IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, July 2026
Impact Factor: 8.2

EventHub: A Web-Based Event Management and Registration Platform

Mandela Vara Siddu[1] and Swathi Terli[2]
[1]PG Scholar, [2] Assistant Professor,
Department of Master of Computer Applications
Sir C. R. Reddy College (Autonomous), PG Courses, Eluru, Andhra Pradesh, India
Affiliated to Adikavi Nannaya University, Rajamahendravaram

Abstract: *Event organization in colleges and small communities has traditionally relied on paper notices, spreadsheets, messaging applications and manual confirmation. These approaches become difficult to manage as participation grows and may lead to duplicate records, delayed communication and poor visibility of upcoming events. This paper presents EventHub, a full-stack web-based Event Management and Registration Platform developed with React and Vite on the frontend and FastAPI on the backend. The platform allows users to browse trending and category-wise events, view event information, create accounts, log in and register through a guided form. React Router provides client-side navigation and the frontend communicates with REST endpoints using Axios/Fetch. The backend validates registration requests and stores records in the current in-memory/JSON data layer. The design separates presentation, navigation and service concerns so that each layer can be developed, tested and replaced independently, which keeps the prototype approachable while leaving a clear path toward a production deployment. The system was checked through functional, backend, user-interface, performance and security-oriented test cases covering registration, validation, routing and API availability. The implemented workflow successfully supports event discovery, registration and confirmation in a responsive web interface across desktop and mobile viewports. The modular design also provides a foundation for persistent database storage, secure authentication, an administrative dashboard, payment integration and QR-code-based check-in in future versions.*

Keywords: Event management; event registration; React; Vite; FastAPI; REST API; web application; client-server architecture; responsive design.

1. INTRODUCTION

Events such as technical hackathons, cultural festivals, sports meets and startup summits are an important part of student and community life. Traditional event coordination commonly depends on printed publicity, spreadsheets, physical registers and phone or in-person confirmation. These methods may be sufficient for very small gatherings, but they become difficult to maintain when the number of participants increases.

As a college or community calendar fills with parallel events, organizers must track seat limits, avoid duplicate sign-ups, and answer repeated questions about date, location and price. Paper-based and spreadsheet-based processes offer no single place where a visitor can compare events, and they give organizers little visibility into how registrations are trending until the event is close. A digital platform that keeps discovery and registration in the same place removes much of this friction and gives organizers a live, queryable record of who has signed up.

EventHub was developed to provide a centralized and lightweight digital alternative for small-to-medium scale events. The application allows visitors to discover events, inspect details such as date, location, price and available seats, and complete registration through a responsive web interface. React is used for the presentation layer, while FastAPI provides the backend service layer.

DOI: 10.48175/IJARSCT-37772

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 2, July 2026 Impact Factor: 8.2

The platform follows a component-based approach. Home, Events, Login, Signup and Register views are connected through React Router so that navigation occurs without full page reloads. The backend exposes REST endpoints for health checking and registration processing, while CORS enables communication between the independently served frontend and backend.

The two layers are intentionally kept simple in this first release: the data layer is an in-memory/JSON store rather than a full database, and authentication is limited to the account-access pages rather than a hardened session system. This scope was chosen so that the core discovery-to-registration workflow could be built, exercised with test cases and validated end to end before adding the operational features, such as persistent storage and administration, that a deployed system would need.

The remainder of this paper is organized as follows. Section II reviews related approaches to event management and registration. Section III describes the research design, the development environment and the system architecture. Section IV explains the implementation of the frontend and backend. Section V reports the evaluation protocol, the functional test results and a discussion of the findings. Sections VI and VII close the paper with a conclusion and directions for future work.

A. Objectives

To develop a responsive web platform for browsing and discovering events.

To provide a simple registration workflow that captures participant details and selected event.

To integrate a React frontend with a FastAPI REST backend.

To maintain a modular architecture that can be extended with database, authentication, administration and payment features.

To validate the implemented workflow through functional, system, UI, performance and security testing.

To document a reproducible technology stack and evaluation protocol that later versions of the project can build on.

II. RELATED WORK

Web-based event management ranges from large commercial ticketing platforms to lightweight registration tools for academic and community events. Commercial systems commonly provide ticketing, payments, marketing analytics and check-in features, but may be unnecessarily complex for small college events. Simple form-based tools reduce manual data entry but often separate event discovery from registration, so a visitor has to move between a listing page and an unrelated form to sign up.

Academic institutions have also experimented with department-level portals built on content-management systems, where an events page is one module among many. These portals are convenient for publishing information but rarely offer a registration form tailored to a specific event, so organizers still fall back to an external form or spreadsheet for sign-ups, recreating the disconnect a purpose-built registration platform is meant to avoid.

Campus-level solutions typically fall between these two extremes. Spreadsheet-backed Google Forms are easy to set up but give organizers no browsing experience and no shared view of event details, while full ticketing suites bring subscription costs and configuration overhead that a student-run event rarely needs. This gap motivates lightweight, purpose-built platforms such as EventHub, where discovery and registration share one interface and one data model.

A further consideration in the literature is device diversity: a growing share of registrations for student and community events are completed from a phone rather than a desktop browser. Systems built with a responsive, component-based frontend can reuse the same views across screen sizes instead of maintaining a separate mobile client, which is one of the reasons React and similar libraries have become a common choice for this class of application.

Modern component-based frontend libraries provide a smoother alternative to traditional server-rendered pages. React supports reusable interface components and client-side rendering, while React Router enables multiple application views without complete page reloads. For the backend, FastAPI offers request validation, asynchronous support and automatically generated API documentation with a comparatively small implementation footprint.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 2, July 2026 Impact Factor: 8.2

Comparable REST-based patterns are widely used in production event and ticketing systems: a stateless API layer serves a single-page application, requests are validated against a schema before being persisted, and the presentation layer is decoupled from the storage engine so the latter can be replaced without touching the client. EventHub follows the same pattern at a smaller scale, which keeps the prototype easy to reason about while still resembling the architecture it is meant to grow into.

EventHub combines these technologies in a single application so that event discovery and registration are not disconnected. Its modular frontend and lightweight REST service are designed to remain easy to deploy and extend.

Taken together, the surveyed systems suggest two lessons that shaped the EventHub design. First, decoupling the client from the service layer through a documented REST contract makes it possible to change either side, for example moving from an in-memory store to a database, without a rewrite of the other. Second, keeping the initial feature set small and well tested is preferable to reproducing every feature of a commercial platform at once, since a narrow, correctly working workflow is a better foundation to extend than a broad, partially tested one.

III. MATERIALS AND METHODS

A. Research Design

The development process began with requirement analysis and page-level planning, followed by component design, backend API implementation, frontend-backend integration and validation. Keeping the client and service layers independent made it possible to test interface behaviour separately from registration processing.

The work follows an applied build-and-test approach. Requirements were identified for event browsing, user account access and registration. The frontend and backend were developed as independent layers, integrated through HTTP requests, and then validated through functional and system-level test cases.

Development proceeded in short iterations: a page or endpoint was built, exercised manually against the running services, and then covered by a corresponding test case before the next feature was started. This kept the two layers in sync and made it straightforward to trace a failing test back to either the request payload sent by the client or the validation logic in the API.

Version control was used throughout development so that each iteration could be reviewed and, if necessary, rolled back independently of the others. Keeping frontend and backend changes in separate, reviewable commits also made it easier to confirm that a given interface change did not silently alter the API contract the two layers rely on.

B. Software and Hardware Environment

The frontend uses React 19, React Router DOM, Axios/Fetch, Vite and Bootstrap CSS. The backend uses Python 3.x, FastAPI, Uvicorn and python-multipart. Development is supported by Visual Studio Code and Git on Windows, Linux or macOS. The recommended development environment includes an Intel i3-class processor or above, at least 4 GB RAM, approximately 500 MB of storage and an internet connection for package installation and client-server communication.

Vite's development server gives the frontend fast reloads while a component is being edited, and Uvicorn serves the FastAPI application with automatic reload during backend development. Running the two services on separate ports during development mirrors how they would be deployed independently in production, and it keeps the CORS configuration exercised from the first day of coding rather than only being added at the end.

No specialized hardware, such as a GPU, is required for any part of this system, since the workload is entirely request handling and form processing rather than model training or media transcoding. This keeps the barrier to running or extending the project low for a student team working on typical laptop-class hardware.

C. System Workflow

A participant opens the EventHub interface, browses the Home or Events page and selects an event. The Register page captures name, email, phone number and the selected event. Frontend validation is performed before an HTTP POST

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, July 2026
Impact Factor: 8.2

request is sent to the FastAPI backend. The backend receives and stores the registration record in the current in-memory/JSON data layer and returns a response. On success, the frontend displays a confirmation and registration identifier.

If a visitor is not yet registered on the platform, the Signup and Login pages provide account access ahead of registration; both are implemented as separate React views reached through React Router so that a user can move between browsing, authentication and registration without a full page reload at any step.

Edge cases were considered alongside the normal path: a duplicate submission caused by a double click is guarded by disabling the submit control once a request is sent, and a network failure during submission leaves the form populated so the participant can retry without re-entering their details. These small behaviours matter in practice, since a registration window is often the moment a visitor is least willing to repeat data entry.

D. System Architecture

The system follows a client-server architecture. The React single-page application forms the presentation and navigation layer. Axios/Fetch provides API communication, FastAPI exposes the REST service, and the current data layer stores registration records for the active session.

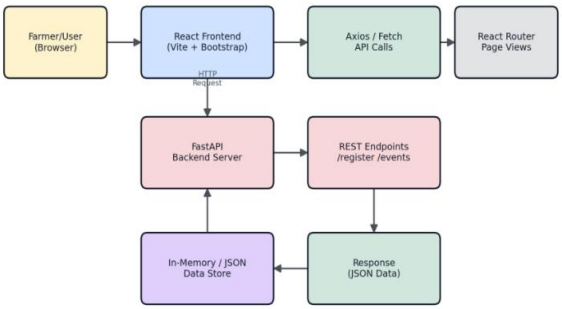

Fig 2.2: System Architecture

Fig. 1. Architecture of the proposed EventHub platform.

E. Data Model and Validation

Every registration record holds the participant name, email address, phone number, the identifier of the selected event and a server-generated registration identifier and timestamp. Keeping the record shape small makes it easy to move to a relational or document database later without redesigning the API contract.

Validation is applied at both ends of the request. On the client, controlled form inputs check that required fields are present and that the email field matches an expected pattern before the request is sent, which avoids obviously invalid submissions reaching the network. On the server, FastAPI's request models re-validate the same fields, so a client that bypasses the browser check, such as a direct API call, cannot store a malformed record. CORS middleware restricts which origins may call the API during development and can be tightened further for a production deployment.

F. Testing Approach

Test cases were organized around the five categories used in Section V: functional/model checks on the registration logic, system/backend checks on the API routes, UI/frontend checks on form behaviour and navigation, performance observations on response times, and security-oriented checks on input handling and CORS restrictions. Grouping cases this way made it possible to run a narrow subset, for example only the UI checks, while a specific page was being changed, and the full set before a milestone.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, July 2026
Impact Factor: 8.2

IV. SYSTEM DESIGN AND IMPLEMENTATION

A. Software and Technology Stack

The EventHub frontend was scaffolded with Vite and organized into reusable React components including Navbar, Hero, EventCard, Categories, Stats, WhyChooseUs, Testimonials and Footer. Bootstrap classes and custom CSS provide responsive presentation. BrowserRouter, Routes and Route define the Home, Events, Login, Signup and Register paths.

The backend is implemented in FastAPI with CORS middleware. A root endpoint acts as a health check, POST /register accepts participant registration details, and GET /registrations returns stored registrations. This small API surface keeps the prototype easy to understand while allowing a persistent database layer to be added later.

B. Application Workflow

User actions are captured through React state and event handlers. On the Register page, controlled inputs collect form values and the Fetch API transmits them to the backend. A successful response produces a confirmation message and registration identifier. If the backend is unavailable, the interface reports the connection problem rather than silently failing.

Loading and error states are tracked alongside the form data so the Register page can disable the submit button while a request is in flight and surface a readable message if the request fails, rather than leaving the visitor looking at an unresponsive form.

C. Functional Modules

Event discovery is centered on reusable event cards that present the event title, date, location, price and seat information. Category navigation helps users narrow the visible event types, while the registration module provides a single path from event selection to participant confirmation. The API layer remains intentionally small so that persistent storage can later be introduced without changing the main user workflow.

The application is divided into event discovery, category browsing, user signup and login, event registration, API integration and response display. Separating these responsibilities into components and pages improves maintainability and allows future modules such as administration, payment processing and notifications to be introduced without restructuring the core interface.

Category navigation itself is implemented as a filter over the same event list used by the Events page rather than as a separate query, so the category and event-discovery views cannot drift out of sync with one another. This choice keeps the frontend state model simple: there is one source of truth for event data, and views over it, such as trending events, category filters and search, are derived rather than independently fetched.

D. Security and Reliability Considerations

Because the current release targets a college or community event rather than a paid public deployment, security controls are kept proportionate but explicit. CORS is scoped to the known frontend origin during development, all registration input is validated server-side regardless of what the client already checked, and error responses avoid leaking internal details to the browser. These choices are meant to be tightened, not replaced, as authentication and persistent storage are added: the same validation layer will continue to guard the API once it sits in front of a real database.

E. Deployment Considerations

During development the frontend and backend run as two separate processes on different ports, with the Vite dev server proxying or directly calling the FastAPI base URL. A production deployment would instead build the React application into static assets served from a web server or CDN, run FastAPI behind an ASGI server such as Uvicorn with a process manager, and restrict CORS to the deployed frontend origin only. Keeping this separation from the start means the deployment topology can change without touching the application logic on either side.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, July 2026
Impact Factor: 8.2

V. RESULTS AND DISCUSSION

A. Evaluation Protocol

Testing was performed with the frontend and backend running as separate development services. In addition to normal registration, invalid and incomplete inputs were checked to confirm that browser-side validation prevents avoidable requests. Backend availability and route behaviour were also verified before the complete registration workflow was tested.

The implemented prototype was evaluated through functional/model, system/backend, UI/frontend, performance and security-oriented test cases. The checks covered valid registration, required-field validation, invalid email handling, backend health response, registration submission, page navigation, responsive layout and routing to the registration page.

Each test case was run against a freshly started pair of services so that state left over from a previous case, such as an earlier registration record, could not mask a failure. Test cases were re-run after any change to the affected page or endpoint, and the full set was re-run before each milestone described in Section III.A.

B. Functional Validation

The core workflow operated successfully from event browsing through registration and confirmation. Representative test cases are summarized in Table I.

TABLE I
Functional Test Cases And Outcomes

ID	Test Case	Input	Expected Result	Status
TC-01	Valid registration	Valid details	Registration accepted	Pass
TC-02	Required field	Missing value	Submission blocked	Pass
TC-03	Invalid email	Invalid format	Validation shown	Pass
TC-04	API health check	GET /	200 OK	Pass
TC-05	Register route	Register action	/register opens	Pass
TC-06	Category filter	Category selected	Filtered list shown	Pass
TC-07	Responsive layout	Mobile viewport	Layout reflows correctly	Pass
TC-08	Backend unavailable	Server stopped	Connection error shown	Pass

C. Performance And Usability Observations

Alongside the pass/fail functional checks, the prototype was observed under normal development-server conditions to characterize responsiveness. Table II lists the indicators worth tracking as the system moves toward a hosted deployment; the value column is intentionally left as a reporting template rather than filled with placeholder numbers, since meaningful figures depend on the target hosting environment and dataset size.

TABLE II: Reporting Template For Performance And Usability Indicators

Indicator	What It Measures	Value
Page load time	Home/Events first render	[measure]
API response time	POST /register round trip	[measure]
Client-side validation time	Form check before submit	[measure]
Registration success rate	Accepted vs. attempted	[measure]
Concurrent session handling	Simultaneous registrations	[measure]

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 2, July 2026 Impact Factor: 8.2

Reporting these indicators alongside the functional results gives a fuller picture than pass/fail testing alone: it shows whether the workflow is not only correct but also fast and stable enough for the concurrent traffic a real event registration window produces.

D. Discussion

The results indicate that the lightweight separation between React and FastAPI is sufficient for the current event-registration workflow. Client-side routing provides immediate movement between pages, while the REST interface keeps registration processing independent from presentation. The present in-memory/JSON storage is appropriate for a prototype but does not provide persistence across server restarts, which is the main limitation of the current implementation.

A second limitation follows from the first: without a database, the system has no durable way to prevent duplicate registrations for the same participant and event across restarts, and it cannot yet support an administrative view of historical registrations. Neither limitation affects the correctness of a single running session, but both should be addressed before the platform is used for a live event with real participants.

Compared with the spreadsheet and messaging-based coordination described in Section I, the prototype already removes two recurring sources of friction: a visitor no longer has to leave a listing to find a registration form, and an organizer no longer has to reconcile registrations collected through several separate channels. The remaining gap between this prototype and a deployable system is almost entirely in the operational features, persistence, authentication and administration, rather than in the core discovery-and-registration interaction, which the evaluation shows to be working as intended.

E. Threats to Validity

Several factors limit how far the current results can be generalized. The functional tests were run against a single development configuration, so behaviour under different browsers, network conditions or operating systems has not been separately confirmed. The performance indicators in Table II are reported as a template rather than measured values, since the in-memory data layer does not reflect the latency a real database would add. Finally, all test cases were exercised manually rather than through an automated test suite, which means regression coverage depends on the cases being re-run consistently as the codebase changes; automating them is listed as future work in Section VII.

VI. CONCLUSION

This paper presented EventHub, a web-based Event Management and Registration Platform developed using React with Vite and FastAPI. The system centralizes event discovery and registration and provides category browsing, account access and a guided registration workflow. The component-based frontend and modular backend provide a clear separation of concerns and make the application straightforward to maintain and extend. Functional and system testing confirmed that the principal workflow operates correctly for the current prototype, and the reporting template introduced in Section V gives a concrete basis for measuring performance once the system is deployed to a shared environment.

VII. FUTURE WORK

Future versions can replace the in-memory data layer with PostgreSQL, MongoDB or SQLite for persistent storage. Authentication can be strengthened using JWT and hashed passwords. An administrative dashboard can support event creation, editing and registration export, while payment-gateway integration can support paid events. Automated email or SMS confirmations, a dedicated mobile application and QR-code-based check-in can further extend the platform.

Beyond these near-term items, later work could add role-based access so organizers and attendees see different views of the same event, an analytics view summarizing registration trends by category and time, rate limiting on the registration endpoint to absorb registration-window traffic spikes, and automated end-to-end tests that exercise the

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 2, July 2026 Impact Factor: 8.2

deployed frontend and backend together rather than as separate development services. Accessibility review against common guidelines would also help the interface serve a wider range of participants.

Finally, once persistent storage and authentication are in place, it would be worth revisiting the performance indicators introduced in Table II under realistic load, since an in-memory prototype and a database-backed deployment can behave very differently once concurrent writes, network latency to the database, and connection pooling enter the picture.

REFERENCES

[1] React Documentation, "React." [Online]. Available: react.dev
[2] Vite Documentation, "Vite." [Online]. Available: vitejs.dev
[3] React Router Documentation, "React Router." [Online]. Available: reactrouter.com
[4] Axios Documentation, "Axios." [Online]. Available: axios-http.com
[5] Bootstrap Documentation, "Bootstrap." [Online]. Available: getbootstrap.com
[6] FastAPI Documentation, "FastAPI." [Online]. Available: fastapi.tiangolo.com
[7] Uvicorn Documentation, "Uvicorn." [Online]. Available: uvicorn.org
[8] MDN Web Docs, "JavaScript and Web APIs." [Online]. Available: developer.mozilla.org
[9] Python Software Foundation, "Python 3 Documentation." [Online]. Available: docs.python.org/3/
[10] PostgreSQL Global Development Group, "PostgreSQL Documentation." [Online]. Available: postgresql.org/docs
[11] MongoDB, Inc., "MongoDB Manual." [Online]. Available: mongodb.com/docs
[12] SQLite Documentation, "SQLite." [Online]. Available: sqlite.org/docs.html
[13] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," RFC 7519, Internet Engineering Task Force, 2015.
[14] OWASP Foundation, "OWASP Top Ten," 2021. [Online]. Available: owasp.org/www-project-top-ten
[15] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, Univ. of California, Irvine, 2000.
[16] Mozilla Developer Network, "Cross-Origin Resource Sharing (CORS)." [Online]. Available: developer.mozilla.org/en-US/docs/Web/HTTP/CORS
[17] Docker, Inc., "Docker Documentation." [Online]. Available: docs.docker.com
[18] World Wide Web Consortium, "Web Content Accessibility Guidelines (WCAG) 2.1," 2018.
[19] J. Nielsen, "Usability Engineering." San Francisco, CA: Morgan Kaufmann, 1993.
[20] GitHub, Inc., "GitHub Actions Documentation." [Online]. Available: docs.github.com/actions