes a search bar, quick shortcuts (top-ranked campuses, live student events, nearby opportunities), and a spotlight section highlighting selected institutions.

B. Back-End and Data Handling

The application layer is responsible for everything the front end cannot safely do on its own: verifying a submitted password against the stored hash, checking that an event edit request actually comes from the institution that owns the event, and running the notification fan-out described in Section VI. Passwords are hashed before storage rather than kept in plain text, and every write to the Events or Followers collections is scoped to the authenticated user's own ID, so that a student session cannot modify another institution's listings even by directly manipulating a request. MongoDB's driver was used to issue the create, read, update, and delete operations that back the modules described in Section V-C, with each collection accessed through a small set of functions (createUser, findEventsByCategory, addFollower, and so on) rather than scattered inline queries, which kept the codebase easier for four people to work on in parallel without stepping on each other's changes.

C. Security Considerations

Beyond password hashing and per-user write scoping, the prototype validates and sanitizes text fields before they are stored, to reduce the risk of a malicious event description or profile field being rendered back into another user's browser unescaped. Session handling relies on a signed session token rather than storing sensitive data client-side. These measures are not exhaustive — a production deployment would additionally need rate limiting on login attempts and a formal security review — but they establish the baseline expected of a system that stores personal student information such as phone numbers and dates of birth.

D. UI/UX Design Principles Applied

Because the platform's success depends on students actually returning to check it, interface design was treated as seriously as the underlying logic. Three principles guided the visual design in particular. First, information hierarchy: on the event feed, the details a student scans first — event title, hosting college, date — are placed and sized so the eye lands on them before the description, following the general expectation that a

listing page should be skimmable in a few seconds rather than requiring a full read. Second, consistent affordances: every actionable element, whether a "Register" button on an event card or a "Follow" toggle on a college profile, uses the same shape, color, and placement convention throughout the app, so a student who has learned the pattern on one screen does not have to relearn it on another. Third, forgiving forms: the registration and login screens surface validation errors next to the specific field that caused them rather than as a single generic message at the top of the page, which was found during informal testing to noticeably reduce the number of failed submission attempts among first-time users. Color was used purposefully rather than decoratively — a consistent blue-to-purple gradient identifies the UniEvent Hub brand across the login screen, dashboard, and event cards, while status colors (red for alerts, green for confirmations) are reserved exclusively for their functional meaning so they are not diluted by unrelated decorative use elsewhere in the interface.

VIII. TESTING

Testing followed the same module boundaries as development, combining manual functional testing with informal usability testing among the project team and a small group of classmates who exercised the prototype as a first-time user would. Table IV lists a representative subset of the test cases executed against the system, covering both the happy path and common failure conditions such as a duplicate registration or a mismatched password.

TABLE IV

REPRESENTATIVE TEST CASES

Test Case	Input	Expected Result	Status
Student registration	Valid college email & fields	Account created, redirected to home	Pass
Duplicate signup	Already-registered email	Signup rejected with message	Pass

Login with wrong password	Valid email, wrong password	Login denied, error shown	Pass
Institution posts event	Valid event details	Event appears in student feed	Pass
Follow institution	Student follows a college	College added to followed list	Pass
Notification on new event	Followed college posts event	Student receives notification	Pass
Category filter	Select "Sports Meet" filter	Only sports events shown	Pass
Unauthorized edit attempt	Student tries to edit another institution's event	Request rejected	Pass

All listed cases passed on the final build used for the results reported in Section IX. Minor issues found earlier in testing — a missing validation message on an empty department field, and a notification that did not clear after being read — were fixed before the build was frozen. Testing also covered boundary and negative scenarios beyond those listed in Table IV, such as submitting an event with a past date, leaving the registration link field empty, and attempting to follow the same institution twice from two different browser tabs; in each case the system either rejected the input with a clear message or, in the double-follow case, silently treated the second request as a no-op rather than creating a duplicate entry in the Followers collection. No case caused the application to crash or return an unhandled error during the testing window, which was taken as a reasonable bar of stability for a prototype at this stage rather than as a claim of production-level robustness.

IX. RESULTS AND DISCUSSION

The prototype was built and tested through its core user journeys: account creation, login, browsing the event feed, and registering interest in an event. Fig. 3 shows the registration screen, where a student fills in their name, institutional email, phone number, college, department, and year of passing before

creating an account; the same screen offers a faculty tab for institutions to register instead.



Fig. 3. Registration screen with role selection for Student and Faculty accounts.

Fig. 4 shows the resulting event feed, where events collected from several colleges are listed together with their host institution, date, and a direct registration link, and can be filtered by category such as tech fest, cultural fest, sports meet, workshops, or networking.



Fig. 4. Event discovery feed showing events aggregated across colleges with category filters.

During testing, students were able to move from opening the site to finding and registering for a relevant event in noticeably fewer steps than searching across separate college pages would take, since every event on the feed already carries its host, date, and category. Institutions were able to publish and edit event details directly, without depending on a third party to relay the information, and the follow mechanism meant that a student who had once followed a college did not need to revisit that college's page to learn about its next event. These outcomes support the central claim of the paper: that a shared, structured platform reduces the effort a student spends discovering events and reduces the effort an institution spends publicizing them, compared to the scattered, informal channels the system was built to replace.

Fig. 6 shows the student home dashboard that greets a user after login, bringing together navigation, a personalized summary strip, quick-access shortcuts, and a spotlight of featured institutions on a single scrollable page rather than spreading these across separate views.

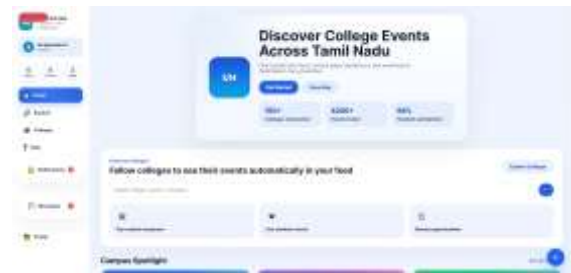


Fig. 6. Student home dashboard with sidebar navigation, activity summary, and campus spotlight.

During testing, students were able to move from opening the site to finding and registering for a relevant event in noticeably fewer steps than searching across separate college pages would take, since every event on the feed already carries its host, date, and category. Institutions were able to publish and edit event details directly, without depending on a third party to relay the information, and the follow mechanism meant that a student who had once followed a college did not need to revisit that college's page to learn about its next event. These outcomes support the central claim of the paper: that a shared, structured platform reduces the effort a student spends discovering events and reduces the effort an institution spends publicizing them, compared to the scattered, informal channels the system was built to replace.

Feedback gathered informally from the classmates who tried the prototype was broadly positive about the clarity of the event feed and the category filters, and the most common suggestion was to add a way to search by keyword in addition to category, which is folded into the future work discussed in Section XIII.

A secondary observation from this informal round of feedback concerned the registration screen shown in Fig. 3: several testers initially expected the Student/Faculty toggle to be a dropdown rather than two side-by-side buttons, which suggests the affordance could be made more visually distinct in a later iteration, even though it did not prevent anyone from completing registration successfully. This kind of small, low-stakes usability friction is exactly what informal testing with peers is meant to surface before a wider rollout, and is noted here for completeness rather than as a defect that blocked any of the test cases in Table IV.

Performance Observations

Under the conditions exercised during testing — a single development server, sample data covering several dozen events across a handful of colleges, and fewer than twenty concurrent

test users — page loads for the login screen, event feed, and student dashboard all completed comfortably within the two-second target set in Section IV-B, with the event feed's category filter updating the visible list instantly since the filtering was performed client-side on data already fetched. These figures are indicative rather than a rigorous load-testing exercise; they were adequate to confirm that the architecture does not introduce an obvious bottleneck at small scale, but a genuine multi-institution deployment, as discussed in Section XI, would need proper load testing before the numbers here could be treated as representative of production behavior.

X. COMPARATIVE DISCUSSION

Returning to the systems surveyed in Section II and summarized in Table I, the working prototype delivers on the two capabilities found to be largely missing from those systems: a multi-institution publishing model, in which any registered college can add its own events into a shared feed rather than the feed belonging to a single organizer, and a follow-based notification mechanism that pushes new events to interested students without requiring a repeat visit. Where FestFlow [1] demonstrates that role-based access works well within a single institution, UniEvent Hub extends the same principle across many institutions at once, and where the social-media-marketing study [2] shows that visibility drives turnout, the follow mechanism here is an attempt to manufacture that visibility deliberately rather than depend on the reach of an individual post. The trade-off is added complexity — supporting many independent publishers and a subscription graph between them is a harder engineering problem than a single-organizer CRUD application — which is part of why a document-oriented database was chosen over a rigid relational one for this project.

XI. LIMITATIONS

The prototype has been tested with sample data modeled on real regional colleges and events rather than with live traffic from many independent institutions, so its behavior under genuine multi-institution load — hundreds of colleges posting concurrently, thousands of students following and being notified — has not yet been observed directly. The notification fan-out described in Section VI runs synchronously within the request that posts an event, which is adequate at the scale tested but would need to move to an asynchronous queue if an

institution's follower count grew very large. The current build also does not yet include the payment handling, mobile application, or recommendation features described as future work, and event moderation currently depends on an administrator reviewing content manually rather than automated filtering. Finally, as noted in Section IX, the platform's value is inherently network-dependent: it is only as useful as the number of colleges that choose to adopt and actively post to it, a constraint common to any multi-sided platform of this kind.

XII. SOCIETAL AND EDUCATIONAL IMPACT

Beyond its immediate technical goals, UniEvent Hub speaks to a broader pattern in how information moves — or fails to move — through student communities. Access to an inter-college event is not evenly distributed today: a student whose social circle happens to include people from well-connected colleges hears about far more opportunities than an equally capable student whose circle does not, purely because of how the information travels. A centralized, subscription-based platform reduces this dependency on social proximity, giving every registered student the same starting visibility into what is available, regardless of which friends they happen to have. For smaller or less prominent colleges, the platform also offers a low-cost way to reach students beyond their own campus without needing a dedicated marketing effort or a large social media following, which is consistent with the observation from Namratha et al. [2] that visibility, not event quality alone, is often what limits turnout.

From an educational standpoint, building UniEvent Hub was itself a significant part of the learning outcome for the team. The project required moving between front-end design, back-end logic, and database modeling within a single cohesive system, rather than practicing each skill in isolation, and required the kind of role-based thinking — who is allowed to do what, and what happens when two different users interact with the same piece of data — that is common in real production systems but rarely emphasized in smaller course assignments. Designing the follow-and-notification feature in particular required reasoning about a many-to-many relationship (many students following many institutions) and its performance implications, which is a step up in complexity

from the more common one-to-many patterns found in the simpler systems surveyed in Section II.

XIII. CONCLUSION AND FUTURE WORK

UniEvent Hub was built to solve a problem most students recognize immediately: finding out about an inter-college event is still largely a matter of luck, depending on which social media page one happens to follow or which friend happens to mention it. By giving colleges a direct way to publish events and giving students a direct way to browse and subscribe to them, the platform replaces that informal chain with a structured one. The working prototype, covering registration, role-based login, event posting, browsing, and a follow-based notification path, demonstrates that the core idea is technically workable with a straightforward stack of HTML5, CSS3, JavaScript, and MongoDB.

Several extensions are planned. A dedicated mobile application would put event alerts where students are most likely to see them promptly. Email and SMS notifications would reach students who do not have the web app open at the moment an event is posted. An AI-based recommendation layer could rank events by relevance to a student's department and past interests rather than showing every event to every student equally. Built-in registration and payment handling would let a student complete sign-up for a paid event without leaving the platform, and an analytics dashboard would let institutions see how many students viewed, followed, or registered for their events, helping them plan future outreach more effectively. On the engineering side, moving the notification fan-out to an asynchronous queue and introducing keyword search alongside category filters — the single most common request from early testers — are the two changes planned first, since both build directly on the architecture already in place rather than requiring a redesign.

More broadly, the value of a platform like UniEvent Hub grows with the number of colleges that adopt it, so onboarding additional institutions is as much a part of the project's future work as any single technical feature. A small pilot rollout to a handful of colleges in the surrounding region, followed by a review of how students and staff actually use the follow and notification features in practice, would be the natural next step before investing further in the features listed above.

REFERENCES

- [1] B. N. Shivarkar, O. S. Mulmule, R. S. Marode, P. A. Chakravarti, S. S. Dhage, and A. S. Joshi, "FestFlow: A centralized college event management system," *Int. J. Eng. Development and Research (IJEDR)*, 2026.
- [2] Namratha J., A. Thiruchelvi, and Natasha J., "Event management and social media marketing," *Int. J. Advance Research, Ideas and Innovations in Technology (IJARIIT)*.
- [3] D. V. S. Kamal, "Event management system," B.Tech Mini Project Report.
- [4] "Event management system," *Int. Res. J. Eng. Technology (IRJET)*, vol. 7, no. 6.
- [5] "Event management system," ResearchGate, Publication no. 358045975.
- [6] "Event management system project report," *Int. J. Creative Research Thoughts (IJCRT)*, Paper ID IJCRT2103420.
- [7] "Event management system project report," ResearchGate, Publication no. 380897786.
- [8] Mozilla Developer Network, HTML/CSS/JavaScript documentation. [Online]. Available: <https://developer.mozilla.org>
- [9] W3Schools, Web development reference. [Online]. Available: <https://www.w3schools.com>