DongGyun Kim

Asite 3D Repo Node.js Backend Developer Assessment Document

This document summarizes my solution for the Asite 3D Repo Backend Developer Assessment, covering API endpoints, tech stack, and technical overview. To enhance clarity, I used tables for API specs, testing, and project structure.

Technology Stack

Backend Framework	Express.js
Database	MongoDB with Mongoose ODM (MongoDB Atlas)
Testing	Jest

Project Structure

Directory/File	Purpose
src/controllers	API endpoint handlers
src/models	MongoDB schemas and data models
src/routes	API route definitions
src/tests	Test files for the application
src/utils	Utility functions and helpers
src/index.js	Application entry point
src/server.js	Express server configuration
.env	Environment variables

Database Schema

Event Schema

Field	Type	Properties
name	String	required: true
date	Date	required: true, unique: true
capacity	Number	required: true, min: 1
costPerTicket	Number	required: true, min: 0
TicketsSold	Number	default: 0
createdAt	Date	default: Date.now

Ticket Schema

Field	Type	Properties
event	ObjectId	ref: "Event", required: true
nTickets	Number	required: true, min: 1
totalCost	Number	required: true
transactionDate	Date	default: Date.now

API Documentation

1. Add a New Event

Endpoint	POST /events
Controller	controllers/eventController.js
Request Body	{"name": "Charity Auction", "date": "31/10/2024", "capacity": 100, "costPerTicket": 5 }
Response	Returns the created event with its unique identifier
Validation	Ensures no other event exists on the same date

2. Add Ticket Transaction

Endpoint	POST /tickets
Controller	controllers/ticketController.js
Request Body	{ "event": "event_id_here", "nTickets": 3 }
Response	Confirmation of the transaction
Validation	Verifies event exists and has sufficient capacity

3. Retrieve Ticket Sales Statistics

Endpoint	GET /stat
Controller	controllers/statsController.js
Response	Monthly statistics for the past 12 months in JSON format
Example	[{ "year": 2024, "month": 9, "revenue": 10203, "nEvents": 10, "averageTicketsSold": 40 }, ...]

Testing

Test Type	Description	File
Unit Tests	Tests for date validation and conversion functions	src/tests/unit.test.js
System Tests	End-to-end API tests including event creation, ticket purchases, and statistics retrieval	src/tests/system.test.js

Installation and Setup

1.Open the repository

2.Install dependencies: npm install

3. .env (file provided)

4.Start the server: npm start or node src/index.js

5.Run tests: npm test

Development Approach and Additional Considerations

The current architecture focuses on simplicity and maintainability while allowing for future expansion

Architectural Decisions

- The app uses a simple structure with controllers, models, routes, and utility files.
- This structure can be refactored later to add service layers if the project grows bigger.

Session Management (for SSO Integration)

- If Single Sign-On (SSO) was required, I would implement secure authentication using either session-based or JWT-based methods, depending on the system requirements.
- For session-based auth, I would set up cookies with HttpOnly, Secure, and SameSite flags to manage session state safely.

Testing Environment

- The provided test setup used an in-memory MongoDB, but I chose MongoDB Atlas to make it easier to check data and debug.
- For production, I would use Docker to keep the environment consistent for development and testing.

Test Coverage

- Right now, test coverage is minimal because of the short time.
- In a real project, I would add:
  - more detailed unit tests , more integration tests, performance testing for higher traffic

Security Measures

- I would improve security by adding:
  - rate limiting and throttling
  - input sanitization
  - secure HTTP headers (like using Helmet) to protect against common attacks like injection.

Error Handling Strategy

- I would create a clear error-handling method.
- It would cover validation errors, server errors, and database errors, and return them in a consistent JSON format.

CI/CD & Deployment

- I would plan simple CI/CD pipelines (for example, using GitHub Actions) to automate testing and deployments.
- I would also set up safe environment variable management.

Future Enhancements

- Add authentication middleware (JWT)
- Add user authorization
- Make API documentation with Swagger
- Add a config folder for settings
- Add logging for events and errors
- Set up Docker for easier development and deployment

Note: The MongoDB Atlas instance I used for development and testing will stay available until the assessment review is done.