Event Booking API Implementation Report

September 22, 2025

1 Introduction

This report outlines the implementation of a TypeScript-based Event Booking API, designed to manage events, retrieve event availability, and process ticket purchases. The system uses Express.js for the server, Mongoose for MongoDB interactions, and TypeScript for type safety. The implementation follows a layered architecture with separate modules for models, services, controllers, routes, and database connections.

2 Project Overview

The Event Booking API provides RESTful endpoints to create events, list all events, check seat availability, and purchase tickets. The system is built with modularity, type safety, and robust error handling in mind, using MongoDB as the database.

2.1 Key Features

- **Create Event**: Create events with a name and sections (containing rows with seat information).
- **List Events**: Retrieve all events stored in the database.
- **Event Availability**: Check available seats for a specific event, organized by sections and rows.
- **Purchase Tickets**: Purchase tickets for specific sections and rows, with validation for seat availability and group discounts (applied for 4+ tickets).

3 Technical Details

The project is organized into TypeScript files to ensure separation of concerns and maintainability.

3.1 Project Structure

- `src/index.ts`: Initializes the Express server, sets up middleware (JSON, CORS), and mounts routes.
- `src/db/db.ts`: Establishes a MongoDB connection using Mongoose.

- `src/routes/event.routes.ts`: Defines API routes for event operations.
- `src/controllers/event.controller.ts`: Handles HTTP requests and responses, delegating logic to the service layer.
- `src/services/event.service.ts`: Implements business logic for event creation, retrieval, availability, and ticket purchases.
- `src/models/event.models.ts`: Defines Mongoose schemas and TypeScript interfaces for events, sections, and rows.
- `package.json`: Lists dependencies and scripts for building and running the application.
- `tsconfig.json`: Configures TypeScript compilation settings.
- `.env`: Stores environment variables (`MONGO_URI`, `PORT`).

3.2 Implementation Details

- **Database Schema**: The `Event` model includes a name, sections (with name and rows), and rows (with name, total seats, and booked seats). Timestamps (`createdAt`, `updatedAt`) are enabled.
- **Service Layer**: The `EventService` class handles business logic, including input validation, database operations, and seat availability calculations. It throws specific errors for invalid inputs or unavailable resources.
- **Controller Layer**: The `EventController` class processes HTTP requests, maps service errors to appropriate HTTP status codes (e.g., 400 for validation errors, 404 for not found, 500 for server errors), and returns JSON responses.
- **Routes**: Endpoints include:
  - `POST /events`: Create a new event.
  - `GET /events`: Retrieve all events.
  - `GET /events/:id/availability`: Get seat availability for an event.
  - `POST /events/:id/purchase`: Purchase tickets for an event.
- **Server**: The Express server runs on port 8888 (configurable via `.env`), with CORS enabled and JSON parsing for request bodies.

3.3 Dependencies

The project uses the following npm packages:

- **Runtime**: `express`, `cors`, `mongoose`, `dotenv`.
- **Development**: `typescript`, `ts-node`, `@types/express`, `@types/cors`, `@types/mongoose`, `@types/node`, `eslint`, `@typescript-eslint/parser`, `@typescript-eslint/eslint-plugin`.

4 Setup Instructions

1. Install dependencies: `npm install`.

2. Create a `.env` file with:

```
MONGO_URI=mongodb://localhost:27017/event_booking
PORT=8888
```

3. Compile TypeScript: `npx tsc`.
4. Start the server: `npm start` (or `npm run dev` for development with `ts-node`).
5. Ensure MongoDB is running locally or provide a valid `MONGO_URI`.

5 Conclusion

The Event Booking API provides a scalable, type-safe solution for managing events and ticket purchases. The modular architecture, with distinct layers for models, services, controllers, and routes, ensures maintainability and extensibility. Future enhancements could include user authentication, additional validation, and unit tests.