Evently: Event Management

Evently is a full-stack web application designed to help people create, discover, and manage events. The goal was to build a clean, fast Event management platform where event organizers can see exactly how their events are performing, while users get a smooth experience finding and booking tickets.

Tech Stack:

I chose these specific tools because they work well together for a fast, modern app:

- Frontend: React with Vite (for speed) and Tailwind CSS (for the "Glassmorphism" look).
- Icons: Lucide-React for those sharp, consistent UI elements.
- Backend: Node.js and Express. It's lightweight and handles many requests easily. CORS for cross communication between frontend and server.
- Database: SQLite. I used this because it's a "file-based" database—perfect for a project like this where we want reliability without the overhead of a massive server.
- Auth: Clerk. This handles all the login/security stuff so I could focus on the actual event features.
- Addons: nodemailer, sonner toast notification

Key Features

- **The Public Grid**

  Users can see a list of all upcoming events. I wrote the code to automatically hide events that have already finished, so the platform always looks fresh and updated, and can be able to discuss about other for a specific event.

- **The Manager Control Panel**

  This contains primary functionalities of the app. If we created an event, we get access to a special dashboard where we can:

  - Track Revenue: It calculates how much money we've made from different ticket tiers (Diamond, Gold, Silver) using direct database math.
  - Broadcast Emails: I built a custom email engine. If an organizer needs to update their guests, they can type one message, and it "dispatches" to everyone at once.
  - Live Cleanup: Managers can delete events if plans change. I made sure that if an event is deleted, all the booking records for that event are wiped too, so the database stays clean.

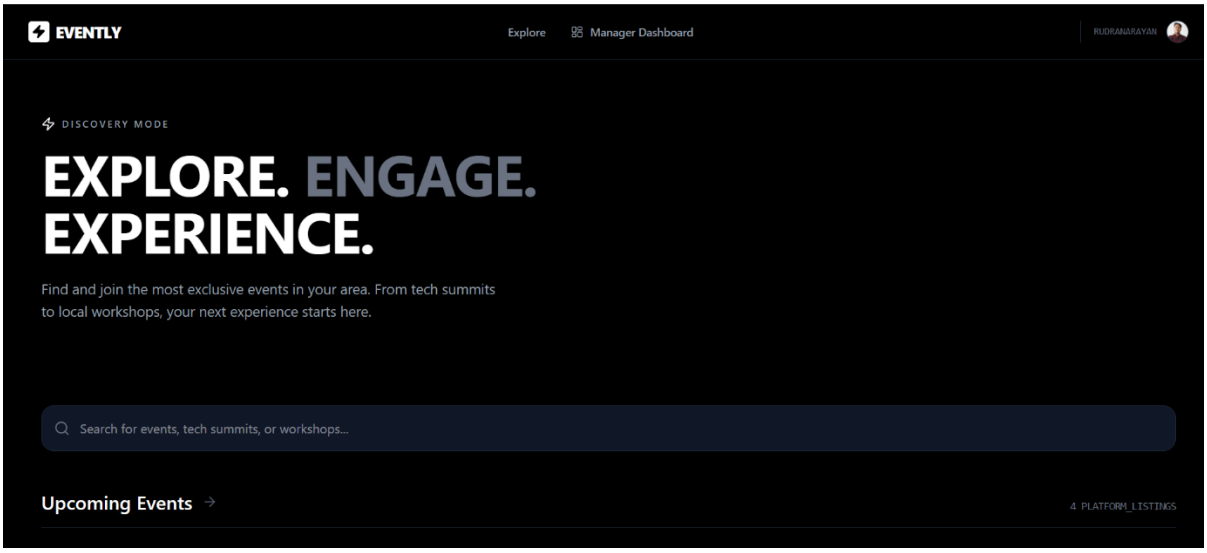
Technical Challenges I Solved

- Making a instant Broadcast using mail Initially, when sending broadcast emails, the screen would freeze while the server worked. I refactored the backend to handle the email sending in the background. This means the user gets a "Success" message immediately, while the emails finish sending on their own.
- I spent a lot of time on the SQL side. Instead of just pulling data and fixing it in React, I used advanced SQL queries to make sure the database does the heavy lifting. This keeps the frontend fast even if there are hundreds of events.

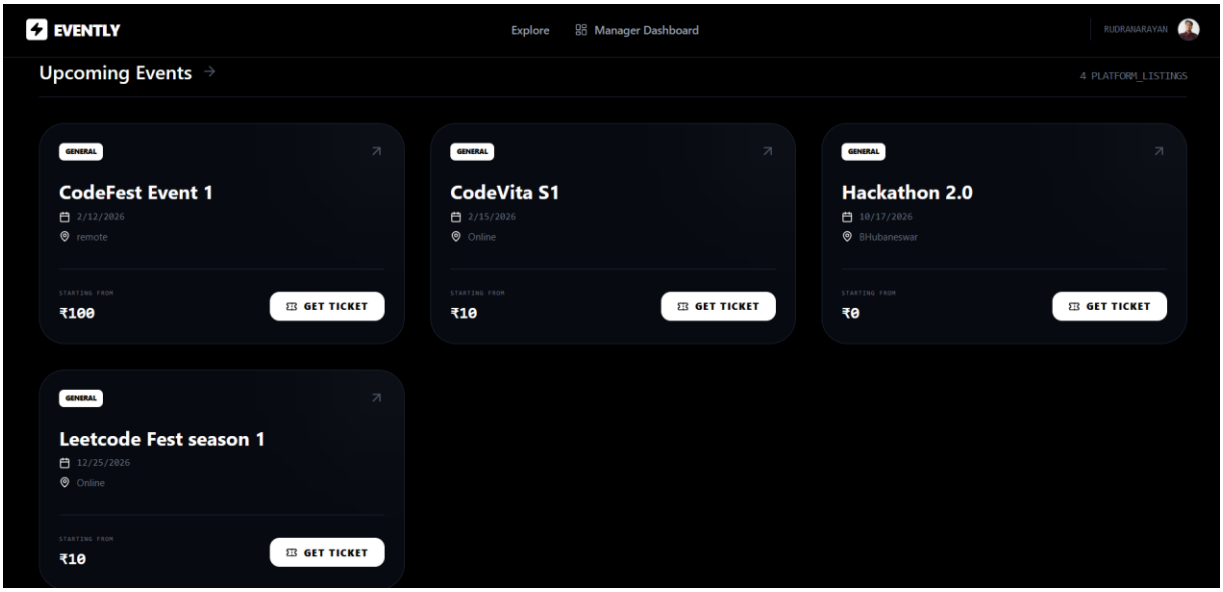
Detail explanation with screens:

1. **Welcome page (Home page): Welcome section with event grid:**

The homepage is designed for instant impact, featuring a bold, dark-themed welcome section that immediately hooks users with a clear call to action. Below this, a smart event grid automatically filters out past dates, ensuring the platform stays clean and relevant. Events are sorted chronologically, so the most urgent, upcoming opportunities always appear first in a sleek, modern layout. By combining a high-energy hero area with a real-time discovery grid, I've created a seamless experience where users can find and book the best events.

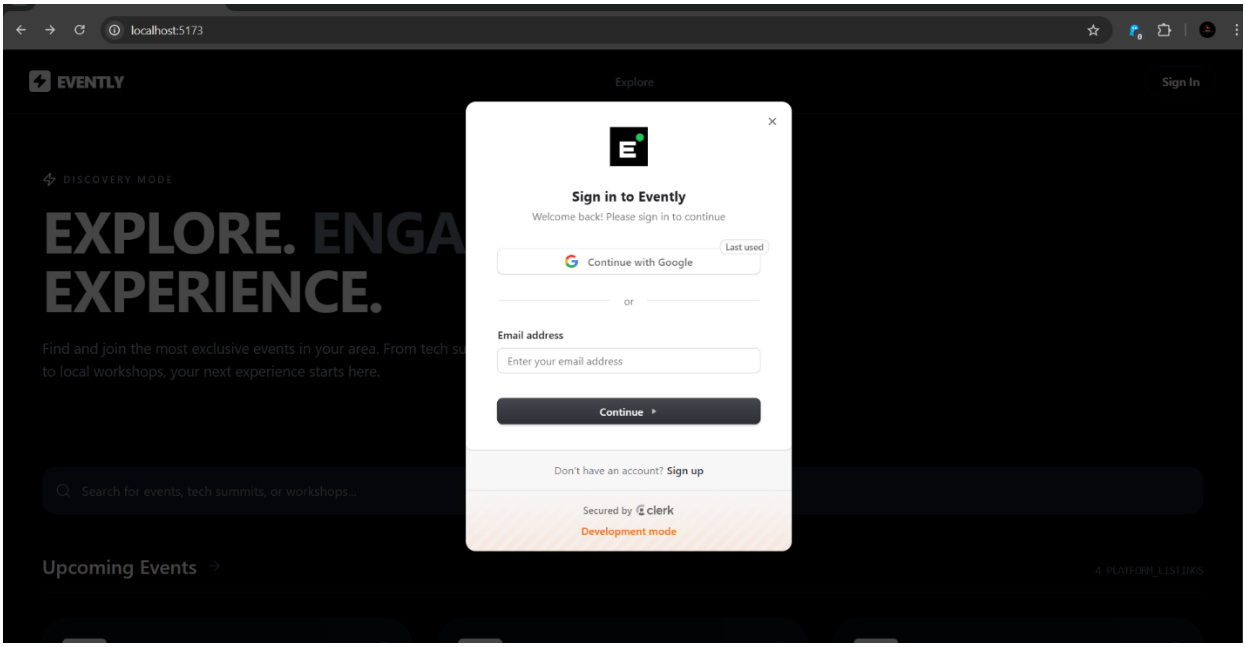


Shows Filtered upcoming events:

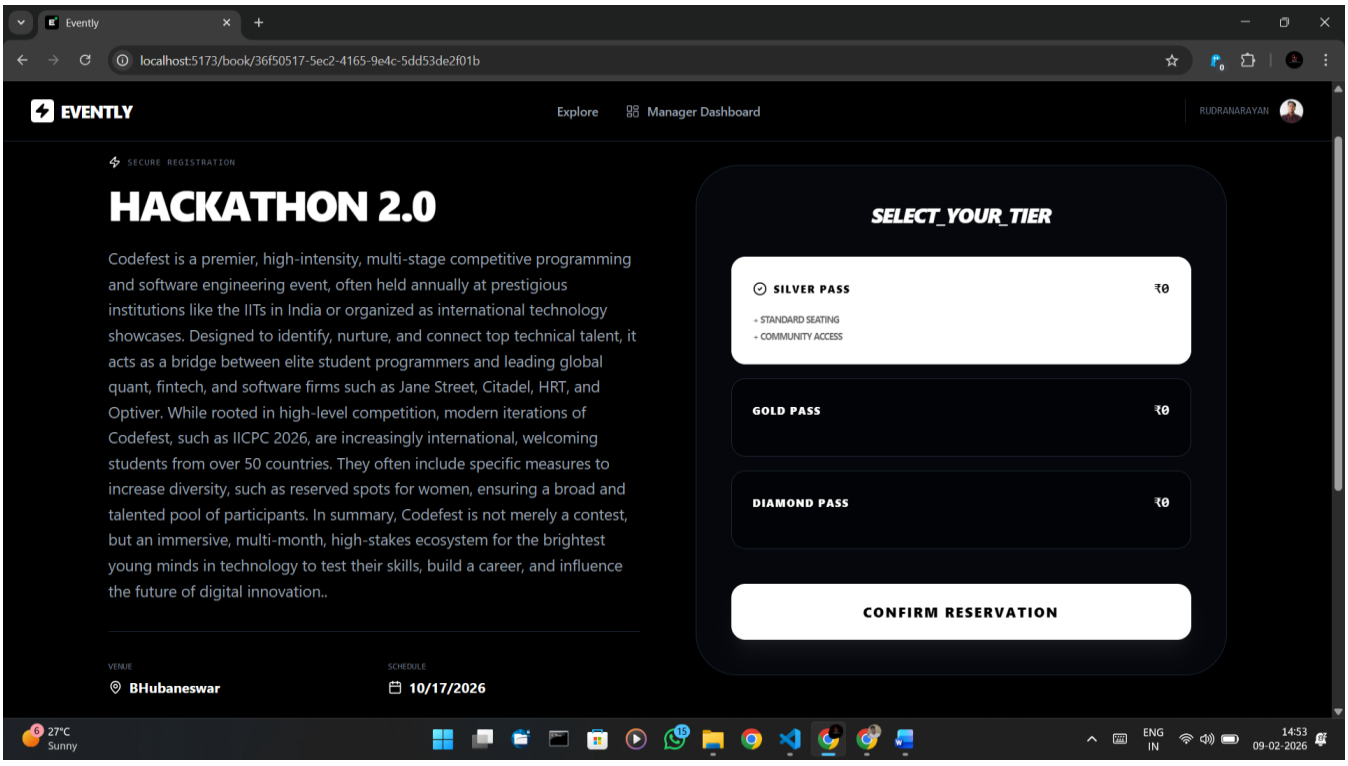


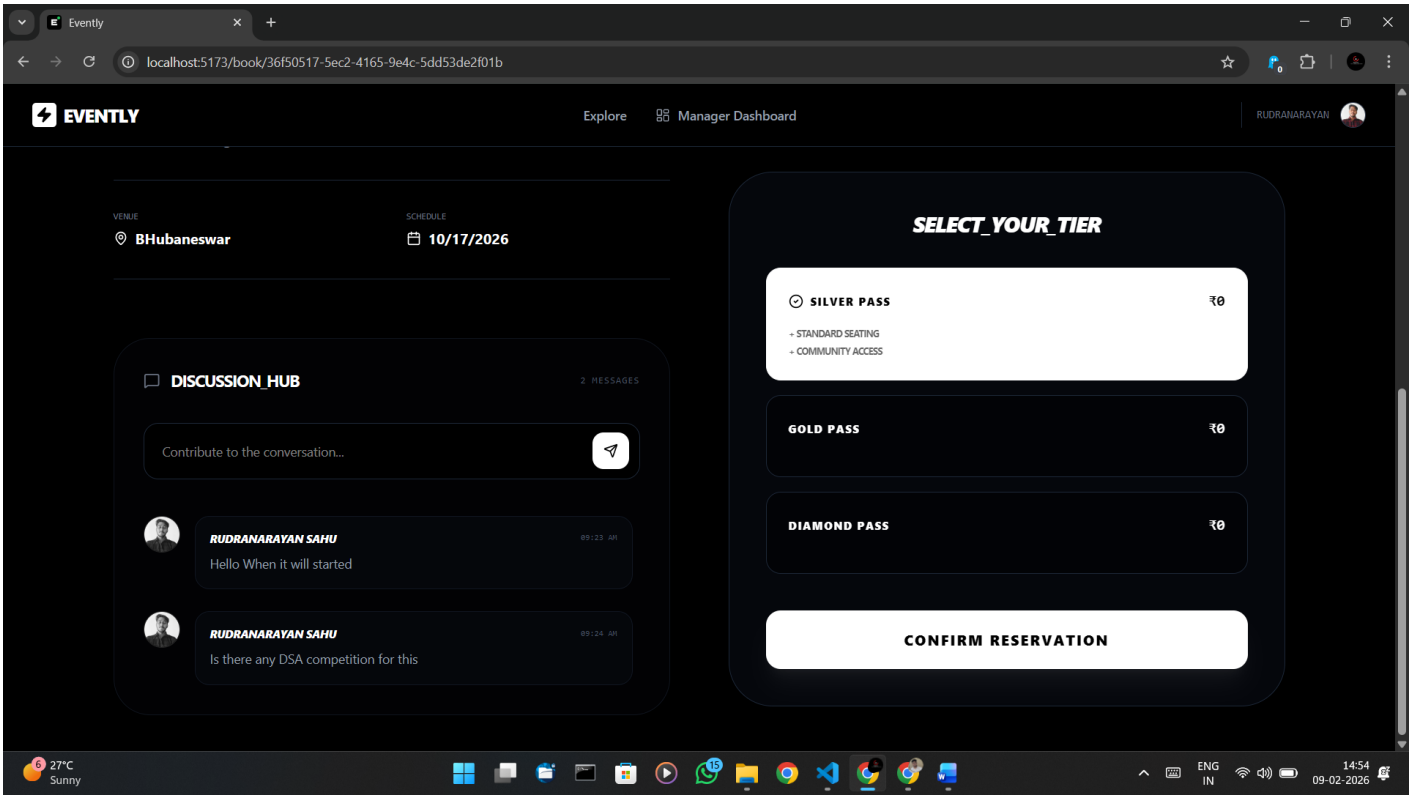
2. **Login and Signup: (Clerk):(Providing protected routing)**

I integrated **Clerk** to handle the entire authentication flow, ensuring that user data is managed securely without me having to store sensitive passwords. Beyond just logins, I used it to set up **Protected Routing**, which acts like a digital gatekeeper for the app. This means that while anyone can browse the public events, only authenticated users can access the booking system or the Manager's Control Panel. It creates a seamless, secure experience where the app "knows" who you are and shows you the right tools based on your account.

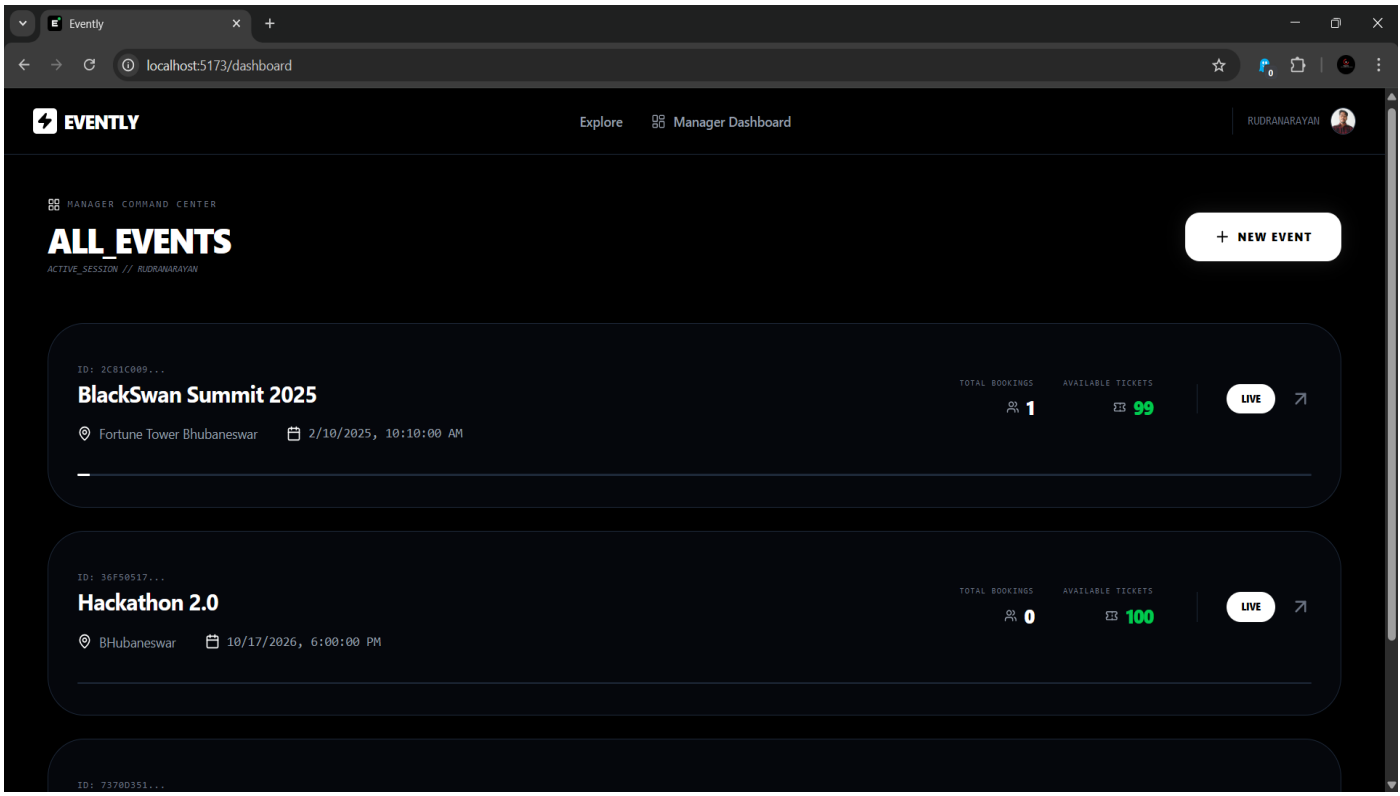


3. **Ticket booking and user engagement : (Protected route):**
   The ticket booking system is built to be intuitive, using **Protected Routes** to ensure that only logged-in users can secure a spot. I designed a tiered selection process where attendees can choose between different levels, like **Diamond** or **Gold**, which adds a layer of engagement and choice. Once a user picks their tier, the system handles the transaction logic and stores the registration in the database, instantly linking that user to the event. At the end of registration there is a engagement section where a logged in user can share feedback or ask about the events or can discuss with others.



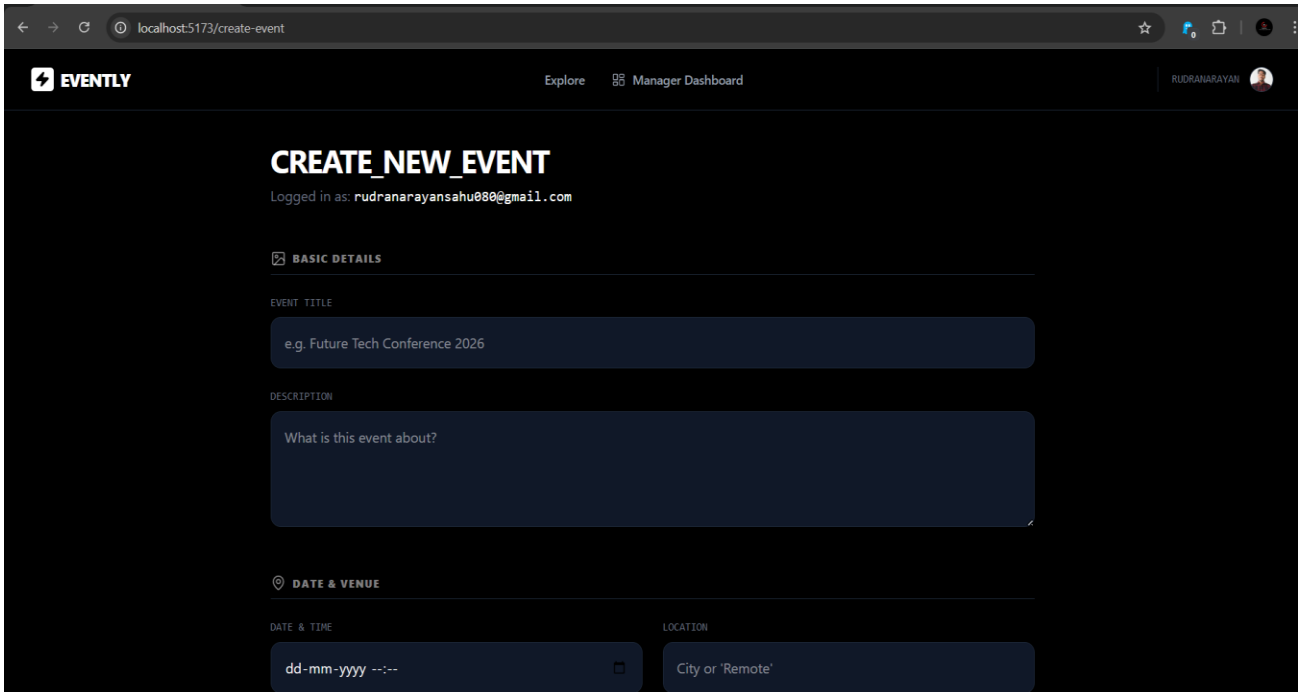


4. **Manager Dashboard (All Events):**
   The Manager Dashboard serves as the central hub for event organizers, providing a high-level overview of every event they've created. I designed this view to give managers instant clarity, showing the status and reach of their entire portfolio at a single glance. It's built to be functional and fast, allowing organizers to quickly jump between different events to check on their progress.



5. **Create new event form:**

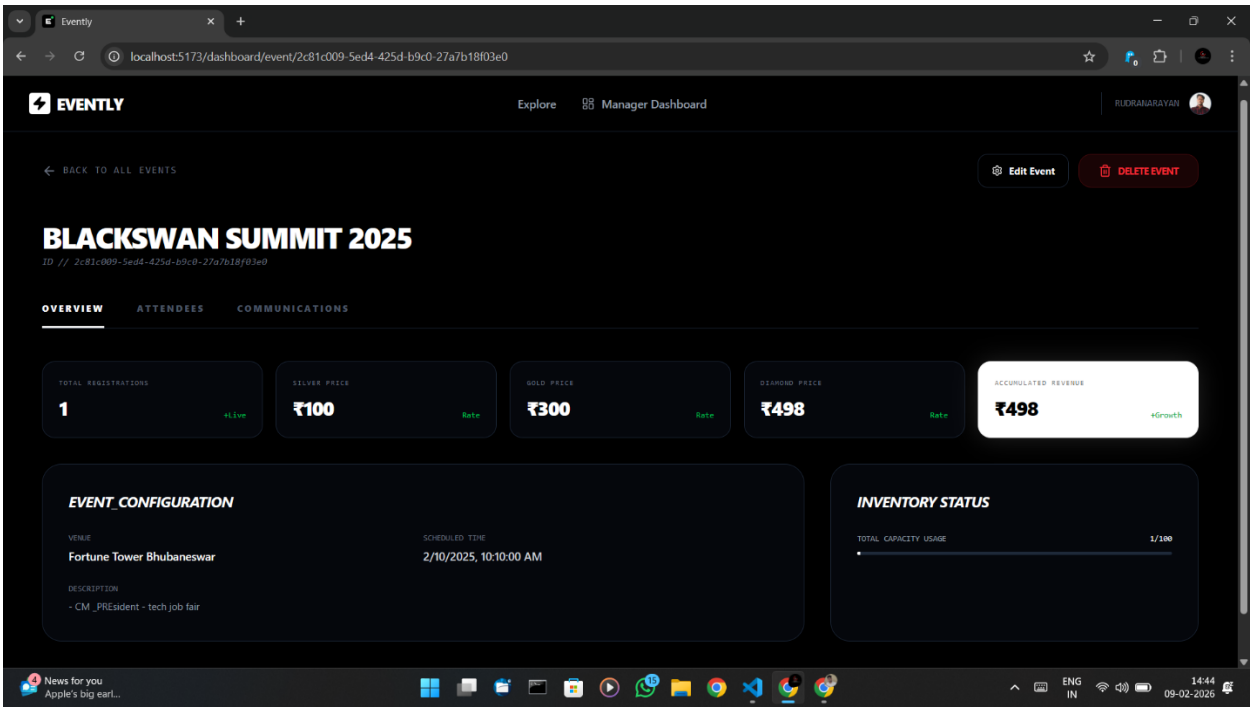
   The **Create Event** form is the entry point for organizers, designed to be as straightforward as possible to encourage event creation. I built it with a clean, single-column layout so managers can quickly input essential details like the event title, date, location, and description without feeling overwhelmed.



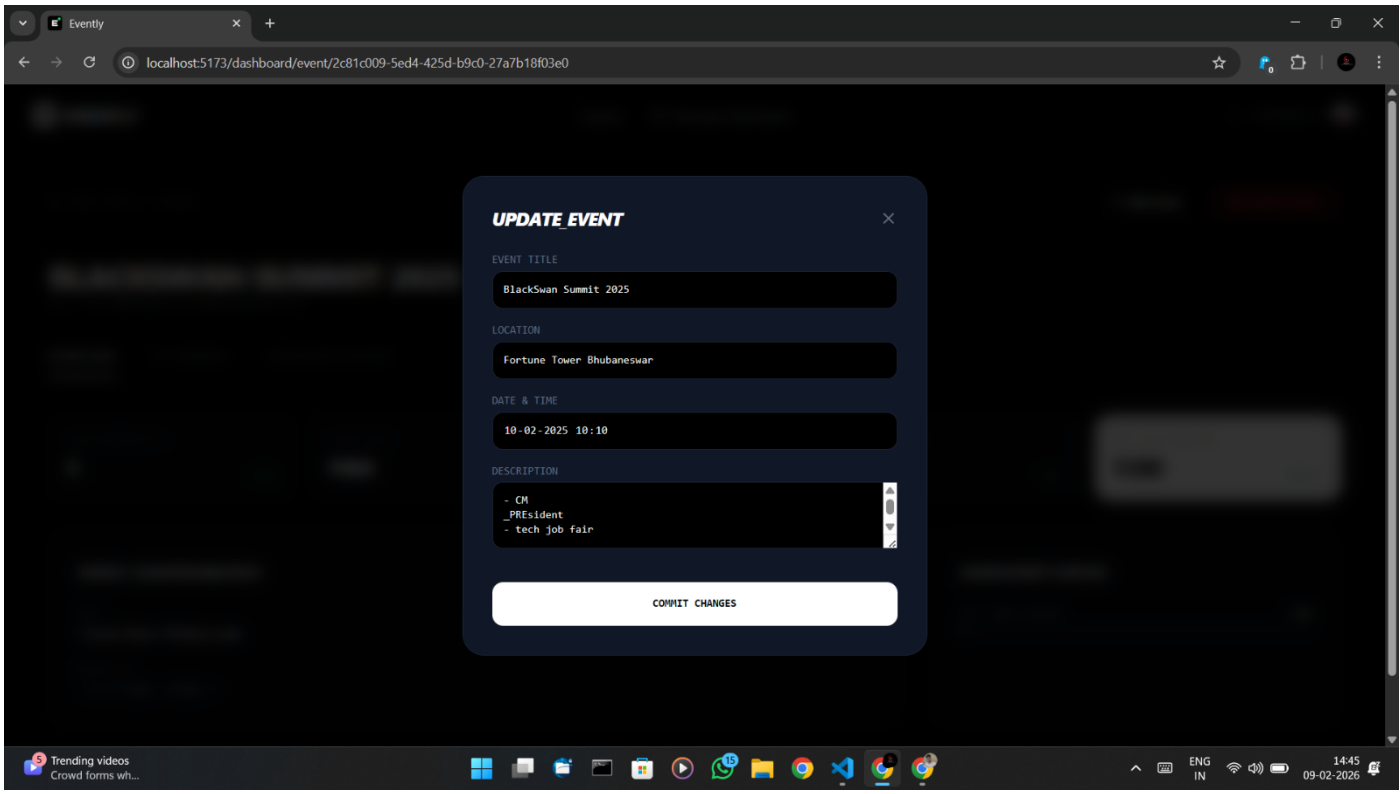
6. **Event Control Section:**

   The Event Control Section is the mission control for each individual event. Once an event is live, I provided managers with a dedicated space to handle the event from monitoring real-time growth to making executive changes.

   This section is where the **Delete** and **Edit** functionalities live, giving organizers full authority over their event's lifecycle. I also integrated the **Broadcast Dispatch** tool here, allowing for instant, high-level communication with the entire attendee list. It's designed to be the "deep dive" area where a manager moves from simply viewing an event to actively running it.

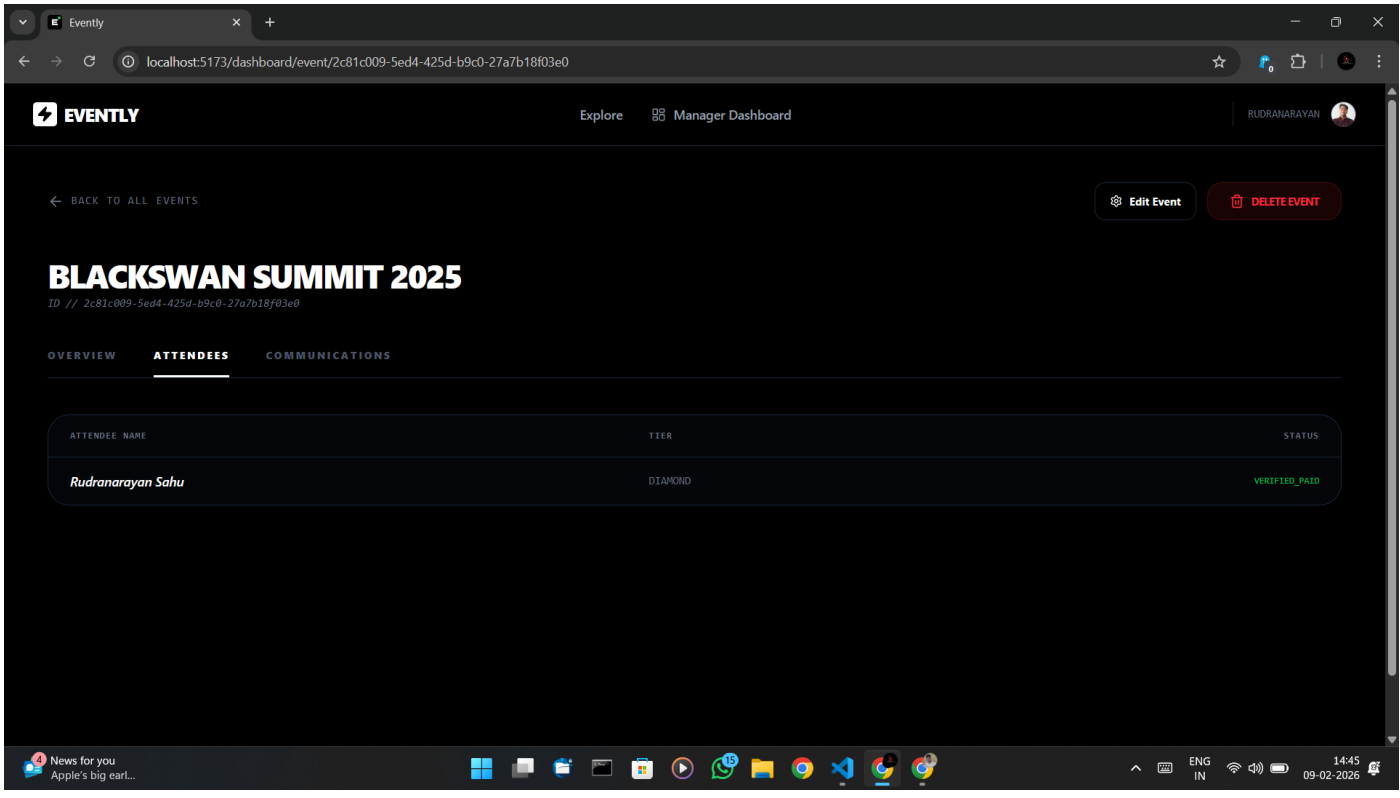


7. Edit Event Form:



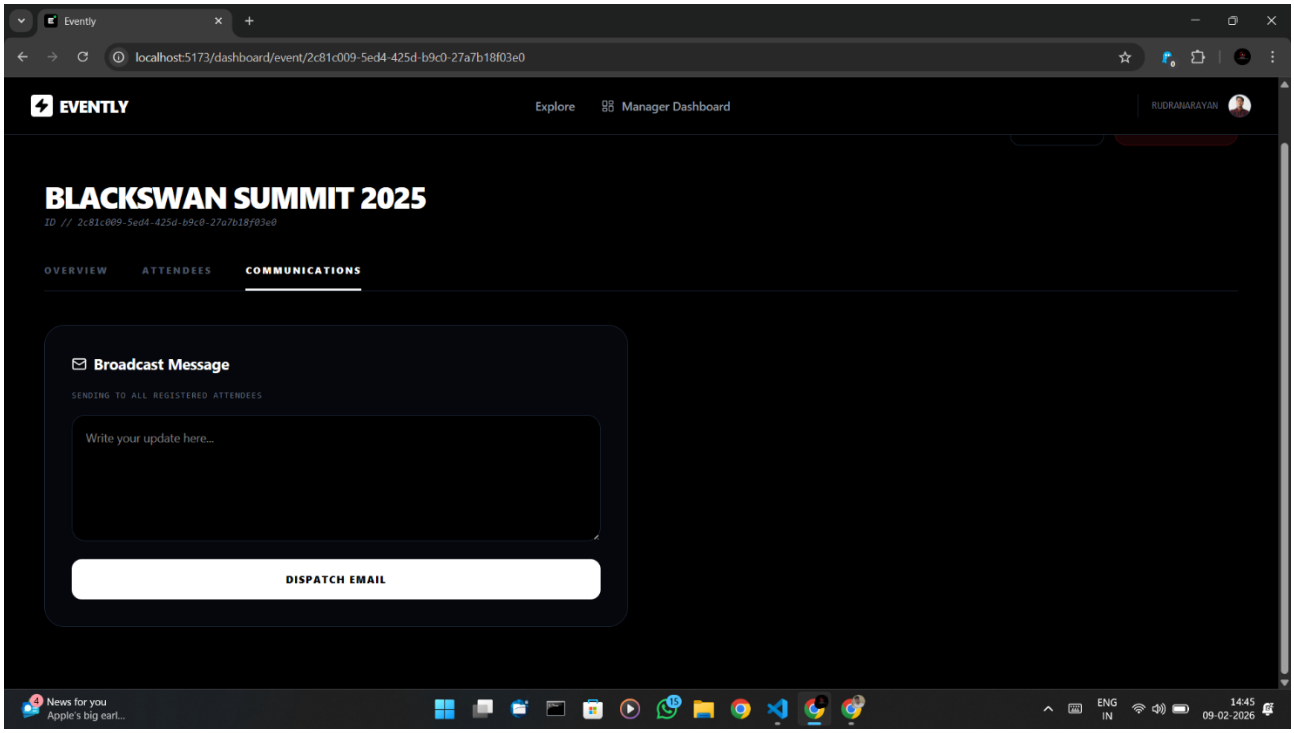
8. **View all Attendees (All attendees according to event):**

The View All Attendees feature provides managers with a transparent, real-time list of everyone who has registered for an event. It pulls data directly from the SQLite database, displaying each attendee's name and their chosen ticket tier, such as Silver, Diamond or Gold.



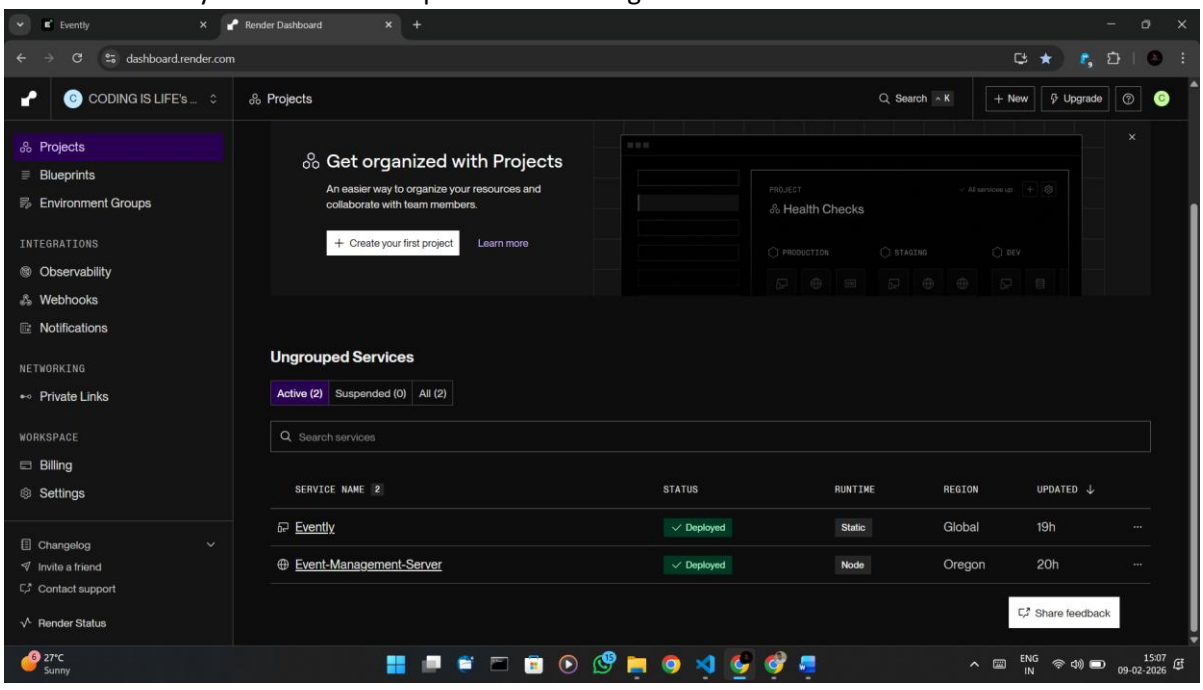
9. Communicate (Email broadcasting):

The Email Broadcasting feature is a powerful tool I built using Nodemailer to bridge the gap between managers and attendees. It allows an organizer to draft a single update and "dispatch" it to every registered guest simultaneously, regardless of how many people are on the list.



10. Deployment:

Deploying on Render required managing a decoupled architecture where the frontend and backend live on separate servers, necessitating a custom CORS configuration to enable secure communication between them. I had to navigate the constraints of the Free Tier, specifically the "cold start" delay where the server sleeps after inactivity, which I handled by adding informative loading states to improve the user experience. By setting up environment variables and distinct build commands, I ensured the SQLite database and Node.js environment stayed stable in a live production setting.



Conclusion

This project isn't just a list of events; it's a tool for communication. It taught me how to handle real-world problems like data synchronization, email delivery, and creating a UI that people actually enjoy looking at.

Live link: https://evently-yebc.onrender.com/
NB: This project deployed in free instance in render so it may take some time to wakeup backend server.

Github: https://github.com/rudranarayan-01/Event-Management