Sports Center Booking System

Anshul Bhardwaj(IIT2021057@iiita.ac.in)

Introduction

The Sports Booking Application is a backend system designed to streamline the process of reserving sports facilities across multiple centers. This robust platform caters to two primary user roles: normal users and center managers, each with distinct capabilities tailored to their needs.

Key features of the application include:

- Multi-center support for various sports and courts
- Role-based access control for users and managers
- Booking creation, viewing, and cancellation for users
- Comprehensive booking management for center managers
- Double-booking prevention to ensure court availability

This system aims to enhance the user experience in sports facility booking while providing efficient management tools for center administrators.

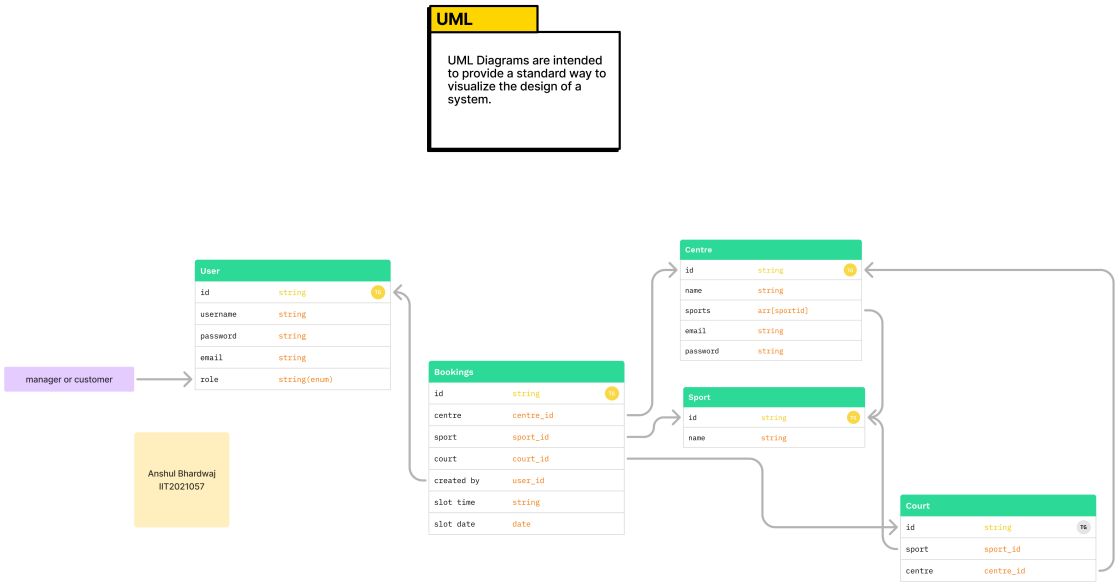
Design Decisions

1. Role Based Access Control

The application differentiates between two types of users: customers and managers. Customers can only interact with their own bookings, while managers can view and manage all bookings within their center.

2. Database Design:

MongoDB: Chosen for its flexibility in handling complex relationships between centers, courts, sports, and users. It is highly scalable compared to other SQL counterparts.



Normalization:

This database schema appears to follow **Third Normal Form (3NF)**, which is a good standard for relational databases. Let me explain:

1. **First Normal Form (1NF)**:
    - **Atomic values**: Each field in a table contains atomic, indivisible values (e.g., `name`, `role`, `slot_time`, etc.), which means the schema adheres to 1NF.
2. **Second Normal Form (2NF)**:
    - **No partial dependencies**: In the `Bookings` table, non-primary key fields (like `centre`, `court`, `sport`) depend on the whole primary key (`id`). There is no partial dependency, ensuring compliance with 2NF.
3. **Third Normal Form (3NF)**:
    - **No transitive dependencies**: Every non-key attribute is fully dependent on the primary key and not on other non-key attributes. For example, in the

`User` table, `email` and `password` depend on the primary key (`id`), not on any other field, meeting the criteria for 3NF.

Advantages and Design Decisions:

1. **Role-Based Structure (`User` table)**:
   - The `User` table includes a **role** field that specifies whether the user is a **customer** or **manager**. This is a good design choice because it simplifies role-based access control.
2. **Booking System Flexibility (`Bookings` table)**:
   - The `Bookings` table references other entities like `centre`, `sport`, and `court`, allowing for flexible relationships between these tables. This makes it easy to scale the booking system across different centers and sports.
   - Storing `slot_time` and `slot_date` as separate fields allows easy querying and manipulation of booking schedules.
3. **Separation of Concerns (`Court`, `Centre`, `Sport` tables)**:
   - By separating `Court`, `Centre`, and `Sport` into distinct tables, you allow the system to efficiently manage relationships between sports facilities. This modular approach improves maintainability and scalability.
   - If a new center or sport is added, it can easily be integrated without changing the existing structure.
4. **Referential Integrity**:
   - Each table maintains proper foreign key relationships (e.g., `sport_id`, `centre_id`, `user_id`), which ensures referential integrity. This avoids issues like orphaned records and ensures that every booking is correctly linked to a valid user, court, and sport.
5. **Enum for Roles**:
   - Using an **enum** for the `role` field in the `User` table is a great decision. It restricts the possible roles a user can have (e.g., "manager" or "customer"), reducing errors and enforcing consistency.

6. **Scalability**:
    - The schema is scalable in that additional tables can easily be added for other features without disrupting the core functionality. For instance, if a review or feedback system were added, it could reference the `Bookings` or `User` tables without requiring significant changes.

3.Authentication:

JWT-based authentication is implemented to secure routes, ensuring only authenticated users can interact with the booking system. The middleware checks the validity of the token before granting access to the requested resources.

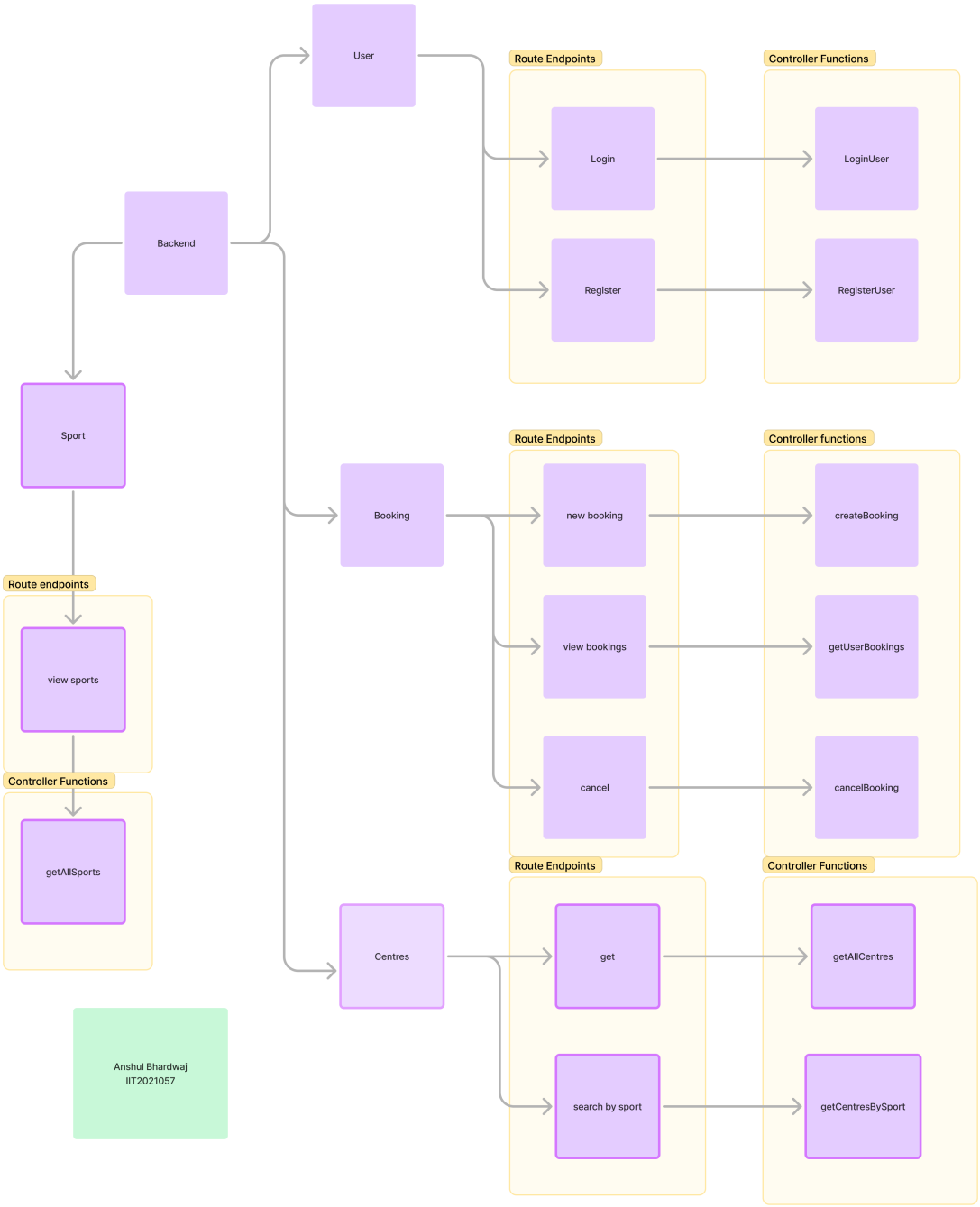
Implementation Details

Backend:

- The backend is built using **Node.js** and **Express.js**, which are commonly used for building fast and scalable web applications.
- **Express.js** handles routing and HTTP requests, allowing users to interact with different endpoints (such as booking, user management, etc.).
- **Mongoose**, an ODM (Object Data Modeling) library, is used to interact with the MongoDB database. It provides schema-based validation and simplifies queries, making it easier to work with MongoDB's flexible, non-relational structure.
- **Controllers**: Each module (user, booking, center, and sport) is managed by its own controller. This modular approach helps keep the code organized and maintainable. Each controller handles the logic for its specific part of the application, like creating, updating, and retrieving data from the database.
- Directory Tree:

```
.
└── backend_booking_app
├── controllers
```

```
│   ├── addCourtAndUpdateCentre.js
│   ├── addDummyCentre.js
│   ├── addDummyUsers.js
│   ├── bookingController.js
│   ├── centreController.js
│   ├── sportController.js
│   └── userController.js
├── db
│   └── connection.js
├── index.js
├── middlewares
│   └── authmiddleware.js
├── models
│   ├── Booking.js
│   ├── Centre.js
│   ├── Court.js
│   ├── Sport.js
│   └── User.js
├── package-lock.json
├── package.json
└── routes
├── bookingRoutes.js
├── centreRoutes.js
├── protectedRoutes.js
├── sportRoutes.js
└── userRoutes.js
```



JWT Authentication:

- **JWT (JSON Web Token)** is used for authenticating users. When a user logs in, they receive a JWT token, which is then used to access protected routes. This

ensures that only authenticated users can perform certain actions (e.g., making a booking or canceling one).

- The token contains encoded user information and is passed in the **Authorization** header of each request.
- In every request to a protected route, the token is verified using the server's secret key. If the token is valid, the user is allowed to proceed; otherwise, access is denied.

Password Security:

- **bcryptjs** is a hashing library used to securely store passwords. When a user registers, their password is hashed (converted into an unreadable string) using bcrypt before it is saved in the database.
- This hashing process involves generating a **salt**, which is a random value added to the password before hashing to make it harder to crack.
- Even if an attacker gains access to the database, the hashed passwords are extremely difficult to reverse-engineer due to the way bcrypt processes them.
- When a user logs in, the plaintext password they provide is compared with the hashed password in the database. **bcrypt** compares the hashed version of the entered password with the stored hash, ensuring secure authentication.

This setup ensures that sensitive data (like passwords) is protected and unauthorized access is prevented, enhancing overall security.

Booking Logic:

- When a user attempts to create a booking, the system first queries the database to check if the requested **court**, **date**, and **time slot** are already booked.
- If an existing booking is found for that specific court and time, the request is immediately denied, preventing double booking.
- This ensures that no two users can book the same court for the same slot, maintaining data integrity and avoiding scheduling conflicts.

Challenges and Solutions

- **Double Booking Prevention**:
  - **Challenge**: Preventing multiple users from booking the same court for the same date and time.
  - **Solution**: A database query in the `BookingController` checks for existing bookings at the court, date, and time. If a conflict is found, the system blocks the new booking request.
- **Role-Based Access Control**:
  - **Challenge**: Implementing different permissions for customers and managers.
  - **Solution**: JWT middleware is used to determine the user's role (customer or manager). Access restrictions are enforced at the route level, ensuring each user can only perform actions allowed by their role.
- **Database Relationships**:
  - **Challenge**: Efficiently managing the relationships between users, courts, centers, and sports.
  - **Solution**: Separate schemas were created for each entity (User, Centre, Sport, Court). Mongoose's `populate` function is used to retrieve related data across collections, simplifying queries involving multiple entities.

Future Improvements

1. **Industry-Level Validation**:
   - Implement robust validation mechanisms for bookings, ensuring inputs meet industry standards and integrating real-time system checks to enhance data integrity.
2. **Slot Suggestions**:

- Offer alternative time slots when the requested slot is unavailable, improving user experience and maximizing resource utilization.

3. **Admin Role**:
   - Add an **admin** role to manage centers, sports, and courts dynamically. This would replace the current approach of using dummy functions for sport, court, centre initialization, allowing admins to create, update, and manage resources through the system.
4. **Flexible Slot Durations**:
   - Introduce more flexible booking slots with customizable durations. Currently, only fixed 1-hour slots (e.g., 1:00-2:00, 2:00-3:00) are available with fixed start and end times. The system could be enhanced to allow users to book variable-length slots, accommodating different needs and improving court usage efficiency.
