A black and yellow logo with a black shield and a black and yellow logo

Description automatically generated, Picture

**Student Information**

Name: Md Hasib Zaman

Student ID: B01016875

Course: CSCI5709 Advanced Web Development

Assignment 2: Individual API Design and Implementation

Contents

[Introduction 3](#_Toc202652116)

[Core Feature 1: Event Management 3](#_Toc202652117)

[Core Feature 2: Seat Map Selection 12](#_Toc202652118)

[Core Feature 3: E-Ticket Generation with QR Code 17](#_Toc202652119)

[Security-Related Endpoints and Risk Mitigation 21](#_Toc202652120)

[Security Risk Mitigation 23](#_Toc202652121)

[Data Integration and Modeling 23](#_Toc202652122)

[Data Models 24](#_Toc202652123)

[CRUD Operations Mapping 26](#_Toc202652124)

[Implementation and Demonstration 27](#_Toc202652125)

[Implementation Details 27](#_Toc202652126)

[Live Deployment 27](#_Toc202652127)

[Endpoint 1: Create Event (POST /api/events) 28](#_Toc202652128)

[Endpoint 2: Show Events (GET /api/events) 29](#_Toc202652129)

[Database Persistence Evidence 30](#_Toc202652130)

[Required Screenshots for Assignment 32](#_Toc202652131)

[Conclusion 34](#_Toc202652132)

# Introduction

For this assignment, I focused on designing and implementing RESTful API endpoints for three core features of our EventsPark project. As the backend development lead for our team, I was responsible for creating the API infrastructure that would support our event booking and ticketing web application. The three core features I selected from our project proposal are Event Management, Seat Map Selection, and E-Ticket Generation with QR Code.

# Core Feature 1: Event Management

The Event Management feature enables organizers to create, manage, and publish events while providing users with comprehensive event discovery capabilities. This feature addresses the need for efficient event lifecycle management from creation to completion.

**API Endpoint Definitions**

POST /api/events

**Description:** Creates a new event with complete configuration including venue, seat map, and pricing tiers.

**HTTP Method:** POST

**Route Path:** /api/events

**Headers:**

Authorization: Bearer <JWT\_TOKEN>

Content-Type: application/json

**Request Body:**

{

"title": "Summer Music Festival 2024",

"description": "A fantastic summer music festival featuring top artists from around the world. Join us for an unforgettable evening of live music, food, and entertainment.",

"date": "2024-07-15T18:00:00.000Z",

"venue": {

"name": "Central Park Arena",

"address": "123 Main Street",

"city": "New York",

"capacity": 5000

},

"category": "Concert",

"priceTiers": [

{

"tier": "VIP",

"price": 150,

"description": "Premium seating with exclusive access to VIP lounge"

},

{

"tier": "Premium",

"price": 100,

"description": "Great view with comfortable seating"

},

{

"tier": "General",

"price": 75,

"description": "Standard seating with good view"

},

{

"tier": "Student",

"price": 50,

"description": "Student discount with valid ID"

}

],

"seatMap": [

{

"row": "A",

"number": 1,

"tier": "VIP"

},

{

"row": "A",

"number": 2,

"tier": "VIP"

},

{

"row": "B",

"number": 1,

"tier": "Premium"

},

{

"row": "B",

"number": 2,

"tier": "Premium"

}

],

"imageUrl": "https://example.com/event-images/summer-festival-2024.jpg",

"tags": ["music", "summer", "festival", "live"],

"status": "draft"

}

**Response Structure (201 Created):**

{

"success": true,

"message": "Event created successfully",

"data": {

"event": {

"\_id": "evt\_64a1b2c3d4e5f6g7h8i9j0",

"title": "Summer Music Festival 2024",

"description": "A fantastic summer music festival featuring top artists from around the world. Join us for an unforgettable evening of live music, food, and entertainment.",

"date": "2024-07-15T18:00:00.000Z",

"venue": {

"name": "Central Park Arena",

"address": "123 Main Street",

"city": "New York",

"capacity": 5000

},

"category": "Concert",

"organizer": {

"\_id": "usr\_64a1b2c3d4e5f6g7h8i9j1",

"name": "Event Organizer Pro"

},

"priceTiers": [

{

"tier": "VIP",

"price": 150,

"description": "Premium seating with exclusive access to VIP lounge"

},

{

"tier": "Premium",

"price": 100,

"description": "Great view with comfortable seating"

},

{

"tier": "General",

"price": 75,

"description": "Standard seating with good view"

},

{

"tier": "Student",

"price": 50,

"description": "Student discount with valid ID"

}

],

"seatMap": [

{

"row": "A",

"number": 1,

"tier": "VIP",

"isAvailable": true,

"isBooked": false

},

{

"row": "A",

"number": 2,

"tier": "VIP",

"isAvailable": true,

"isBooked": false

},

{

"row": "B",

"number": 1,

"tier": "Premium",

"isAvailable": true,

"isBooked": false

},

{

"row": "B",

"number": 2,

"tier": "Premium",

"isAvailable": true,

"isBooked": false

}

],

"totalSeats": 4,

"availableSeats": 4,

"status": "draft",

"imageUrl": "https://example.com/event-images/summer-festival-2024.jpg",

"tags": ["music", "summer", "festival", "live"],

"createdAt": "2024-12-01T10:00:00.000Z",

"updatedAt": "2024-12-01T10:00:00.000Z"

}

}

}

**Error Conditions:**

* 400 Bad Request: Invalid request data or missing required fields
* 401 Unauthorized: Invalid or missing authentication token
* 403 Forbidden: User not authorized to create events (requires organizer/admin role)
* 422 Validation Error: Data validation failures

**Error Response (403):**

{

"success": false,

"error": "Access denied",

"message": "You do not have permission to create events. Organizer or admin role required.",

"timestamp": "2024-12-01T10:00:00.000Z"

}

**GET /api/events**

**Description:** Retrieves events with advanced filtering, search, and pagination capabilities.

**HTTP Method:** GET

Route Path: /api/events

Query Parameters:

* page (optional, number): Page number for pagination (default: 1)
* limit (optional, number): Number of events per page (default: 10, max: 50)
* search (optional, string): Search events by title or description
* category (optional, string): Filter by event category
* status (optional, string): Filter by event status
* date (optional, string): Filter events on specific date (YYYY-MM-DD)
* startDate (optional, string): Start date for date range filtering
* endDate (optional, string): End date for date range filtering
* organizer (optional, string): Filter by organizer ID
* sortBy (optional, string): Sort field (date, title, price)
* sortOrder (optional, string): Sort order (asc, desc)

**Request Example:**

GET /api/events?category=Concert&status=published&page=1&limit=5&sortBy=date&sortOrder=asc

**Response Structure (200 Success):**

{

"success": true,

"message": "Events retrieved successfully",

"data": {

"events": [

{

"\_id": "evt\_64a1b2c3d4e5f6g7h8i9j0",

"title": "Summer Music Festival 2024",

"description": "A fantastic summer music festival featuring top artists",

"date": "2024-07-15T18:00:00.000Z",

"category": "Concert",

"venue": {

"name": "Central Park Arena",

"city": "New York",

"capacity": 5000

},

"availableSeats": 4850,

"totalSeats": 5000,

"priceTiers": [

{

"tier": "VIP",

"price": 150,

"description": "Premium seating with exclusive access"

},

{

"tier": "General",

"price": 75,

"description": "Standard seating"

}

],

"status": "published",

"organizer": {

"\_id": "usr\_64a1b2c3d4e5f6g7h8i9j1",

"name": "Event Organizer Pro"

},

"imageUrl": "https://example.com/event-images/summer-festival-2024.jpg",

"tags": ["music", "summer", "festival"]

}

],

"pagination": {

"currentPage": 1,

"totalPages": 5,

"totalItems": 50,

"hasNextPage": true,

"hasPrevPage": false,

"limit": 10

},

"filters": {

"category": "Concert",

"status": "published",

"sortBy": "date",

"sortOrder": "asc"

}

}

}

**GET /api/events/:id**

**Description:** Retrieves detailed information about a specific event including complete seat map and pricing.

**HTTP Method:** GET

**Route Path:** /api/events/:id

Path Parameters:

* id (required, string): Event ID

Request Example:

GET /api/events/evt\_64a1b2c3d4e5f6g7h8i9j0

Response Structure (200 Success):

{

"success": true,

"data": {

"event": {

"\_id": "evt\_64a1b2c3d4e5f6g7h8i9j0",

"title": "Summer Music Festival 2024",

"description": "A fantastic summer music festival featuring top artists from around the world. Join us for an unforgettable evening of live music, food, and entertainment.",

"date": "2024-07-15T18:00:00.000Z",

"venue": {

"name": "Central Park Arena",

"address": "123 Main Street",

"city": "New York",

"capacity": 5000

},

"category": "Concert",

"organizer": {

"\_id": "usr\_64a1b2c3d4e5f6g7h8i9j1",

"name": "Event Organizer Pro",

"email": "organizer@example.com"

},

"priceTiers": [

{

"tier": "VIP",

"price": 150,

"description": "Premium seating with exclusive access to VIP lounge"

},

{

"tier": "Premium",

"price": 100,

"description": "Great view with comfortable seating"

},

{

"tier": "General",

"price": 75,

"description": "Standard seating with good view"

},

{

"tier": "Student",

"price": 50,

"description": "Student discount with valid ID"

}

],

"seatMap": [

{

"row": "A",

"number": 1,

"tier": "VIP",

"isAvailable": true,

"isBooked": false

},

{

"row": "A",

"number": 2,

"tier": "VIP",

"isAvailable": false,

"isBooked": true

}

],

"totalSeats": 5000,

"availableSeats": 4850,

"status": "published",

"imageUrl": "https://example.com/event-images/summer-festival-2024.jpg",

"tags": ["music", "summer", "festival", "live"],

"createdAt": "2024-12-01T10:00:00.000Z",

"updatedAt": "2024-12-01T10:00:00.000Z"

}

}

}

**Error Conditions:**

* 404 Not Found: Event not found
* 400 Bad Request: Invalid event ID format

**PUT /api/events/:id**

**Description:** Updates an existing event. Only the event organizer or admin can modify events.

**HTTP Method:** PUT

**Route Path:** /api/events/:id

Headers:

Authorization: Bearer <JWT\_TOKEN>

Content-Type: application/json

**Request Body:**

{

"title": "Updated Summer Music Festival 2024",

"description": "Updated description for the summer music festival",

"status": "published",

"priceTiers": [

{

"tier": "VIP",

"price": 175,

"description": "Updated VIP experience"

}

]

}

**Response Structure (200 Success):**

{

"success": true,

"message": "Event updated successfully",

"data": {

"event": {

"\_id": "evt\_64a1b2c3d4e5f6g7h8i9j0",

"title": "Updated Summer Music Festival 2024",

"description": "Updated description for the summer music festival",

"status": "published",

"updatedAt": "2024-12-01T11:00:00.000Z"

}

}

}

**DELETE /api/events/:id**

**Description:** Deletes an event. Only the event organizer or admin can delete events.HTTP

Method: DELETE

Route Path: /api/events/:id

Headers:

Authorization: Bearer <JWT\_TOKEN>

# Core Feature 2: Seat Map Selection

The Seat Map Selection feature enables users to view and choose specific seats within a venue through an interactive visual interface. This feature addresses the challenge of manual seat selection and double-booking issues.

**API Endpoint Definitions**

**GET /api/events/:id/seats**

**Description**: Retrieves the complete seat map for a specific event with availability status.

**HTTP Method:** GET

**Route Path:** /api/events/:id/seats

Path Parameters:

* id (required, string): Event ID

Request Example:

GET /api/events/evt\_64a1b2c3d4e5f6g7h8i9j0/seats

Response Structure (200 Success):

{

  "success": true,

  "data": {

    "eventId": "evt\_64a1b2c3d4e5f6g7h8i9j0",

    "eventTitle": "Summer Music Festival 2024",

    "venue": {

      "name": "Central Park Arena",

      "capacity": 5000

    },

    "seatMap": [

      {

        "row": "A",

        "number": 1,

        "tier": "VIP",

        "price": 150,

        "isAvailable": true,

        "isBooked": false,

        "seatId": "seat\_A1\_001"

      },

      {

        "row": "A",

        "number": 2,

        "tier": "VIP",

        "price": 150,

        "isAvailable": false,

        "isBooked": true,

        "seatId": "seat\_A2\_002"

      },

      {

        "row": "B",

        "number": 1,

        "tier": "General",

        "price": 75,

        "isAvailable": true,

        "isBooked": false,

        "seatId": "seat\_B1\_003"

      }

    ],

    "statistics": {

      "totalSeats": 5000,

      "availableSeats": 4850,

      "bookedSeats": 150,

      "vipSeats": 500,

      "generalSeats": 4500

    },

    "priceTiers": [

      {

        "tier": "VIP",

        "price": 150,

        "description": "Premium seating with exclusive access",

        "availableCount": 450

      },

      {

        "tier": "General",

        "price": 75,

        "description": "Standard seating",

        "availableCount": 4400

      }

    ]

  }

}

**Error Conditions:**

* 404 Not Found: Event not found
* 400 Bad Request: Invalid event ID format
* 500 Internal Server Error: Database connection issues

**POST /api/bookings**

**Description:** Creates a new booking with selected seats and handles payment processing.

**HTTP Method:** POST

**Route Path:** /api/bookings

Headers:

Authorization: Bearer <JWT\_TOKEN>

Content-Type: application/json

**Request Body:**

{

  "eventId": "evt\_64a1b2c3d4e5f6g7h8i9j0",

  "seats": [

    {

      "row": "A",

      "number": 1,

      "tier": "VIP",

      "price": 150,

      "seatId": "seat\_A1\_001"

    },

    {

      "row": "B",

      "number": 1,

      "tier": "General",

      "price": 75,

      "seatId": "seat\_B1\_003"

    }

  ],

  "paymentMethod": "credit\_card",

  "paymentToken": "tok\_visa\_1234567890",

  "specialRequests": "Please provide wheelchair access if possible"

}

**Response Structure (201 Created):**

{

  "success": true,

  "message": "Booking created successfully",

  "data": {

    "booking": {

      "\_id": "bkg\_64a1b2c3d4e5f6g7h8i9j2",

      "user": {

        "\_id": "usr\_64a1b2c3d4e5f6g7h8i9j3",

        "name": "John Doe",

        "email": "john.doe@example.com"

      },

      "event": {

        "\_id": "evt\_64a1b2c3d4e5f6g7h8i9j0",

        "title": "Summer Music Festival 2024",

        "date": "2024-07-15T18:00:00.000Z"

      },

      "seats": [

        {

          "row": "A",

          "number": 1,

          "tier": "VIP",

          "price": 150,

          "seatId": "seat\_A1\_001"

        },

        {

          "row": "B",

          "number": 1,

          "tier": "General",

          "price": 75,

          "seatId": "seat\_B1\_003"

        }

      ],

      "totalAmount": 225,

      "status": "confirmed",

      "paymentStatus": "paid",

      "paymentMethod": "credit\_card",

      "paymentIntentId": "pi\_1234567890",

      "bookingDate": "2024-12-01T10:30:00.000Z",

      "specialRequests": "Please provide wheelchair access if possible"

    }

  }

}

**Error Conditions:**

* 400 Bad Request: Invalid request data or missing required fields
* 409 Conflict: Seat already booked by another user
* 402 Payment Required: Payment processing failed
* 401 Unauthorized: Invalid or missing authentication token
* 404 Not Found: Event not found

**Error Response (409):**

{

  "success": false,

  "error": "Seat already booked",

  "message": "One or more selected seats are no longer available. Please select different seats.",

  "conflictingSeats": ["seat\_A1\_001"],

  "timestamp": "2024-12-01T10:30:00.000Z"

}

# Core Feature 3: E-Ticket Generation with QR Code

The E-Ticket Generation feature creates digital tickets with QR codes upon successful booking, enabling secure event entry validation and eliminating the need for physical tickets.

**API Endpoint Definitions**

**POST /api/tickets/generate**

**Description:** Generates digital tickets with QR codes for a confirmed booking.HTTP **Method:** POST

**Route Path:**/api/tickets/generate

Headers:

Authorization: Bearer <JWT\_TOKEN>

Content-Type: application/json

**Request Body:**

{

  "bookingId": "bkg\_64a1b2c3d4e5f6g7h8i9j2"

}

Response Structure (201 Created):

json

Apply to EventsPark\_A...

{

  "success": true,

  "message": "Tickets generated successfully",

  "data": {

    "tickets": [

      {

        "\_id": "tkt\_64a1b2c3d4e5f6g7h8i9j4",

        "booking": "bkg\_64a1b2c3d4e5f6g7h8i9j2",

        "user": {

          "\_id": "usr\_64a1b2c3d4e5f6g7h8i9j3",

          "name": "John Doe",

          "email": "john.doe@example.com"

        },

        "event": {

          "\_id": "evt\_64a1b2c3d4e5f6g7h8i9j0",

          "title": "Summer Music Festival 2024",

          "date": "2024-07-15T18:00:00.000Z",

          "venue": {

            "name": "Central Park Arena",

            "address": "123 Main Street"

          }

        },

        "seat": {

          "row": "A",

          "number": 1,

          "tier": "VIP"

        },

        "ticketNumber": "TKT-20241201-12345",

        "qrCode": "...",

        "qrCodeData": "tkt\_64a1b2c3d4e5f6g7h8i9j4|usr\_64a1b2c3d4e5f6g7h8i9j3|evt\_64a1b2c3d4e5f6g7h8i9j0|A1|VIP",

        "price": 150,

        "status": "active",

        "issuedAt": "2024-12-01T10:30:00.000Z",

        "validUntil": "2024-07-16T18:00:00.000Z"

      },

      {

        "\_id": "tkt\_64a1b2c3d4e5f6g7h8i9j5",

        "booking": "bkg\_64a1b2c3d4e5f6g7h8i9j2",

        "user": {

          "\_id": "usr\_64a1b2c3d4e5f6g7h8i9j3",

          "name": "John Doe",

          "email": "john.doe@example.com"

        },

        "event": {

          "\_id": "evt\_64a1b2c3d4e5f6g7h8i9j0",

          "title": "Summer Music Festival 2024",

          "date": "2024-07-15T18:00:00.000Z",

          "venue": {

            "name": "Central Park Arena",

            "address": "123 Main Street"

          }

        },

        "seat": {

          "row": "B",

          "number": 1,

          "tier": "General"

        },

        "ticketNumber": "TKT-20241201-12346",

        "qrCode": "...",

        "qrCodeData": "tkt\_64a1b2c3d4e5f6g7h8i9j5|usr\_64a1b2c3d4e5f6g7h8i9j3|evt\_64a1b2c3d4e5f6g7h8i9j0|B1|General",

        "price": 75,

        "status": "active",

        "issuedAt": "2024-12-01T10:30:00.000Z",

        "validUntil": "2024-07-16T18:00:00.000Z"

      }

    ]

  }

}

**Error Conditions:**

* 400 Bad Request: Invalid booking ID or booking not found
* 409 Conflict: Tickets already generated for this booking
* 401 Unauthorized: Invalid authentication token
* 403 Forbidden: User not authorized to generate tickets for this booking

**POST /api/tickets/validate**

**Description:** Validates a ticket using QR code data for event entry.

**HTTP Method:** POST

**Route Path:** /api/tickets/validate

Headers:

Authorization: Bearer <JWT\_TOKEN>

Content-Type: application/json

**Request Body:**

{

  "qrCodeData": "tkt\_64a1b2c3d4e5f6g7h8i9j4|usr\_64a1b2c3d4e5f6g7h8i9j3|evt\_64a1b2c3d4e5f6g7h8i9j0|A1|VIP"

}

Response Structure (200 Success):

{

  "success": true,

  "data": {

    "valid": true,

    "ticket": {

      "\_id": "tkt\_64a1b2c3d4e5f6g7h8i9j4",

      "ticketNumber": "TKT-20241201-12345",

      "eventId": "evt\_64a1b2c3d4e5f6g7h8i9j0",

      "eventTitle": "Summer Music Festival 2024",

      "eventDate": "2024-07-15T18:00:00.000Z",

      "seat": {

        "row": "A",

        "number": 1,

        "tier": "VIP"

      },

      "user": {

        "name": "John Doe",

        "email": "john.doe@example.com"

      },

      "status": "active",

      "validUntil": "2024-07-16T18:00:00.000Z",

      "validationTime": "2024-12-01T10:30:00.000Z"

    }

  }

}

**Error Conditions:**

* 400 Bad Request: Invalid QR code data format
* 404 Not Found: Ticket not found
* 410 Gone: Ticket already used
* 403 Forbidden: Ticket expired
* 401 Unauthorized: Invalid authentication token

**Error Response (410):**

{

  "success": false,

  "error": "Ticket already used",

  "message": "This ticket has already been used for event entry",

  "ticketNumber": "TKT-20241201-12345",

  "usedAt": "2024-12-01T09:15:00.000Z",

  "timestamp": "2024-12-01T10:30:00.000Z"

}

# Security-Related Endpoints and Risk Mitigation

**Authentication Endpoints**

**POST /api/auth/register**

Description: Registers a new user account with role assignment.

**HTTP Method:** POST

**Route Path:** /api/auth/register

Request Body:

{

  "name": "John Doe",

  "email": "john.doe@example.com",

  "password": "SecurePassword123!",

  "role": "user",

  "phone": "+1234567890"

}

**Response Structure (201 Created):**

{

  "success": true,

  "message": "User registered successfully",

  "data": {

    "user": {

      "\_id": "usr\_64a1b2c3d4e5f6g7h8i9j3",

      "name": "John Doe",

      "email": "john.doe@example.com",

      "role": "user",

      "isActive": true

    },

    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiJ1c3JfNjRhMWIyYzNkNGU1ZjZnN2g4aTlqMyIsInJvbGUiOiJ1c2VyIiwiaWF0IjoxNzM1NzE5ODAwLCJleHAiOjE3MzU4MDUyMDB9.signature"

  }

}

**POST /api/auth/login**

**Description:** Authenticates user credentials and returns JWT token.

**HTTP Method:** POST

**Route Path:** /api/auth/login

**Request Body:**

{

  "email": "john.doe@example.com",

  "password": "SecurePassword123!"

}

Response Structure (200 Success):

{

  "success": true,

  "message": "Login successful",

  "data": {

    "user": {

      "\_id": "usr\_64a1b2c3d4e5f6g7h8i9j3",

      "name": "John Doe",

      "email": "john.doe@example.com",

      "role": "user"

    },

    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiJ1c3JfNjRhMWIyYzNkNGU1ZjZnN2g4aTlqMyIsInJvbGUiOiJ1c2VyIiwiaWF0IjoxNzM1NzE5ODAwLCJleHAiOjE3MzU4MDUyMDB9.signature"

  }

}

**Authorization Endpoints**

**GET /api/auth/me**

Description: Retrieves current user profile using JWT token.

HTTP Method: GET

Route Path: /api/auth/me

Headers:

Authorization: Bearer <JWT\_TOKEN>

**Response Structure (200 Success):**

{

  "success": true,

  "data": {

    "user": {

      "\_id": "usr\_64a1b2c3d4e5f6g7h8i9j3",

      "name": "John Doe",

      "email": "john.doe@example.com",

      "role": "user",

      "phone": "+1234567890",

      "isActive": true,

      "createdAt": "2024-12-01T10:00:00.000Z"

    }

  }

}

## Security Risk Mitigation

**Injection Attack Prevention:**

* All user inputs are validated using express-validator library
* MongoDB queries use Mongoose ODM which prevents NoSQL injection
* Parameterized queries ensure SQL injection protection
* Input sanitization removes malicious scripts and characters

**Broken Access Control Prevention:**

* JWT tokens contain user role information for authorization
* Role-based middleware validates user permissions before endpoint access
* Resource ownership validation ensures users can only access their own data
* Token expiration and refresh mechanisms prevent session hijacking

**Additional Security Measures:**

* Password hashing using bcryptjs with salt rounds of 10
* Rate limiting prevents brute force attacks
* CORS configuration restricts cross-origin requests
* HTTPS enforcement for all API communications

# Data Integration and Modeling

**Entity-Relationship Diagram (ERD)**

The database schema consists of four main entities with the following relationships:

User (1) --- (M) Booking (M) --- (1) Event

Booking (1) --- (M) Ticket

Event (1) --- (M) Ticket

# Data Models

**User Entity**

{

  \_id: ObjectId (required, primary key),

  name: String (required, max 50 characters),

  email: String (required, unique, validated format),

  password: String (required, min 6 characters, hashed),

  role: String (required, enum: ['user', 'organizer', 'admin']),

  phone: String (optional, validated format),

  isActive: Boolean (default: true),

  resetPasswordToken: String (optional),

  resetPasswordExpire: Date (optional),

  createdAt: Date (auto-generated),

  updatedAt: Date (auto-updated)

}

**Event Entity**

{

  \_id: ObjectId (required, primary key),

  title: String (required, max 100 characters),

  description: String (required, max 1000 characters),

  date: Date (required, future date validation),

  venue: {

    name: String (required),

    address: String (required),

    city: String (required),

    capacity: Number (required, min 1)

  },

  category: String (required, enum: categories),

  organizer: ObjectId (required, ref: User),

  priceTiers: [{

    tier: String (required, enum: ['VIP', 'Premium', 'General', 'Student']),

    price: Number (required, min 0),

    description: String (optional)

  }],

  seatMap: [{

    row: String (required),

    number: Number (required),

    tier: String (required),

    isAvailable: Boolean (default: true),

    isBooked: Boolean (default: false)

  }],

  totalSeats: Number (required),

  availableSeats: Number (required),

  status: String (required, enum: ['draft', 'published', 'cancelled', 'completed']),

  imageUrl: String (optional),

  tags: [String] (optional),

  createdAt: Date (auto-generated),

  updatedAt: Date (auto-updated)

}

**Booking Entity**

{

  \_id: ObjectId (required, primary key),

  user: ObjectId (required, ref: User),

  event: ObjectId (required, ref: Event),

  seats: [{

    row: String (required),

    number: Number (required),

    tier: String (required),

    price: Number (required, min 0)

  }],

  totalAmount: Number (required, min 0),

  status: String (required, enum: ['pending', 'confirmed', 'cancelled', 'refunded']),

  paymentStatus: String (required, enum: ['pending', 'paid', 'failed', 'refunded']),

  paymentMethod: String (required, enum: ['credit\_card', 'debit\_card', 'paypal', 'stripe']),

  paymentIntentId: String (optional),

  bookingDate: Date (default: current timestamp),

  specialRequests: String (optional, max 500 characters),

  cancellationReason: String (optional),

  refundAmount: Number (default: 0),

  createdAt: Date (auto-generated),

  updatedAt: Date (auto-updated)

}

**Ticket Entity**

{

  \_id: ObjectId (required, primary key),

  booking: ObjectId (required, ref: Booking),

  user: ObjectId (required, ref: User),

  event: ObjectId (required, ref: Event),

  seat: {

    row: String (required),

    number: Number (required),

    tier: String (required)

  },

  ticketNumber: String (required, unique),

  qrCode: String (required, base64 encoded image),

  qrCodeData: String (required, encrypted ticket data),

  price: Number (required, min 0),

  status: String (required, enum: ['active', 'used', 'cancelled', 'expired']),

  issuedAt: Date (default: current timestamp),

  usedAt: Date (optional),

  cancelledAt: Date (optional),

  cancellationReason: String (optional),

  validUntil: Date (required),

  createdAt: Date (auto-generated),

  updatedAt: Date (auto-updated)

}

# CRUD Operations Mapping

**User Operations:**

* Create: POST /api/auth/register
* Read: GET /api/auth/me
* Update: PUT /api/auth/profile
* Delete: PUT /api/auth/deactivate

**Event Operations:**

* Create: POST /api/events (organizer/admin only)
* Read: GET /api/events, GET /api/events/:id
* Update: PUT /api/events/:id (organizer/admin only)
* Delete: DELETE /api/events/:id (organizer/admin only)

**Booking Operations:**

* Create: POST /api/bookings
* Read: GET /api/bookings/my-bookings
* Update: PUT /api/bookings/:id/cancel
* Delete: Not applicable (soft delete via cancellation)

**Ticket Operations:**

* Create: POST /api/tickets/generate
* Read: GET /api/tickets/my-tickets, GET /api/tickets/:id
* Update: PUT /api/tickets/:id/use, PUT /api/tickets/:id/cancel
* Delete: Not applicable (soft delete via cancellation)

# Implementation and Demonstration

For this assignment, I implemented the **Event Management** feature as my primary demonstration, focusing on two core API endpoints: **Create Event** and **Show Events**. This feature was chosen because it represents the foundation of our event booking system and demonstrates comprehensive CRUD operations, role-based access control, and complex data validation.

## Implementation Details

**Technology Stack:**

* **Backend Framework:** Node.js with Express.js for robust server-side development
* **Database:** MongoDB with Mongoose ODM for flexible document storage
* **Authentication:** JWT (JSON Web Tokens) for secure user sessions
* **Validation:** Express-validator for comprehensive input validation
* **Security:** bcryptjs for password hashing with salt rounds
* **Deployment:** Render Cloud Platform for reliable hosting

**Chosen Endpoints for Implementation:**

1. **POST /api/events** - Create new event with complete configuration
2. **GET /api/events** - Retrieve events with advanced filtering and pagination

## Live Deployment

**Deployment Platform:** Render Cloud Platform

**Live API URL:** <https://eventspark-api.onrender.com>

**Health Check Endpoint:** <https://eventspark-api.onrender.com/api/health>

**Source Code Repository:** <https://github.com/hsbzmn7/eventspark-api>

The application is successfully deployed and accessible 24/7. The health check endpoint confirms the API is running and database connections are active.

### Endpoint 1: Create Event (POST /api/events)

**Implementation Overview**

This endpoint enables event organizers to create comprehensive event listings with detailed configurations including venue information, seat maps, pricing tiers, and metadata. The implementation includes role-based access control ensuring only authorized users can create events.

**Technical Implementation**

The endpoint validates user authentication and authorization, processes complex nested data structures, generates seat maps automatically, and ensures data integrity through comprehensive validation rules. The system calculates total and available seats based on the provided seat map configuration.

**Key Features Demonstrated**

* **Role-based Access Control:** Only organizers and admins can create events
* **Complex Data Validation:** Validates nested objects, arrays, and business rules
* **Automatic Calculations:** Computes seat statistics and availability
* **Error Handling:** Comprehensive error responses with meaningful messages
* **Data Persistence:** Stores complete event data in MongoDB

**Testing Evidence**

**Successful Creation Response (201):**

The endpoint successfully creates events with all required fields and returns a comprehensive response including the generated event ID, calculated seat statistics, and complete event details.

**Error Scenarios Tested:**

* **401 Unauthorized:** Attempting to create event without authentication token
* **403 Forbidden:** Regular user attempting to create event without organizer privileges
* **400 Bad Request:** Invalid data format or missing required fields
* **422 Validation Error:** Business rule violations (e.g., past event dates)

### Endpoint 2: Show Events (GET /api/events)

**Implementation Overview**

This endpoint provides comprehensive event discovery capabilities with advanced filtering, search functionality, and pagination. It supports multiple query parameters for flexible event retrieval and includes real-time availability information.

**Technical Implementation**

The endpoint implements MongoDB aggregation pipelines for efficient querying, supports multiple filter combinations, provides pagination with metadata, and includes search functionality across event titles and descriptions. The system maintains real-time seat availability calculations.

**Key Features Demonstrated**

* **Advanced Filtering:** Category, status, date range, and organizer filtering
* **Search Functionality:** Text-based search across event content
* **Pagination:** Efficient data retrieval with page metadata
* **Real-time Data:** Current seat availability and booking statistics
* **Flexible Querying:** Multiple parameter combinations supported

**Testing Evidence**

**Successful Retrieval Response (200):**

The endpoint successfully retrieves events with applied filters and returns structured data including event details, pagination information, and applied filter metadata.

**Query Parameter Testing:**

* **Category Filtering:** Filtering events by specific categories
* **Date Range Filtering:** Retrieving events within date ranges
* **Status Filtering:** Filtering by event status (draft, published, etc.)
* **Pagination Testing:** Testing different page sizes and navigation
* **Search Functionality:** Text-based event discovery

### Database Persistence Evidence

**MongoDB Atlas Implementation**

The application successfully persists all event data in MongoDB Atlas cloud database. The database schema includes comprehensive collections for users, events, bookings, and tickets with proper relationships and indexing.

**Database Collections**

1. **Users Collection:** Stores user accounts with role-based access control
2. **Events Collection:** Contains complete event data with seat configurations
3. **Bookings Collection:** Tracks booking transactions and seat reservations
4. **Tickets Collection:** Stores digital tickets with QR code data

**Data Integrity Features**

* **Referential Integrity:** Proper relationships between collections
* **Data Validation:** Schema-level validation ensuring data quality
* **Indexing:** Optimized database performance with strategic indexes
* **Real-time Updates:** Automatic seat availability calculations

**Authentication and Security Testing**

**JWT Authentication Flow**

The implementation includes comprehensive JWT-based authentication with proper token generation, validation, and expiration handling. All protected endpoints require valid authentication tokens.

**Role-Based Access Control**

The system implements granular role-based access control ensuring users can only perform actions appropriate to their assigned roles (user, organizer, admin).

**Security Testing Results**

* **Successful Authentication:** Valid login with token generation
* **Token Validation:** Protected endpoint access with valid tokens
* **Authorization Testing:** Role-based permission enforcement
* **Security Failures:** Proper handling of invalid tokens and unauthorized access

**Error Handling and Validation**

**Comprehensive Error Responses**

The implementation includes detailed error handling with appropriate HTTP status codes, meaningful error messages, and helpful debugging information.

**Validation Testing**

* **Input Validation:** Comprehensive data format and business rule validation
* **Authentication Errors:** Proper handling of missing or invalid tokens

A screenshot of a computer

AI-generated content may be incorrect.

Fig 1: If we go with a wrong token we are getting this response

* **Authorization Errors:** Clear messaging for insufficient permissions
* **Resource Errors:** Appropriate responses for missing or invalid resources

**Performance and Scalability**

**Response Time Optimization**

The implemented endpoints demonstrate efficient response times with proper database indexing and query optimization. The pagination system ensures scalable data retrieval for large datasets.

**Database Performance**

* **Indexed Queries:** Strategic database indexing for fast retrieval
* **Aggregation Pipelines:** Efficient MongoDB aggregation for complex queries
* **Connection Pooling:** Optimized database connection management
* **Caching Considerations:** Architecture supports future caching implementation

**Testing Workflow and Evidence**

**Postman Testing Methodology**

I conducted comprehensive testing using the automated Postman collection that includes:

* **Automated Token Management:** JWT tokens automatically extracted and used
* **Response Data Extraction:** Test data saved to variables for subsequent requests
* **Complete Workflow Testing:** End-to-end testing from registration to event management
* **Error Scenario Coverage:** Comprehensive error case testing

### Required Screenshots for Assignment

The testing evidence includes annotated screenshots demonstrating:

A screenshot of a computer

AI-generated content may be incorrect.

Fig 2: Successful Event Creation: Complete event creation with all fields

A screenshot of a computer

AI-generated content may be incorrect.

Fig 3: Event Retrieval with Filtering: Multiple filter combinations tested

A screenshot of a computer

AI-generated content may be incorrect.

Fig 4: Database Evidence: MongoDB Atlas screenshots showing persisted data

# Conclusion

The implementation successfully demonstrates the Event Management feature with two fully functional API endpoints deployed on a public cloud platform. The solution includes comprehensive authentication, authorization, validation, and error handling while maintaining excellent performance and scalability characteristics. The live deployment at https://eventspark-api.onrender.com provides concrete evidence of the implementation's functionality, while the comprehensive testing documentation ensures the solution meets all assignment requirements. The database persistence evidence confirms that the system properly stores and manages event data with real-time availability calculations. This implementation showcases my ability to design and deploy production-ready RESTful APIs with proper security measures, comprehensive testing, and real-world functionality that addresses the core requirements of event management systems.