Event Management Backend System Report

EventSync Team

September 4, 2025

Contents

1 Executive Summary

This report provides a comprehensive overview of the Event Management Backend, a robust API system designed for managing events, tickets, users, seat maps, notifications, and analytics. Built with Node.js, Express, and MongoDB, the system supports user authentication, event creation, ticket booking with QR code generation, seat management, and administrative analytics. The backend ensures security through JWT-based authentication and role-based authorization, with a modular architecture for scalability and maintainability. This report details the system's architecture, features, API endpoints, and setup instructions, serving as a guide for developers and stakeholders.

2 Project Overview

The Event Management Backend is a RESTful API designed to streamline event organization and ticket management. It caters to two primary user roles: regular users who book tickets and admins who manage events, users, and analytics. Key functionalities include event creation, ticket booking with seat selection, QR code generation for tickets, real-time notifications, and detailed analytics for event insights.

2.1 Objectives

- Provide a secure and scalable backend for event management.
- Enable seamless ticket booking with QR code integration.
- Support administrative tasks with analytics and user management.
- Ensure robust security through authentication and authorization.
- Facilitate real-time notifications for users and admins.

2.2 Target Audience

- Event organizers (admins) needing tools to manage events and track performance.
- End users looking to browse events, book tickets, and manage their profiles.
- Developers integrating the backend with front-end applications.

3 Technical Architecture

The backend is built using modern web technologies and follows a modular, layered architecture.

3.1 Technology Stack

- **Node.js**: Server-side JavaScript runtime for high-performance APIs.

- **Express**: Web framework for routing and middleware.
- **MongoDB**: NoSQL database for flexible data storage.
- **Mongoose**: Object Data Modeling (ODM) for MongoDB.
- **JWT**: JSON Web Tokens for secure authentication.
- **Bcrypt**: For password hashing.
- **Joi**: For input validation (e.g., event creation).
- **QRCode**: Library for generating ticket QR codes.
- **Helmet**: For securing HTTP headers.
- **Morgan**: For request logging.
- **CORS**: For cross-origin resource sharing.

3.2 System Architecture

The system follows a Model-View-Controller (MVC) pattern:

- **Models**: Define database schemas (User, Event, Ticket, SeatMap, Notification).
- **Controllers**: Handle business logic for API endpoints.
- **Routes**: Map HTTP requests to controller functions.
- **Middlewares**: Manage authentication, authorization, and error handling.

3.3 File Structure

```
event-management-backend/├──
 config/│└──
    db.js               % MongoDB connection├──
 controllers/│├──
    analyticsController.js % Analytics endpoints│├──
    authController.js      % Authentication logic│├──
    eventController.js     % Event management│├──
    notificationController.js % Notification handling│├──
    seatMapController.js   % Seat map management│├──
    ticketController.js    % Ticket booking│└──
    userController.js      % User management├──
 middlewares/│├──
    authMiddleware.js      % JWT and admin checks│└──
    errorHandler.js        % Error handling├──
 models/│├──
    Event.js              % Event schema│├──
    Notification.js        % Notification schema│├──
    SeatMap.js            % Seat map schema│├──
    Ticket.js             % Ticket schema│└──
```

```
    User.js                % User schema├──
routes/│├──
    analyticsRoutes.js     % Analytics routes│├──
    authRoutes.js          % Authentication routes│├──
    eventRoutes.js         % Event routes│├──
    notificationRoutes.js  % Notification routes│├──
    seatMapRoutes.js       % Seat map routes│├──
    ticketRoutes.js        % Ticket routes│└──
    userRoutes.js          % User routes├──
utils/│├──
    qrGenerator.js         % QR code generation│└──
    validators.js          % Joi validation├──
server.js                   % Server entry point└──
.env                        % Environment variables
```

4 Features

- **User Management**: Register, login, and update profiles with role-based access (admin/user).
- **Event Management**: Create, update, delete, and filter events by status, search, or popularity.
- **Ticket Booking**: Book tickets with seat selection and QR code generation.
- **Seat Management**: Create seat maps, reserve/book seats, and release expired reservations.
- **Notifications**: User-specific, broadcast, and admin notifications.
- **Analytics**: Dashboards for event counts, revenue, attendee demographics, and sales reports.

5 API Endpoints

All endpoints are prefixed with `/api`. Authentication is required for most routes, with admin-only routes restricted to users with the `admin` role.

5.1 Authentication (`/api/auth`)

- **POST /register**: Register a new user (name, email, password, age, gender, location, interests).
- **POST /login**: Login and receive a JWT token.

5.2 Events (`/api/events`)

- **GET /**: List events with filtering (status, search) and sorting (date, price, popularity).

- **POST** /: Create an event (admin only).
- **GET** /**:id**: Get event details.
- **PUT** /**:id**: Update event (admin only).
- **DELETE** /**:id**: Delete event (admin only).
- **PUT** /**:id**/**seats**: Update event seats (admin only).
- **POST** /**events**/**tickets**: Book tickets.

5.3 Tickets (`/api/tickets`)

- **POST** /**book**: Book a ticket with QR code.
- **GET** /**my**: Get user's tickets.
- **GET** /**event**/**:eventId**: Get tickets for an event (admin only).
- **PUT** /**:id**: Update ticket (admin only).
- **GET** /**all**: Get all tickets (admin only).

5.4 Seat Maps (`/api/seatmaps`)

- **POST** /: Create seat map (admin only).
- **GET** /**:eventId**: Get seat map.
- **GET** /**:eventId**/**available**: Get available seats.
- **GET** /**:eventId**/**seat**/**:seatNumber**: Get seat details.
- **POST** /**reserve**: Reserve seats.
- **POST** /**book**: Book seats.
- **DELETE** /**:eventId**/**seat**/**:seatNumber**: Release seat.
- **PUT** /**:eventId**: Update seat map (admin only).

5.5 Users (`/api/users`)

- **GET** /**profile**: Get user profile and stats.
- **PUT** /**profile**: Update user profile.
- **GET** /: List users (admin only).
- **GET** /**:id**: Get user by ID (admin only).
- **PUT** /**:id**: Update user (admin only).
- **DELETE** /**:id**: Delete user (admin only).

5.6 Notifications (`/api/notifications`)

- **GET** /: Get user notifications.
- **PUT** /**:id/read**: Mark notification as read.
- **PUT** /**read-all**: Mark all notifications as read.
- **DELETE** /**:id**: Delete a notification.
- **DELETE** /: Clear all notifications.

5.7 Analytics (`/api/analytics`)

- **GET** /**dashboard**: Admin dashboard (events, bookings, revenue).
- **GET** /**attendees**: Attendee demographics.
- **GET** /**reports**: Sales reports by date.
- **GET** /**insights**: Event-specific insights (attendees, social media).

5.8 Health Check

- **GET** /**api/health**: Server status and uptime.

6 Setup Instructions

6.1 Prerequisites

- Node.js (v14 or higher)
- MongoDB (local or cloud, e.g., MongoDB Atlas)
- npm or yarn

6.2 Installation

1. Clone the repository:

```
git clone <repository-url>
cd event-management-backend
```

2. Install dependencies:

```
npm install
```

3. Create a `.env` file:

```
PORT=5000
MONGO_URI=<your-mongodb-uri>
JWT_SECRET=<your-jwt-secret>
NODE_ENV=development
```

4. Start the server:

```
npm start
```

7 Security

- **Authentication**: JWT tokens in `Authorization: Bearer <token>` header.
- **Authorization**: Admin-only routes use `admin` middleware.
- **Passwords**: Hashed with bcrypt.
- **Validation**: Joi for input validation.
- **HTTP Security**: Helmet for secure headers, CORS for cross-origin requests.
- **Error Handling**: Centralized middleware for consistent error responses.

8 Error Handling

Errors return JSON with:

- `message`: Error description.
- `stack`: Stack trace (omitted in production).

Common status codes:

- 200: Success
- 201: Resource created
- 400: Bad request
- 401: Unauthorized
- 403: Forbidden
- 404: Not found
- 500: Server error

9 Future Improvements

- Add rate limiting to prevent API abuse.
- Implement email notifications.
- Support multiple ticket types (e.g., VIP).
- Enhance analytics with visualizations.
- Add unit and integration tests.

10 Conclusion

The Event Management Backend provides a robust, secure, and scalable solution for event and ticket management. Its modular design and comprehensive API make it suitable for integration with various front-end applications. Future enhancements will further improve its functionality and user experience.