Event Management System: Backend and Frontend Report

Contents

1 Executive Summary

The Event Management System is a comprehensive web application designed to streamline event organization, ticket booking, and analytics. It consists of a **backend** (Node.js, Express, MongoDB) and a **frontend** (React, Vite, Tailwind CSS), working together to provide a seamless experience for users and administrators. The backend handles data management, authentication, and API services, while the frontend delivers an intuitive, responsive interface for event browsing, ticket purchasing, and analytics visualization. This report details the architecture, features, components, API integration, setup instructions, and fu- ture enhancements for both systems, providing a holistic view for developers and stakeholders.

2 System Overview

The Event Management System supports two user roles: **users** (who browse events, book tickets, and manage profiles) and **admins** (who manage events, users, and analytics). The backend provides a robust RESTful API, while the frontend offers a modern, interactive UI. The system supports event creation, ticket booking with QR code generation, seat management, notifications, and detailed analytics.

2.1 Objectives

- Provide a scalable backend API for event and ticket management.
- Deliver a user-friendly, responsive frontend for seamless interaction.
- Ensure secure authentication and role-based access control.
- Offer comprehensive analytics for event organizers.
- Support real-time notifications and interactive seat selection.

2.2 Target Audience

- **Users**: Individuals browsing and booking events.
- **Admins**: Event organizers managing events and analytics.
- **Developers**: Teams integrating or extending the system.

3 Technical Architecture

The system is divided into backend and frontend components, integrated via a RESTful API.

3.1 Backend Technology Stack

- **Node.js**: Server-side JavaScript runtime.

- **Express**: Web framework for RESTful APIs.
- **MongoDB**: NoSQL database for data storage.
- **Mongoose**: ODM for MongoDB schema management.
- **JWT**: JSON Web Tokens for authentication.
- **Bcrypt**: For password hashing.
- **Joi**: For input validation.
- **QRCode**: For generating ticket QR codes.
- **Helmet**: For securing HTTP headers.
- **Morgan**: For request logging.
- **CORS**: For cross-origin requests.

3.2 Frontend Technology Stack

- **React**: JavaScript library for UI components.
- **Vite**: Fast build tool and development server.
- **Tailwind CSS**: Utility-first CSS framework.
- **React Router**: For client-side routing.
- **Axios**: For API requests.
- **JWT Decode**: For decoding JWT tokens.
- **Nivo**: For bar and lollipop charts.
- **Recharts**: For line and donut charts.
- **Lucide React** and **React Icons**: For UI icons.

3.3 System Architecture

- **Backend**: Follows MVC pattern with models (schemas), controllers (busi- ness logic), routes (API endpoints), and middlewares (authentication, error handling).
- **Frontend**: Component-based architecture with reusable UI components, pages for routes, and services for API calls.
- **Integration**: The frontend communicates with the backend via Axios, us- ing JWT tokens for authentication.

4 File Structure

4.1 Backend

```
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
```

4.2 Frontend

```
src/├──
 assets/                    % Static assets├──
 common/                    % Reusable components (e.g., Button)
 pages/|├                   ├──
 –                         % Admin dashboard|├──
    AdminDashboard.jsx     % Analytics visualizations|├──
    Analytics.jsx          % Ticket booking|├──
    Bookings.jsx           % Event category
    EventCategories.jsx    management|├──
    EventDetailsPage.js    % Event details|├──
    x                      % Login page|├──
    Login.jsx
```

```
    MyTickets.jsx           % User's tickets|├──
    Notifications.jsx       % Notifications
    page|├── Register.jsx    % Registration page|├──
    Settings.jsx            % User settings|├──
    Support.jsx             % Support page|├──
    UserEvent.jsx           % Event browsing|├──
    AllAttendeeInsights.jsx % All events insights|├──
    SingleEventAttendeeInsights.jsx % Single event
    insights|└── AddEvent.jsx   % Add event page├──
 services/|├──
    authService.js          % Authentication APIs|├──
    eventService.js         % Event APIs|└──
    notificationService.js % Notification
    APIs├──
 utils/|├──
    authUtils.js            % Authentication utilities|└──
    qrUtils.js              % QR code
 generation├── components/|├──
    AnalyticsCards.jsx      % Analytics cards|├──
    AttendeeLocationsCard.jsx % Attendee
    locations|├── BarChart.jsx  % Bar chart|├──
    Card.jsx                % Generic card|├──
    DonutChart.jsx          % Donut chart|├──
    EventCard.jsx           % Event card|├──
    EventDetails.jsx        % Event details|├──
    EventStatusLegend.jsx   % Event status legend|├──
    LatestEvent.jsx         % Latest event with seat
    map|├── LineChart.jsx   % Line chart|├──
    LolipopChart.jsx        % Lollipop chart|├──
    Navbar.jsx              % Event management
    header|├── Notifications.jsx     % Notifications
    component|├── RegisterForm.jsx   % Registration
    form|├── SeatPicker.jsx % Seat selection|├──
    SocialMediaCard.jsx     % Social media metrics├──
 App.jsx                     % Main app with routing├──
 main.jsx                    % React entry point└──
 index.css                   % Global styles
```

5 Features

5.1 Backend

- **User Management**: Register, login, profile updates, and admin user management.
- **Event Management**: Create, update, delete, and filter events.
- **Ticket Booking**: Book tickets, generate QR codes, and manage ticket sta-

tuses.

- **Seat Management**: Create and manage seat maps, reserve/book seats.
- **Notifications**: User-specific, broadcast, and admin notifications.
- **Analytics**: Dashboards for event counts, revenue, and attendee demograph- ics.

5.2 Frontend

- **Authentication**: Login and registration forms with error handling.
- **Event Browsing**: Filter, search, and view event details.
- **Ticket Booking**: Interactive seat picker with QR code generation.
- **Notifications**: Display latest notifications with pagination.
- **Analytics**: Visualize attendee data and sales with charts.
- **Responsive UI**: Tailwind CSS for desktop and mobile compatibility.

6 API Endpoints and Integration

The frontend communicates with the backend via Axios, with tokens added to requests automatically.

6.1 Key Endpoints

- **Authentication (`/api/auth`)**:
  - `POST /register`: Register user.
  - `POST /login`: Login and receive JWT.
- **Events (`/api/events`)**:
  - `GET /`: List events with filters.
  - `POST /`: Create event (admin).
  - `GET /:id`, `PUT /:id`, `DELETE /:id`: Event CRUD.
  - `POST /events/tickets`: Book tickets.
- **Tickets (`/api/tickets`)**:
  - `POST /book`: Book ticket with QR code.
  - `GET /my`: User's tickets.
- **Seat Maps (`/api/seatmaps`)**:
  - `POST /`: Create seat map (admin).
  - `GET /:eventId`: Get seat map.

  - `POST /reserve`, `POST /book`: Reserve/book seats.
- **Users (`/api/users`)**:
  - `GET /profile`, `PUT /profile`: User profile management.
  - `GET /`, `GET /:id`, `PUT /:id`, `DELETE /:id`: Admin user management.
- **Notifications (`/api/notifications`)**:
  - `GET /`: Get user notifications.
  - `PUT /:id/read`, `PUT /read-all`: Manage notifications.
- **Analytics (`/api/analytics`)**:
  - `GET /dashboard`: Admin dashboard stats.
  - `GET /attendees`, `GET /reports`, `GET /insights`: Analytics data.

6.2 Integration

The frontend uses `authService.js` for authentication, `eventService.js` for event operations, and `notificationService.js` for notifications. Axios in `api.js` adds JWT tokens to requests, ensuring secure communication.

7 Frontend Component Details

- **AnalyticsCards.jsx**: Displays metrics (age, gender, location, interests) with trend indicators.
- **AttendeeLocationsCard.jsx**: Table for attendee locations with color-coded counts.
- **BarChart.jsx**: Bar chart for location data.
- **Card.jsx**: Generic card for metrics.
- **DonutChart.jsx**: Donut chart for percentage-based analytics.
- **EventCard.jsx**: Event details with delete option (admin).
- **EventDetails.jsx**: Detailed event view with editable fields (admin).
- **EventStatusLegend.jsx**: Legend for event statuses.
- **LatestEvent.jsx**: Latest event with seat map (paid, reserved, empty).
- **LineChart.jsx**: Line chart for sales with metrics.
- **LolipopChart.jsx**: Lollipop chart for attendee age.
- **Navbar.jsx**: Header with filters, search, and admin controls.
- **Notifications.jsx**: Displays latest 5 notifications.
- **RegisterForm.jsx**: Registration form with validation.
- **SeatPicker.jsx**: Interactive seat selection grid.

- **SocialMediaCard.jsx**: Social media engagement metrics.

8 Setup Instructions

8.1 Prerequisites

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

8.2 Backend Setup

1. Clone the backend repository:

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

2. Install dependencies:

```
npm install
```

3. Create `.env` file:

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

4. Start the server:

```
npm start
```

8.3 Frontend Setup

1. Clone the frontend repository:

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

2. Install dependencies:

```
npm install
```

3. Create `.env` file:

```
VITE_API_URL=http://localhost:5000/api
```

4. Start the development server:

```
npm run dev
```

9 Security

- **Backend**:
  - JWT-based authentication with tokens in `Authorization: Bearer <token>` header.
  - Admin-only routes protected by middleware.
  - Passwords hashed with bcrypt.
  - Joi for input validation.
  - Helmet for secure headers, CORS for cross-origin requests.
- **Frontend**:
  - JWT tokens stored in `localStorage`.
  - `PrivateRoute` component restricts access by role.
  - Error messages displayed from backend responses.

10 Error Handling

- **Backend**: Centralized error handler returns JSON with `message` and `stack` (omitted in production). Common status codes: 200, 201, 400, 401, 403, 404, 500.
- **Frontend**: Displays API error messages (e.g., login/registration failures) and handles loading states.

11 Future Improvements

- **Backend**:
  - Add rate limiting to prevent API abuse.
  - Implement email notifications.
  - Support multiple ticket types (e.g., VIP).
  - Add unit and integration tests with Jest/Mocha.
- **Frontend**:
  - Add unit tests with Jest/React Testing Library.
  - Implement real-time notifications with WebSockets.
  - Enhance accessibility (ARIA labels, keyboard navigation).
  - Add loading spinners and skeleton screens.
  - Support internationalization (i18n).

12 Conclusion

The Event Management System, with its robust backend and intuitive frontend, provides a comprehensive solution for event organization and ticket manage- ment. The backend's scalable API and the frontend's responsive UI ensure a seamless experience for users and admins. Future enhancements will further improve functionality, security, and user experience.