# Eventra: An Event Management System (EMS) for Ticketing, Attendance Verification, and Real-Time Coordination

Stuti Mehla, PhD
Department of Computer Science & Engineering, Panipat Institute of Engineering & Technology Panipat, Haryana, India

Sunil Dhull, PhD
Department of Mechanical Engineering, Panipat Institute of Engineering and Technology,

Anju Gandhi, PhD
Department of Computer Science & Engineering, Panipat Institute of Engineering & Technology Panipat, Haryana, India

Prapti Choudhary
Department of Computer Science & Engineering Panipat Institute of Engineering & Technology Samalkha, Panipat, Haryana, India

Shivam Kumar
Department of Computer Science & Engineering Panipat Institute of Engineering & Technology Samalkha, Panipat, Haryana, India

Sandip Yadav
Department of Computer Science & Engineering Panipat Institute of Engineering & Technology Samalkha, Panipat, Haryana, India

Venu Sharma
Department of Computer Science & Engineering Panipat Institute of Engineering & Technology Samalkha, Panipat, Haryana, India

Geeta Rani
Department of Computer Science & Engineering, Panipat Institute of Engineering & Technology, Panipat, Haryana, India

## ABSTRACT
This study presents Eventra, a web-based Event Management System (EMS) that consolidates event discovery, tiered ticketing, online payment, QR-verified attendance, and post-event reporting into a single console for three user roles — participant, organizer, and administrator. The system was built on the MERN stack (MongoDB, Express.js, React.js, Node.js) and secured with JSON Web Token (JWT) authentication and role-based access control. This paper reports the requirements analysis, domain research, system architecture, and implementation of the EMS, and presents the technologies selected, the interface design, and a developer-conducted verification of the core participant, organizer, and administrator journeys against usability and functional criteria. The system was found to satisfy its stated functional requirements end-to-end, with one notable design limitation — a non-atomic seat-reservation step — identified and documented for future correction.

## Keywords
Event Management System, MERN stack, role-based access control, QR code attendance, real-time web application, online payment integration, system architecture

## 1. INTRODUCTION
An Event Management System is used to keep track of all the activities associated with an event publishing accurate information, allocating a fixed number of seats across ticket tiers, collecting payment, issuing verifiable proof of purchase, checking attendees in without long queues or duplicate entries, and afterward producing a defensible account of revenue and turnout [1]. Several of these concerns operate concurrently, which makes coordination difficult when they are handled with disconnected tools. To keep track of all these activities, this study proposes Eventra, a web-based console that keeps event, booking, payment, ticket, and attendance records in one authoritative database rather than spread across a registration form, a separate payment link, a printed guest list, and a manually compiled spreadsheet.

An event may be thought of as a gathering to which participants are invited for a specific period a conference, a paid workshop, a college activity, or a community meet-up, among other occasions. An event management system can be thought of as a platform for organizers and administrators to administer events and a gateway for participants to discover and book [2]. The process of responsibly managing an event includes setting ticket tiers and capacity, collecting payment, verifying attendance at the door, and reviewing feedback and turnout after the event. This project is a web-based portal that aids the functioning of an event organizer as well as participants and platform administrators [3].

Eventra is a software project that performs the functions of an event manager by providing role-scoped consoles for three account types. Participants can register, discover published events through search and filters, book a free or paid ticket tier, pay through Stripe when required, and receive a QR-coded ticket. Organizers can create and manage their own events, review attendees, scan tickets at the door, and pull reports. Administrators approve organizer accounts and individual events before they go live, and oversee the platform as a whole. All information is saved in a shared database, and every booking is issued a unique ticket identifier; this removes the need for a customer to visit an office or exchange e-mails with an organizer before a booking is confirmed, and it gives organizers real-time visibility into how an event is filling up.

# 2. LITERATURE REVIEW

## 2.1 Analyzing the Existing System

In the manual process that Eventra replaces, a prospective attendee typically contacts an organizer directly, or fills in a generic form tool that is not connected to a payment page, to describe which event they want to attend, how many seats they need, and how they intend to pay. The organizer separately tracks who has paid, prints or compiles a guest list for the door, and reconciles revenue after the event by hand. Each hand-off between tools the form, the payment link, the printed list, the spreadsheet is a place where data can drift out of sync, for example a seat marked sold on the form but never reconciled against the payment link, or a printed list that does not reflect a last-minute cancellation. Eventra's proposed system instead routes booking, payment, ticketing, and check-in through one API and one database, so that a seat that has been booked, a payment that has settled, and a ticket that has been scanned are always the same authoritative record rather than three separate ones.

## 2.2 System Gathering Requirement

There are two types of system requirements that were gathered for this project. Functional requirements describe the coherent, non-redundant behaviours the system must expose to its three user roles: account registration and role-based login; event creation with an administrator approval workflow; seat-aware, tier-based booking; Stripe-backed payment and refunds; automatic QR and PDF ticket issuance; camera-based QR attendance scanning with duplicate-entry prevention; real-time seat-count and notification updates; star-rated feedback with organizer replies; and exportable revenue, attendance, booking, and feedback reports. Non-functional requirements define the quality constraints under which those functions must operate: passwords must be hashed and never stored in plain text; every protected route must verify a JSON Web Token and an explicit role before executing; seat counts must stay reconcilable with payment status even if a client disconnects mid-flow; and the interface must remain understandable to a first-time user without training, since, unlike an internal enterprise tool participants using Eventra are the general public [4, 5].

## 2.3 Domain Research

Three technical domains underpin Eventra's design. The first is token-based authentication: a JSON Web Token (JWT), as standardized in RFC 7519, is a signed, self-contained credential that lets a server verify a request's identity and role without a database lookup on every call, which is why Eventra pairs a short-lived access token with a longer-lived, httpOnly refresh cookie rather than server-side session storage [5]. The second is real-time web communication: unlike a client that must repeatedly poll an endpoint to notice a seat count has changed, a WebSocket connection (implemented here through Socket.IO) lets the server push a change to every interested client the moment it happens, at the cost of maintaining an open connection per active user [6]. The third is payment-card security: routing card entry through a hosted payment-provider field (Stripe Elements) rather than through Eventra's own form is what keeps raw card data out of Eventra's servers entirely, which substantially reduces the PCI-DSS compliance scope the system would otherwise carry. Together, these three domain choices — token-based auth, push-based real-time updates, and provider-hosted payment fields — are what let a small platform offer bank-grade session security, live seat visibility, and card payments without operating its own session store, message broker, or payment vault.

## 2.4 Similar Systems

As is the case with most systems built to serve a broad market, the closest points of comparison come from established commercial platforms rather than a single direct competitor. Three comparable systems were reviewed for this study.

Eventbrite. Eventbrite is a widely used self-service event registration and ticketing platform that lets an organizer publish an event page, sell tiered tickets, and collect payment without contacting a sales team. Its strength is breadth of reach and payment handling; organizers have comparatively little control over the booking, attendance, and reporting workflow beyond what the platform's own dashboard exposes, and there is no source-level access to extend it.

Meetup. Meetup centers on recurring community groups and RSVP-based (rather than paid-ticket-based) event attendance, with discovery driven by group membership and interest tags. It is well suited to informal, free, recurring gatherings but is not built around paid ticket tiers, seat capacity enforcement, or door-side QR verification, which are central to Eventra's scope.

Cvent. Cvent is an enterprise-oriented event and venue management suite aimed at large conferences and corporate events, offering venue sourcing, registration, and analytics as a paid enterprise product. It covers much more of the event-planning lifecycle (venue sourcing, budgeting) than Eventra attempts to, but at a cost and configuration overhead aimed at professional event-management teams rather than an individual organizer or a small institution.

Relative to these systems, Eventra's distinguishing characteristic is not any single feature but that it is a self-contained, source-available system combining role-gated event moderation, tiered ticket inventory, QR-verified attendance, and real-time seat propagation in one codebase that a small organization can run, inspect, and extend, rather than a hosted product whose internals are opaque and whose workflow cannot be adapted.

# 3. PROBLEM STATEMENT

Event, festival, and conference coordination is the process used to plan, manage, and organize a gathering, including determining budget and cost objectives and producing a post-event account of turnout and revenue. Currently, most small and mid-sized event organizers manage booking and planning in a traditional, manual manner, which requires a prospective attendee to contact the organizer directly and explain what they want to attend, how many seats they need, and when, which takes a significant amount of time and effort for both sides.

A related issue is that most existing event management web applications that do offer self-service booking do not offer tiered seat inventory with automatic enforcement, verifiable QR-based attendance, and organizer-facing real-time visibility together in one system an organizer is often left combining a registration tool with a separate payment link and a manually kept door list [7, 8]. Additionally, some of these existing tools are difficult for a first-time customer to use and provide no direct channel back to the organizer if the customer has a question mid-booking. Eventra has been created with these issues in mind: it allows a participant to browse, book, and pay for an event on a single console, gives the organizer a real-time, seat-accurate view of their event without leaving the same console, and verifies attendance at the door in a way that structurally rejects a duplicate scan rather than merely discouraging one.

# 4. AIMS AND OBJECTIVES

To assess, develop, and document a web-based Event Management System that enables organizers to publish and moderate events, participants to discover and book them, and administrators to oversee the platform, all from a single console, removing the need to coordinate an event across several disconnected tools.

To accomplish this purpose, the following objectives were identified and pursued: to analyze how events are currently coordinated without a unified system, and to compile the functional and non-functional requirements a replacement system must satisfy; to research the domain foundations — authentication, real-time communication, and payment security — needed to implement those requirements soundly; to design a role-based system architecture and data model spanning participant, organizer, and administrator concerns; to implement a working system covering authentication, event moderation, tiered booking, Stripe payment, QR/PDF ticketing, QR attendance scanning, real-time notifications, and reporting; and to verify the implemented system against its stated requirements and to document any limitations found during that verification.

# 5. RESEARCH METHOD

## 5.1 Technical Research

A technical study was performed to identify the critical development factors, technology requirements, and implementation issues for successful development of the proposed system. Researchers focused on the right set of technologies, working on their interoperability and making sure that the tools we chose can integrate seamlessly with each other to make sure the application is reliable, easily scalable, simple to maintain and high performing. The selection process has been a comparative study of various software frameworks, programming environments, database management systems and interface development technologies. The most important factors were ease of development, flexibility, performance, scalability, data management capabilities, community support, security and compatibility with other components of the system.

The entire app was written in JavaScript end to end, using the same approach to developing on the client and the server. That cuts down on the complexity of the technology and lets the various application components talk to each other smoothly. React.js was chosen for the front-end development. React.js is a component-based architecture, reusable, flexible and able to develop dynamic and interactive user interfaces. With React.js you can decompose the app UI into distinct reusable components. This process makes the development process more structured and easier to modify and maintain in the future. In addition, its virtual DOM mechanism improves the rendering performance by updating only the required parts of the UI when the application data changes.

The most important technology for server-side development was Express.js in the Node.js runtime environment. Node.js is a JavaScript runtime that runs on an asynchronous event-driven architecture. This can be especially useful for apps that might have multiple simultaneous user requests, database access and API calls. Express.js is a lightweight and flexible framework to build RESTful APIs, manage requests and responses, middleware, authentication and communication between front-end and database layers. Express. Express is a nice way to handle application logic and have modular and structured server-side architecture.

MongoDB is used as a database management system. MongoDB is a document database that has a flexible schema design, which means you can store structured and unstructured data in it. Organized information MongoDB stores the information in a BSON document, not in tables and relationships as in traditional relational databases. This means that data structures can have different attributes and nested object properties. Flexibility is a very useful feature for the proposed application, where some entities may have multi-level information and dynamic changing attributes. For example, an event may have different types of tickets with different prices, availability, booking information and other attributes specific to that event. For the right use cases, this type of information maps naturally into a flexible data structure in MongoDB's document model, reducing the need for complex relational mapping.

Mongoose is used as Object Data Modeling (ODM) library to make database processes easier and to provide an efficient way to interact between our application and MongoDB. MongoDB has schema definitions, data validation, structured querying, and working with the database as a model [9]. References and methods of population can be related to each other different data entities. This makes it easier to work with the data in a more consistent way when dealing with the database. This is especially true when dealing with complex application objects. Moreover, MongoDB is a perfect match with Node.js and Express.js, allowing to stick to a JavaScript-based approach to data management in the core layers of the development. This allows data objects to be manipulated and processed with minimum transformation between the application interface, server-facing logic and database operations [10].

The application UI was built with the help of the primary styling framework, Tailwind CSS. Tailwind CSS is a utility-first framework that provides ready-to-use CSS classes that you can apply directly to your HTML elements and React components [11]. This allows for more flexibility when creating responsive layouts, consistent typography, colour schemes, spacing, and interactive visual elements. It also makes creating reusable interfaces easier and reduces the need for multi-level custom CSS. Responsive Design Responsive framework with utility classes to control responsiveness at breakpoints. This means the application is detailed and responsive, and interactive for all screen sizes of desktop, tablet and mobile devices. Building a consistent and maintainable UI is easier with React.js.

Another factor taken into account in the choice of technology was the compatibility of the chosen tools and the possibility of future extensions of the system. So, React.js, Node.js, Express.js, MongoDB, and Mongoose are all coming together to form a unified development environment aka MERN tech stack [12, 13]. The architecture enables modular development, efficient data exchange, and easier maintenance and allows extending the application functionality without significant changes in the existing infrastructure. You have JavaScript all the way so there is less overhead when you switch between different languages and data representations when learning and developing.

To summarize, the selected technologies are giving a good trade off between flexibility in development, performance in application, adaptability in database, responsiveness in user interface and maintainability in long term. Together, the capabilities meet the development needs of the proposed system and provide a basis for scalability and future functional extensions. Table I shows the technologies and versions used in the implementation of the presented application. Table I.

Summary of selected choices

| Type | Name | Version |
|---|---|---|
| Front-end Framework | React.js | 18.3.1 |
| Back-end Framework | Express.js | 4.19.2 |
| Database / ODM | MongoDB / Mongoose | 8.5.1 |
| Real-time Layer | Socket.IO | 4.7.5 |
| Payment Gateway | Stripe SDK | 16.2.0 |
| State Management | Redux Toolkit | 2.2.6 |
| CSS Framework | Tailwind CSS | 3.4.6 |
| Build Tool | Vite | 5.3.4 |
| Runtime | Node.js | ≥18.x |

## 5.2 EMS Development

Eventra was constructed feature-by-feature as a working skeleton rather than a design-only specification: authentication, events, bookings, tickets, QR attendance, Stripe payments, Socket.IO notifications, and reports were each wired end-to-end against a real database before the next feature was layered on top, following a consistent request path of route → middleware → controller → service → model. This order was deliberate — authentication and role middleware had to exist before any role-gated feature could be meaningfully tested, and the booking/ticketing pipeline had to exist before attendance scanning had anything to scan. AI-flavoured features (recommendations, chatbot replies, sentiment analysis) were isolated behind a single service module and satisfied with deterministic heuristics carrying explicit TODO markers, so the surrounding product could be built and demonstrated without a live model dependency [14].

## 5.3 Requirements and Domain Analysis

Rather than collecting requirements from live user interviews, this project's requirements were compiled from two sources: a feature-by-feature analysis of comparable commercial platforms (Section II-D) to establish what a competitive booking-and-ticketing flow needs to support, and a direct analysis of the operational needs of an event's lifecycle — moderation before publication, tiered inventory, payment settlement, door verification, and post-event reporting — as the backbone of the functional requirements in Section II-B. This approach was chosen because it grounds each requirement in a concrete, checkable capability (a route, a schema field, a UI action) rather than in an unverifiable claim about user preference.

## 5.4 Design Validation

In place of a quantitative survey, design validation was carried out by tracing each core workflow directly against the implemented source: the booking, payment, and ticket-issuance pipeline; the QR attendance scan and its duplicate-rejection behaviour; and the role-gated approval chain from event creation to public listing. This tracing surfaced both confirmations and one substantive limitation. On the positive side, the attendance and feedback uniqueness constraints were confirmed to be enforced at the database layer (via unique indexes) rather than only in application code, which makes them robust to a bug in any single code path. On the limitation side, the booking flow was found to decrement seat inventory optimistically — reading the remaining seat count and writing back an updated value in two separate steps rather than one atomic operation — which is safe under the single-user testing performed here but would allow two participants racing for the last seat in a tier to both succeed under real concurrent traffic. This finding is carried forward into the system architecture discussion and is treated as a required fix, not a hypothetical one, before any production deployment at scale.

# 6. SYSTEM ARCHITECTURE

## 6.1 Administrator (Super Admin) Features

Using a registered e-mail and password, the administrator logs in and reaches an administrative dashboard. From there the administrator can review pending organizer accounts and pending events and either approve or reject each one before it becomes visible to the public; an event stays invisible to participants until both its organizer and the event itself have cleared this checkpoint. The administrator can manage all platform users, including deactivating an account, and can review audit logs and activity logs recording administrative actions and system events for later review. A system-wide analytics view aggregates bookings, revenue, and attendance across every organizer on the platform, not just one organizer's own events.

**Login.** The admin signs in with an e-mail and password; a JWT is issued and a role check restricts every subsequent admin route to the super_admin role specifically. Approve Organizers / Events. Each pending organizer account and each pending event is reviewed and either published or rejected; rejected items remain invisible to participants. Manage Users. The admin can view, search, and deactivate any account on the platform. Audit and Activity Logs, Administrative actions and platform events are recorded and browsable for later review. System Analytics, Platform-wide revenue, booking, and attendance figures are aggregated across all organizers. Logout. Ends the session and clears the refresh token.

## 6.2 Organizer Features

An approved organizer logs in to an organizer-scoped dashboard. The organizer can create a new event — title, description, banner image, category, venue, date/time, capacity, and one or more ticket tiers (free, paid, VIP, or early-bird) — and submit it for administrator approval. Once published, the organizer can view attendee lists, scan QR tickets at the door using a device camera, and pull revenue, attendance, booking, and feedback reports for their own events, exportable as PDF, Excel, or CSV. The organizer can also reply to participant feedback left on their events.

Create / Manage Events. Organizers define ticket tiers and capacity and submit the event for approval; a published event can later be edited or cancelled. View Bookings and Attendees. A live, seat-accurate view of who has booked each tier, updated in real time as new bookings arrive. Scan Attendance. QR scanning at the door by camera, with a manual ticket ID as backup and strict rejection of a second scan of the same ticket. Reports & Analysis: Reports on revenue, attendance, bookings, and feedback for the organizer's own events in PDF, Excel, or CSV. Reply to feedback. Each feedback item will be responded to with one response, visible to the participant who submitted it. Sign out. Ends session and clears refresh token.

## 6.3 Participant Features

Any visitor can browse published events; registering or logging in unlocks booking. The website dashboard displays the participant's bookings, tickets, and wishlist, and the top navigation offers Browse Events, Dashboard, and account actions.

Register / Login. A new participant registers with a name, e-mail, and password; the developer kept the form to the minimum fields needed (name, e-mail, password, and an "attend vs. host" role choice) to lower the barrier to signing up.

Browse and search events with full-text and filtered searches across published events, including pagination. Book a ticket. Select a tier and quantity; free tiers confirm instantly, and paid tiers proceed to Stripe checkout. Make Payment. Card entry through Stripe Elements: the participant never enters card details into an Eventra-hosted field. My tickets. Each booked seat is issued its own QR-coded, downloadable PDF ticket, shown with a VALID or USED status badge. Feedback and Wishlist. Participants can rate and comment on an event they attended and can save events of interest to a wishlist for later. Real-time Notifications. Booking confirmations, payment results, and seat-count changes are pushed to the participant live via Socket. IO, without a page refresh. Logout. Ends the session and clears the refresh token.

## 7. PROJECT PLAN

User Acceptance Testing (UAT), often referred to as beta testing or end-user testing, is the stage of software development where the target audience or a representative of the intended user base tests the program against real usage scenarios rather than a menu-driven test script. UAT is important because it helps confirm that a web-based system behaves correctly under actual usage patterns, and it surfaces flaws before the system reaches end users. For Eventra, the UAT criteria were framed around five dimensions common to interactive systems: Graphical User Interface (GUI), Ease of Use, Functionality, Consistency, and Performance. Because this study was carried out as a single-developer technical project without access to an external panel of test users, Section IX reports these criteria as a developer-conducted, scenario-based verification exercised against the running system and its source, rather than as an independent human-subjects usability study; this distinction is made explicit so the evidence is not overstated.

## 8. IMPLEMENTATION

The following includes mockups of the Eventra interface, reconstructed directly from the implemented React components and the project's Tailwind design tokens (brand violet #6D3FD1, amber accent #F59E0B), together with a brief discussion of the human–computer interaction (HCI) principles applied to each page.

### 8.1 Register Page

Fig. 1 shows the register page, where a new participant can enter a full name, e-mail, and password, and choose whether they intend to attend or host events, before clicking “Sign Up.” A user who already has an account can instead click “Have an account? Sign in” to reach the login page. The form was kept to three fields plus a single role choice to reduce the barrier between a first-time visitor and a completed registration, applying the “aesthetic and minimalist design” HCI principle: the page is free of components that do not serve the task of registering.

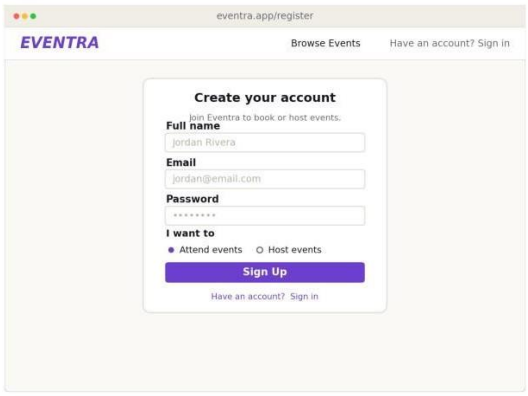


**Fig. 1 Register page**

Fig. 2 shows the login page, where a returning user enters an e-mail and password and clicks “Sign In” to reach their role-appropriate dashboard, or clicks “New here? Sign up” if they do not yet have an account. This page applies the “ease of use” HCI principle: the layout is identical in structure to the register page so a user who has just seen one immediately understands the other, and a single primary action (Sign In) is visually dominant.

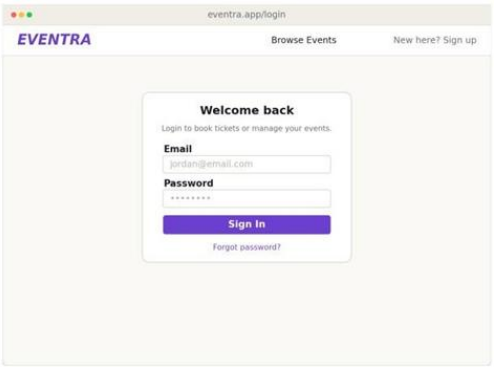


**Fig. 2 Login page**

### 8.3 Event Discovery Page

Fig. 3 shows the event discovery page, which is the home page for a logged-in or anonymous browsing participant. Each event is shown as a card with its category badge, title, date, venue, and starting price or a “FREE” label, so a participant can compare several events without opening any of them. This applies the “recognition rather than recall” HCI principle: the price, date, and venue a participant needs to decide whether to open an event are visible on the card itself, rather than requiring the participant to remember to check a details page.

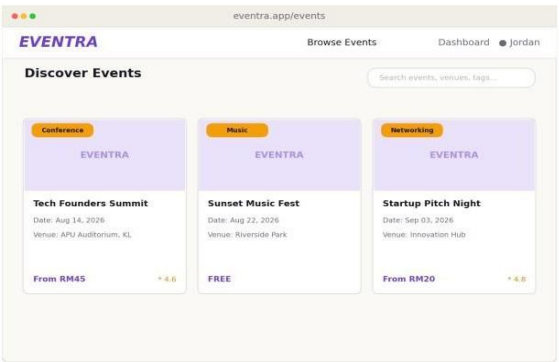


**Fig. 3. Event discovery page**

## 8.4 My Tickets Page

Fig. 4 shows the participant's My Tickets page. Each booked seat is rendered as its own ticket card carrying a unique ticket ID, a scannable QR code, and a VALID or USED status badge, with a Download PDF action. This page applies two HCI principles at once: "visibility of system status," since the VALID/USED badge tells the participant at a glance whether a ticket has already been redeemed at the door, without needing to ask an organizer; and "error prevention," since the same uniqueness constraint that turns a ticket USED at the door (Section V-D) is what stops that same ticket from being scanned a second time, so the badge shown here is never out of sync with what the door scanner will accept.

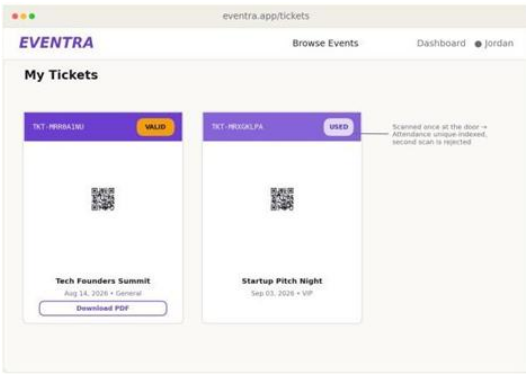


**Fig. 4. My Tickets page, showing QR-coded tickets and their check-in status**

# 9. DEVELOPER-CONDUCTED VERIFICATION

As noted in Section VII, the three scenarios below were exercised and rated by the developer against the running system and its source code, using the seeded participant, organizer, and administrator accounts; they are reported here in the same criteria format UAT commonly uses (GUI, Ease of Use, Functionality, Consistency, Performance) but should be read as an internal verification pass, not as independent end-user feedback.

Across the three scenarios, the core requirements set out in Section II-B were confirmed to work end-to-end, and the two structural safeguards inspected most closely — the unique-index rejection of a duplicate ticket scan and the dual organizer/event approval gate held in every trial. The one recurring caveat, noted consistently under Functionality, is the non-atomic seat-reservation step identified in Section V-D, which single-user verification of this kind is structurally unable to rule out under concurrent load; this is why it is carried forward as a required item, not an optional one, in the future work discussed in Section X.

# 10. CONCLUSION

This project has created a web-based Event Management System that can be used to book any kind of ticketed event, from a paid workshop to a free community meet-up. Eventra is developed so that anyone who wants to attend an event can register and book it directly, while the organizer manages the event, its capacity, and its attendees from the same console, and an administrator moderates what goes live. One of the system's central strengths is that a booking, its payment, and its ticket are always the same authoritative database record rather than three records kept in three different tools, which removes the reconciliation problem identified in Section II-A.

As discussed in the literature review, most existing systems either provide broad-reach ticketing without workflow customization (Eventbrite), RSVP-style attendance without paid tiers (Meetup), or enterprise-scale planning at a cost and complexity aimed at professional teams (Cvent). Eventra's contribution is a self-contained system combining role-gated moderation, tiered ticketing, QR-verified attendance, and real-time seat visibility that a small organization can run and adapt directly.

Critical development elements were addressed in this technical investigation, and comparative research served as the foundation for the project's technical judgments. At the same time, this study must acknowledge that the system has not been evaluated under real concurrent user load, and the design-validation pass in Section V-D surfaced a genuine limitation — optimistic, non-atomic seat reservation.

# 10. REFERENCES

[1] Rafati Niya, S., Bachmann, S., Brasser, C., Bucher, M., Spielmann, N., & Stiller, B. (2022). Deti: a decentralized ticketing management platform. *Journal of Network and Systems Management*, *30*(4), 62.

[2] "Smart College Event Management System Using MERN Stack," Int. J. Res. Appl. Sci. Eng. Technol. (IJRASET), vol. 11, no. 3, 2023.

[3] Gupta, N., Gupta, J., & Oza, P. (2026, June). Campus Event Scheduling System (CESS): An Automated Role-Based Workflow for Academic Event Management. In *2026 1st International Conference on Innovation & Growth in Next-Gen Intelligent Technology & Engineering (IGNITE)* (pp. 1-6). IEEE.

[4] Shabbir, A., Deraman, A., Hassan, M. B., Sarker, K. U., & Kamal, S. (2025). Prioritizing non-functional requirements and influencing factors for API quality framework: An industry approach. *International Journal of Advanced Computer Science and Applications*, *16*(9).

[5] Jones, M., Bradley, J., & Sakimura, N. (2015). Rfc 7519: Json web token (jwt).

[6] Liu, Q., & Sun, X. (2012). Research of web real-time communication based on web socket. *International Journal of Communications, Network and System Sciences*, *5*(12), 797-801.

[7] "QR Code Based Attendance System," Int. Res. J. Modernization Eng. Technol. Sci. (IRJMETS), 2023.

[8] "Enhancing Attendance Tracking Using Animated QR Codes: A Case Study," ResearchGate, 2023.

[9] Shwetha, H., Prajwal, D., & Sridharan, S. (2024, April). From MongoDB to React: Unleashing the Power of MERN in E-commerce. In *2024 International Conference on Emerging Technologies in Computer Science for Interdisciplinary Applications (ICETCS)* (pp. 1-6). IEEE.

[10] "The Synergy of Events and Technology: MERN Stack Solutions for Seamless Event Rentals," Int. J. Res. Appl. Sci. Eng. Technol. (IJRASET), vol. 12, no. 3, 2024.

[11] Aamir, Umesh, Sohel, Chitransh, Shrivastava, V., & Varshney, R. (2025, December). Building Responsive UI in Full Stack Applications Using Tailwind CSS and React. In *International Conference on Information and Communication Technology for Competitive Strategies* (pp. 171-178). Cham: Springer Nature Switzerland.

[12] "Event Dashboard using MERN Stack," GeeksforGeeks,

2025.

[13] TechLabs Aachen, "Eventara: Streamlining Event Management with the MERN Stack," Medium, Digital Shaper Program, winter term 2024/2025.

[14] MongoDB Inc., "Mongoose ODM Documentation," mongoosejs.com.