Booking App for Sports Technology Company

Aditya Srivastava (IEC2021031)

1. Introduction The Sports Facility Booking System is a web application designed to manage reservations for various sports facilities. It supports two user roles: regular users and administrators. The backend is built with Node.js, Mongoose, TypeScript, Express, and Zod for data validation, providing a robust and reliable system. JWT-based authentication and role-based access control ensure secure access to features. The system also uses ESLint and Prettier for maintaining code quality and consistency in formatting.

3. Implementation Details

Technologies Used:

1. **Backend Architecture:**

- The backend architecture was designed to be modular and scalable. Using **Node.js** with **Express.js** for the server ensures a structured API system that can handle multiple routes for different functionalities such as managing facilities, bookings, and user data. **Mongoose** is used to interact with the **MongoDB** database, which provides flexibility and scalability for storing information about facilities, bookings, and users.
- **Zod** is used for data validation on both the client and server sides, ensuring that incoming data adheres to strict types and formats, improving system reliability and minimizing errors.

2. **Frontend:** React.js for building the user interface.

2. Features

- **User Registration and Authentication: Users can sign up and log in using their credentials.**
- **Role-Based Access Control (User and Admin): Users and admins have different levels of access based on their roles.**
- **CRUD Operations for Bookings: The system allows creating, updating, and canceling bookings.**
- **JWT-Based Authentication: Secure authentication using JWT tokens.**
- **Facility Availability Check: Users can check the availability of facilities before making a booking.**
- **User Profile Management: Users can manage their profiles and view their booking history.**

3. Tech Stack

- **Backend: Node.js, Express**
- **Database: MongoDB, Mongoose**
- **Language: TypeScript**
- **Data Validation: Zod**
- **Authentication: JWT**
- **Linting & Formatting: ESLint, Prettier**

4. API Endpoints

Here's a brief overview of the core API endpoints:

- **Auth:**
  - `POST /api/auth/signup` **– Register a new user**
  - `POST /api/auth/login` **– User login**
- **Facility:**
  - `POST /api/facility` **– Create a facility (Admin only)**
  - `PUT /api/facility/:id` **– Update a facility (Admin only)**
  - `GET /api/facility` **– Get all facilities**
  - `DELETE /api/facility/:id` **– Soft delete a facility by ID (Admin only)**
- **Bookings:**
  - `GET /api/check-availability` **– Check the availability of slots for a specific date**
  - `POST /api/bookings` **– Create a booking (User only)**
  - `GET /api/bookings` **– Get all bookings (Admin only)**
  - `GET /api/bookings/user` **– Get user-specific bookings (User only)**
  - `DELETE /api/bookings/:id` **– Cancel a booking by ID (User only)**

5. User Roles

- **User:**
  - **Can create and manage personal bookings.**
  - **Can view available facilities.**
- **Admin:**
  - **Can manage all users and bookings.**
  - **Can add, remove, and update facilities.**
  - **Has full access to all user functionalities.**

5. Frontend and Backend Integration:

The integration between frontend and backend is managed through REST APIs with JSON as the data format. The frontend, developed using React, makes HTTP requests to the backend to handle user authentication, bookings, and other functionalities. The choice of React ensures a responsive user interface, while the backend handles all the business logic.

3. API Design:

The API endpoints are designed with REST principles, ensuring clear separation between resource management and actions. CRUD operations are enabled for bookings and facilities, allowing admins to fully manage the system. A GET request for checking facility availability was added to ensure users can see if their desired time slot is free before making a booking.

4. Scalability:

The use of MongoDB provides a highly scalable database solution, which can handle a growing number of users, facilities, and bookings. The non-relational nature of MongoDB ensures flexibility in how the data is structured, allowing new features or data types to be added with minimal impact on existing functionality. The backend is structured to accommodate horizontal scaling in the future.

4. Challenges and Solutions

- **Challenge 1: Real-Time Slot Management**
    - **Solution:** Implemented server-side validation in booking.controller.js to check slot availability before creating a new booking, preventing overlaps.
- **Challenge 2: Integration between Backend and Frontend**
    - **Solution:** Ensured consistent use of JSON format for responses across all APIs and used Cors middleware to resolve cross-origin issues.
- **Challenge 3: Meeting the Deadline**
    - **Solution:** Managed time effectively through regular commits, clear task prioritization, and focused sprints to meet project requirements and complete the integration on time.

5. Future Improvements

- **Scalability and High Load Management:** To handle increased traffic and bookings, incorporating a Content Delivery Network (CDN) and planning for a high-level design with load balancers, distributed databases, and horizontal scaling will be essential.
- **Locking Mechanism:** Introduce a locking mechanism to prevent race conditions when multiple users attempt to book the same slot simultaneously. Techniques like optimistic or pessimistic locking at the database level will help maintain data consistency.

- **Real-Time Booking Updates:** Implement WebSockets or similar technology to provide real-time updates on slot bookings, ensuring instant notifications for customers and operations teams.
- **Admin Dashboard:** Develop a comprehensive operations management dashboard with detailed reporting, analytics, and activity logs for better oversight of booking patterns and resource utilization.
- **Payment Integration:** Integrate a secure payment gateway to allow customers to make payments directly through the application, streamlining the booking process and enhancing user convenience.
- **Authentication with Role-Based Access Control (RBAC):** Enhance the authentication system by implementing RBAC to ensure appropriate access levels for different user types (e.g., customers, center managers, admins).

Conclusion This project has successfully developed a booking system capable of managing complex booking needs for a sports technology company. Utilizing Node.js, MongoDB, and React.js, the system is scalable and prepared for future feature enhancements.