System Design Documentation

1. System Design Approach

Top-Down, Layered Design

- Start with high-level decomposition.
- Identify major subsystems (Auth, Events, Teams, Registration, Inbox, AI, Admin).
- Clearly define responsibilities and interfaces before working on modules.

Modularity & SRP

- Each subsystem has one main responsibility.
- Controllers → request routing
- Services → business logic
- Repositories/DAOs → data storage

API-First

- Clear REST API + WebSocket API.
- Python ML backend exposes controlled, versioned endpoints (recommend, bot).
- Node gateway integrates with these services.

Asynchronous & Decoupled Integration

- Use job queues for:
    - Rebuilding recommendation index
    - Bulk notifications
- Keeps request latency low.

Security by Design

- JWT authentication
- Role-based middleware
- checkSuspension guard for suspended users/entities

Instrumentation

- Simple telemetry:
    - Basic analytics counters
    - Error logging
- No heavy ETL or warehousing.

Incremental Extensibility

- Add new ML models, notification channels, event types, or form fields without breaking core flows.

2. Design Goals

Reliability & Correctness

- Consistent event + registration state.
- Atomic transitions: Pending → Approved → Completed.

Scalability

- Stateless Node servers behind a load balancer.
- Python AI service scales independently.

Responsiveness

- UI should stay fast.
- Heavy tasks executed asynchronously via queues.

Role-Based Security

- Guarded access for:
    - Admin
    - Organizer
    - Student
    - Sponsor

Extensibility

- Easy support for:
    - New event types
    - More ML features
    - Dynamic registration fields

Usability

- Clear dashboards for each role.
- Helpful errors + validations.

Data Integrity & Recoverability

- MongoDB transactions for multi-doc workflows.
- Atlas backups enabled.

Cost-Effectiveness

- Managed services:
    - MongoDB Atlas
    - Qdrant Cloud
    - Cloudinary

3. Interface Design (Black-Box View)

3.1 External Interfaces

User Interface (Web Client)

- Tech: React (Vite), Tailwind, Redux
- Interfaces: REST + WebSocket
- Direct Cloudinary uploads or proxied through backend.

Admin Interface

- Rich dashboard for:

    - College approvals
    - Suspensions
    - Reports & actions

Python ML Backend (FastAPI)

- POST /recommender/query → top-K recommended events
- POST /bot → chatbot lifecycle
- POST /index/add, /delete, /rebuild → vector index ops

Datastore

- MongoDB Atlas → users, events, registrations, inbox, ads, teams
- Qdrant → embeddings + ANN queries
- Cloudinary → media storage

Third-Party LLM API

- Groq API for:
    - Mongo query generation
    - Chatbot summarization

OpenStreetMap / Geocoding

- Tile provider + geocoding via backend proxy.

Email / SMS Provider

- Email verification
- Event reminders (optional)

3.2 Inputs & Outputs

Inputs

- User actions: register, publish event, send messages, upload media
- Admin actions: approve colleges, suspend entities, resolve reports
- ML triggers: publish/complete events → update vectors
- Chat requests → user messages to bot

Outputs

- JSON API responses
- Real-time WebSockets (chat + notifications)
- Email notifications
- Recommendation lists
- Chatbot responses

3.3 Non-Functional Interfaces

Authentication

- JWT issued by Node backend
- Python service validates tokens through gateway

Rate-Limiting

- Limits for:
    - ML endpoints
    - Geocoding proxy

Observability

- Centralized logs
- Request trace logs
- Basic API metrics

A — Subsystem Decomposition

Each subsystem = high cohesion + low coupling.

Authentication Subsystem

- **Responsibilities:**
    - Registration, login
    - JWT issuance & validation
    - Forgot

    - Email verification
    - OAuth hooks
- **Interfaces:**
    - POST /auth/login
    - POST /auth/register
    - POST /auth/forgot-password
- **Data Owned:**
    - User credentials (hashed)
    - Session tokens
    - Email verification tokens

User Management Subsystem

- **Responsibilities:**
    - Profile CRUD
    - Resume uploads
    - Achievements
    - Role changes
    - Directories (sponsors/organizers)
- **Interfaces:**
    - GET /profile/
    - PUT /profile/
    - GET /profile/:id
- **Data Owned:**
    - User collection
    - Resume URL
    - Achievements

Event Management Subsystem

- **Responsibilities:**
    - Create, draft, publish, edit, complete events

    - Suspend events
    - Timeline, gallery, rules
    - Sub-events
    - Check-in flags
- **Interfaces:**
    - POST /organizer/save
    - POST /organizer/publish
    - PUT /organizer/edit/:id
    - PUT /organizer/complete/:id
    - GET /events/:id
- **Data Owned:**
    - Event documents
    - Timeline
    - Gallery images

Registration & Check-in Subsystem

- **Responsibilities:**
    - Dynamic form builder
    - Submission
    - Payment screenshot handling
    - Approval flow
    - Check-in generation & marking
- **Interfaces:**
    - GET /registration/:eventId/form
    - POST /registration/submit
    - POST /registration/checkin
    - GET /registration/:eventId/:participantId/status
- **Data Owned:**
    - Registration forms

    - Payment screenshot URLs
    - Check-in records

Team Management Subsystem

- **Responsibilities:**
    - Create teams
    - Invite/approve members
    - Assign roles
    - Authorization lists
- **Interfaces:**
    - POST /organizer/team/create
    - GET /organizer/team/list
    - POST /student/team/create
- **Data Owned:**
    - OrganizerTeam
    - StudentTeam

Inbox & Notification Subsystem

- **Responsibilities:**
    - Draft + send inbox entities
    - Approvals
    - Status transitions
    - In-app notifications
    - Email push for critical events
- **Interfaces:**
    - POST /inbox/drafts
    - POST /inbox/send
    - GET /inbox/arrivals
    - PUT /inbox/approve/:id
- **Data Owned:**

- InboxEntity
- Notification records

Sponsor Management Subsystem

- **Responsibilities:**
  - Sponsor profiles
  - Ad drafts & publish
  - Track views + likes
  - Analytics
- **Interfaces:**
  - POST /sponsor/ads
  - GET /sponsor/ads
  - PATCH /sponsors/ads/:adId/view
- **Data Owned:**
  - SponsorAd

Search & Filters Subsystem

- **Responsibilities:**
  - Index events
  - Full-text search
  - Auto-suggestions
  - Faceted filters
  - Geo queries
- **Interfaces:**
  - GET /search?q=
  - GET /search/suggest?q=
  - GET /events/nearby?lat=&lng=&radius=
- **Data Owned:**
  - Search index
  - Cached suggestions

Map Annotation Subsystem

- **Responsibilities:**
  - Store annotations
  - Import/export GeoJSON
  - Serve map layers
  - Geocoding proxy
- **Interfaces:**
  - POST /events/:id/annotations
  - GET /events/:id/annotations
  - GET /geocode/search?q=
- **Data Owned:**
  - Event.location.mapAnnotations

AI Subsystem (Python Microservices)

- **Responsibilities:**
  - Embedding generation
  - Qdrant vector ops
  - Chatbot Mongo query generation + execution
  - Index rebuild
- **Interfaces:**
  - POST /ai/recommend
  - POST /ai/bot
  - POST /ai/index/add
  - POST /ai/index/delete
  - POST /ai/rebuild-index
- **Data Owned:**
  - Vector index
  - Temporary caches / prompt info

Admin & College Management Subsystem

- **Responsibilities:**
    - College approvals
    - Suspend/unsuspend entities
    - Admin reports
    - User-filed report management
- **Interfaces:**
    - PATCH /admin/colleges/:collegeId/handle
    - PATCH /admin/suspend/:modelType/:id
    - POST /admin/report/:modelType/:id
- **Data Owned:**
    - College collection
    - Report documents

Media & File Service

- **Responsibilities:**
    - Handle image/resume uploads
    - Provide signed URLs
    - Centralized upload service
- **Interfaces:**
    - Direct Cloudinary upload
    - POST /media/upload
- **Data Owned:**
    - Cloud URLs stored in related docs

B — Architectural Layers

Presentation Layer (Frontend)

- **Stack:** React (Vite), Tailwind, Redux Toolkit, react-hook-form

- **Responsibilities:**
    - Render pages
    - Client-side routing
    - Socket.IO client
    - Form validation
    - Send requests & uploads
- **Security:**
    - Token storage (HTTP-only cookie / secure storage)
    - Optional CSRF

API / Application Layer (Node.js Express)

- **Responsibilities:**
    - Expose REST + WebSocket gateway
    - Implement business logic
    - Authorization middleware
    - Proxy to Python AI service
    - Orchestrate multi-step workflows
- **Modules:** Controllers, Services, Repositories, Jobs

AI Layer (Python Microservices)

- **Stack:** FastAPI, HuggingFace, Qdrant client, LangChain/LangGraph, Groq API
- **Responsibilities:**
    - Vector ops
    - Recommendations
    - Chatbot Mongo query generation
    - Safe executor
    - Lightweight authenticated endpoints

Data Layer

- MongoDB Atlas
- Qdrant

- Cloudinary

Infrastructure & Ops

- **Frontend → Vercel**

- **Backend → Render**
- **Database → MongoDB Atlas**
- **Vector DB → Qdrant**

Subsystem composition Diagram