Event Ticket Booker – Backend Development Assignment

Introduction

In this assignment, you will build a complete event ticket booking system using Node.js, Express, and MongoDB. This project will test your ability to create a production-quality backend API that handles user authentication, event management, ticket booking, and payment processing.

Your task is to implement a system that allows users to register events, book tickets, process payments, and manage the entire event lifecycle. This project will assess your backend development skills in a real-world scenario

Learning Objectives

By completing this project, you will demonstrate your ability to:

- Design and build a RESTful API with proper architecture patterns
- Implement secure authentication and authorization systems
- Create complex database models with appropriate relationships
- Integrate third-party services (payment processing)
- Handle edge cases and implement proper error management
- Organize code in a maintainable, deployment-ready structure

Project Requirements

1. Project Setup and Authentication

Requirements:

- ☐ Set up the project with a clean, organized structure (models, routes, controllers, middleware)
- ☐ Implement a User model that supports multiple roles (admin, organizer, attendee)
- ☐ Create a user registration endpoint that validates inputs and allows role selection (bw organizer & attendee)
- ☐ Build a secure login endpoint that generates JWT tokens
- ☐ Implement password hashing using bcrypt
- ☐ Create middleware to protect routes based on authentication status
- ☐ Implement role-based authorization middleware
- ☐ Build a complete password reset flow with token-based verification

Evaluation Criteria:

- Security of authentication implementation
- Proper validation of user inputs
- Correct implementation of JWT-based authentication
- Effective role-based permission handling

Deliverable: Complete authentication system with user registration, login, and role-based access control.

2. Event Management

Event Modeling

- [ ] Create a comprehensive Event model with the following fields:
  - Basic info: title, description, category
  - Logistics: location, date, time
  - Capacity tracking: total tickets, available tickets
  - Status management: draft, published, cancelled
  - Media: poster image URL
  - Pricing information
  - Relationships: organizer reference (User model)

Organizer Features

- [ ] Implement a protected endpoint for event creation (organizers only)
- [ ] Create endpoints for updating event details (with proper ownership validation)
- [ ] Build event deletion functionality with appropriate checks
- [ ] Implement an endpoint to list events created by the authenticated organizer
- [ ] Create functionality to toggle event status (draft → published → cancelled)

Attendee Features

- [ ] Build a public endpoint to list all published events
- [ ] Implement pagination for event listings
- [ ] Create search functionality by title/description
- [ ] Implement a filtering system by category, price range, date, location
- [ ] Create an endpoint to view complete details of a single event

Evaluation Criteria:

- Proper database modeling and relationships
- Implementation of ownership validation
- Effective search and filtering implementation
- API design and organization

Deliverable: Complete event management system with CRUD operations and filtering capabilities.

3. Ticket Booking System

Ticket Modeling

- [ ] Design a Booking/Ticket model that includes:
  - References to User (attendee) and Event
  - Quantity of tickets booked
  - Total price calculation
  - Status tracking (reserved, paid, cancelled)
  - Unique ticket reference number
  - Appropriate timestamp data

Booking Functionality

- [ ] Create an endpoint for reserving tickets
- [ ] Implement validation to check ticket availability before booking
- [ ] Build logic to prevent duplicate bookings by the same user
- [ ] Create a system to generate unique ticket reference numbers
- [ ] Implement an endpoint for users to view their booked tickets
- [ ] Add functionality to cancel a ticket (with appropriate business rules)
- [ ] Create logic to update available ticket counts when bookings are made/cancelled

Evaluation Criteria:

- Concurrency handling in the booking process
- Validation of business rules
- Unique identifier generation
- Relationship management between models

Deliverable: Functional ticket booking system with validation and reference generation.

4. Payment Integration

Payment Setup

- [ ] Set up integration with Paystack payment gateway
- [ ] Implement a payment initialization endpoint
- [ ] Create a webhook or verification endpoint for payment confirmation

Payment Flow

- [ ] Connect payment flow to the ticket booking process
- [ ] Update ticket status based on payment outcome
- [ ] Implement a bypass flow for free events
- [ ] Add comprehensive error handling for payment failures
- [ ] Store payment receipts and link them to bookings
- [ ] Create an endpoint to view payment history for a booking

Evaluation Criteria:

- Successful integration with external payment service
- Proper error handling for payment scenarios
- Security of payment information
- Transaction management

Deliverable: Complete payment flow integration with Paystack, supporting both paid and free events.

5. Admin Features and Notifications

Admin Dashboard Endpoints

- [ ] Create admin-only endpoints to view and manage all users
- [ ] Build endpoints to view/manage all events across organizers
- [ ] Implement an event approval workflow (optional moderation layer)
- [ ] Create admin endpoints for viewing all bookings with filtering options

- [ ] Build basic reporting endpoints (tickets sold, revenue generated)

Notification System

- [ ] Set up email service integration using nodemailer
- [ ] Create a booking confirmation email template
- [ ] Implement email sending logic for successful bookings
- [ ] Build a reminder system for upcoming events
- [ ] (Advanced) Set up scheduled jobs for automated reminders

Evaluation Criteria:

- Implementation of admin privileges
- Report generation functionality
- Email template design and delivery
- Scheduled task management

Deliverable: Admin management capabilities and email notification system.

6. API Refinements and Documentation

API Improvements

- [ ] Add comprehensive input validation to all endpoints
- [ ] Implement consistent error handling across the application
- [ ] Create a standardized API response format
- [ ] Add rate limiting to prevent API abuse
- [ ] Implement a request logging system

Documentation and Deployment Preparation

- [ ] Create Swagger/OpenAPI documentation for all endpoints
- [ ] Build a Postman collection with example requests
- [ ] Document database schema and relationships
- [ ] Prepare environment configuration for deployment
- [ ] Write setup instructions for local development

Evaluation Criteria:

- Quality and completeness of documentation
- Consistency of API responses
- Implementation of security measures
- Code organization and readiness for deployment

Deliverable: Production-ready API with documentation and deployment preparation.

Technical Specifications

Required Project Structure

Organize your code using this structure:

```
src/
  ├── config/            # Configuration files
  ├── controllers/       # Request handlers
  ├── middleware/        # Custom middleware
  ├── models/            # Database models
  ├── routes/            # API routes
  ├── services/          # Business logic
  ├── utils/             # Helper functions
  └── app.js             # App entry point
```

Evaluation Criteria

Your project will be evaluated based on:

1. **Code organization and architecture** - How well is your code structured?
2. **Security practices** - Have you implemented proper authentication, authorization, and data protection?
3. **Database design** - Are your models and relationships appropriate for the problem?
4. **Error handling** - Does your application handle errors gracefully?
5. **API documentation** - Is your API well-documented and easy to understand?
6. **Feature completeness** - Have you implemented all required functionality?
7. **Code quality** - Is your code readable, maintainable, and following best practices?

Resources

- Express.js documentation: https://expressjs.com/
- Mongoose documentation: https://mongoosejs.com/docs/
- JWT authentication: https://jwt.io/
- Paystack API documentation: https://paystack.com/docs/api/
- Nodemailer for sending emails: https://nodemailer.com/

Submission Guidelines

Submit your completed project as a GitHub repository with:

1. Complete source code
2. README with setup instructions
3. API documentation
4. Environment variables template (.env.example)
5. Any additional documentation explaining your implementation decisions