EVENTSPHERE

### **Developer Guide for "EventSphere"**

### **Introduction**

#### **1.1 Purpose of this Guide**

This Developer Guide provides a comprehensive technical reference for the design, development, testing, deployment, and maintenance of EventSphere. It is intended for software engineers, system architects, database administrators, testers, and DevOps engineers who will be responsible for building, operating, and evolving the system. The document contains detailed architecture descriptions, database schemas, API design guidance, code conventions, testing strategies, deployment steps, and maintenance and monitoring recommendations. It also includes diagrams that visually describe the system topologies and workflows.

#### **1.2 Audience**

This document targets the following roles:

* **Frontend Developers:** Implement UI/UX, responsive components, and client-side logic.
* **Backend Developers:** Build RESTful APIs, business logic, and integration layers.
* **Database Engineers:** Design and optimize relational schema, indexing, and migrations.
* **QA/Test Engineers:** Create test plans, automate tests, and validate system behavior.
* **DevOps/System Administrators:** Deploy and monitor production systems, and manage CI/CD pipelines and backups.

#### **1.3 References**

* EventSphere Software Requirements Specification (SRS) v1.0
* ReactJS, Bootstrap, Node.js/Express, MySQL official documentation
* OWASP Top 10 and WCAG 2.1 accessibility standards

### **2. System Overview**

#### **2.1 Background and Problem Statement**

Colleges and universities organize a wide variety of events, but traditional communication methods often lead to missed updates, low participation, and administrative overhead. EventSphere centralizes these activities into a single web platform that supports registration, attendance tracking, media management, certificate issuance, and analytics.

#### **2.2 Core Features**

EventSphere implements the following high-level features:

* **Role-Based Access Control:** Differentiated access for Visitor, Participant, Organizer, and Admin roles.
* **Event Lifecycle Management:** Creation by Organizers, approval by Admins, editing, cancellation, and archiving.
* **Registration & Capacity Controls:** Real-time slot availability, waitlists, and automated promotion from waitlist.
* **Attendance Management:** QR code check-in and attendance reports.
* **Certificate Management:** Uploading and distribution of certificates post-event.
* **Media Gallery:** Upload, moderate, and categorize images and videos.
* **Notifications:** Email and push-notifications for registrations, updates, and announcements.
* **Calendar Integration:** .ics exports and Add-to-Calendar buttons.
* **Social Share:** Pre-formatted messages for popular platforms.
* **Analytics & Reporting:** Participation metrics, top events, feedback trends, and exportable reports.

### **3. System Architecture**

EventSphere uses a **three-tier architecture**, separating presentation, application logic, and data persistence. This allows independent scaling and maintainability.

* **Frontend Layer:** Implemented in **ReactJS**, it is responsible for rendering user views, handling client-side routing, and performing API calls.
* **Backend Layer:** Implements **RESTful endpoints** using **Node.js** and **Express**. Key modules include authentication, event management, and notification services.
* **Data Layer:** **MySQL** is used for its relational integrity and support for transactions. Tables are normalized to at least 3NF, with indexing on frequently queried columns.

### **4. Database Design**

This section specifies the relational schema, essential table definitions, and indexing strategy.

#### **4.1 Core Tables and Columns**

* **Table: Users**
  + user\_id (INT): PK, AUTO\_INCREMENT
  + email (VARCHAR(100)): UNIQUE, used for login
  + password\_hash (VARCHAR(255)): bcrypt hashed password
  + role (ENUM('participant','organizer','admin')): User role
  + created\_at (DATETIME): Account creation timestamp
* **Table: UserDetails**
  + detail\_id (INT): PK, AUTO\_INCREMENT
  + user\_id (INT): FK -> Users(user\_id)
  + full\_name (VARCHAR(100)): User's full name
  + mobile (VARCHAR(15)): Contact number
  + department (VARCHAR(100)): Academic department
  + enrollment\_no (VARCHAR(50)): Student enrollment number
* **Table: Events**
  + event\_id (INT): PK, AUTO\_INCREMENT
  + title (VARCHAR(150)): Event title
  + description (TEXT): Detailed description
  + category (VARCHAR(50)): Event category
  + date (DATE): Event date
  + time (TIME): Start time
  + venue (VARCHAR(100)): Event location
  + organizer\_id (INT): FK -> Users(user\_id)
  + max\_participants (INT): Capacity
  + status (ENUM('pending','approved','cancelled')): Approval lifecycle

#### **4.2 Additional Tables**

Registrations, Attendance, Feedback, Certificates, MediaGallery, EventSeating, Waitlist, CalendarSync, and EventShareLog are also implemented as per the SRS.

### **5. Functional Modules - Detailed**

#### **5.1 User Management Module**

* **User Registration:** POST /api/v1/auth/register. The system validates input, hashes passwords with bcrypt, and sends a verification email.
* **Authentication:** **JWT tokens** are issued for stateless sessions. **Refresh tokens** are used to renew access tokens. Admin logins require **Two-Factor Authentication (2FA)**.

#### **5.2 Event Management Module**

* **Event Creation:** Organizers create an event via POST /api/v1/events. The event remains in a pending state until an Admin approves it.
* **Capacity & Waitlist:** A transaction is used to decrement available seats upon registration. If capacity is full, the user is added to a waitlist.

#### **5.3 Registration Workflow - Example API**

A sample API sequence for registration would be:

1. GET /api/v1/events/:id - fetches event details and current slots.
2. POST /api/v1/events/:id/register - attempts registration with a user token.

### **6. Attendance and Certificate Management**

Attendance is captured using a QR code system. The QR payload contains a signed token that the organizer's scanning app sends to the backend for verification. Certificates are then uploaded by organizers or generated via a templating engine.

### **7. Media Gallery and Content Moderation**

Organizers can upload media, and Admins moderate the content. Large files should be stored in **object storage** (e.g., AWS S3) and served via a **CDN**. Thumbnails and optimized formats are generated server-side.

### **8. Notifications and Calendar Integration**

The Notification Service handles emails for confirmations and reminders. Push notifications can be used for PWAs. Calendar integration is provided via an .ics file.

### **9. Security Considerations**

* **Authentication & Authorization:** Use bcrypt for password hashing, JWT for sessions, and 2FA for admin accounts. **Role-based access control (RBAC)** must be enforced at both the API and UI levels.
* **Input Validation:** Sanitize all inputs and use **parameterized queries** to prevent SQL injection.
* **Data Protection:** Encrypt sensitive data at rest and in transit using **HTTPS**.

### **10. Testing Strategy - Detailed**

Testing is tiered:

* **Unit Tests:** For business logic.
* **Integration Tests:** To validate API endpoints and database transactions.
* **End-to-End Tests:** To simulate full user journeys.
* **Performance & Load Testing:** To test the system under load using tools like k6 or JMeter.

### **11. Deployment and CI/CD**

A robust CI/CD pipeline ensures repeatable deployments. Steps include linting, testing, building artifacts (frontend static bundle, backend Docker image), and deploying to staging and production environments.

### **12. Monitoring, Logging and Maintenance**

Monitor application health using metrics (CPU, memory, response times) and centralize logs. Implement regular database backups with off-site storage.

### **13. Operational Runbook**

This section should cover common operational tasks like restarting services, rolling back releases, and safely running database migrations.

### **14. Future Enhancements and Roadmap**

Planned enhancements include integrating a payment gateway for certificate fees, a recommendation engine for events, and building a mobile application.