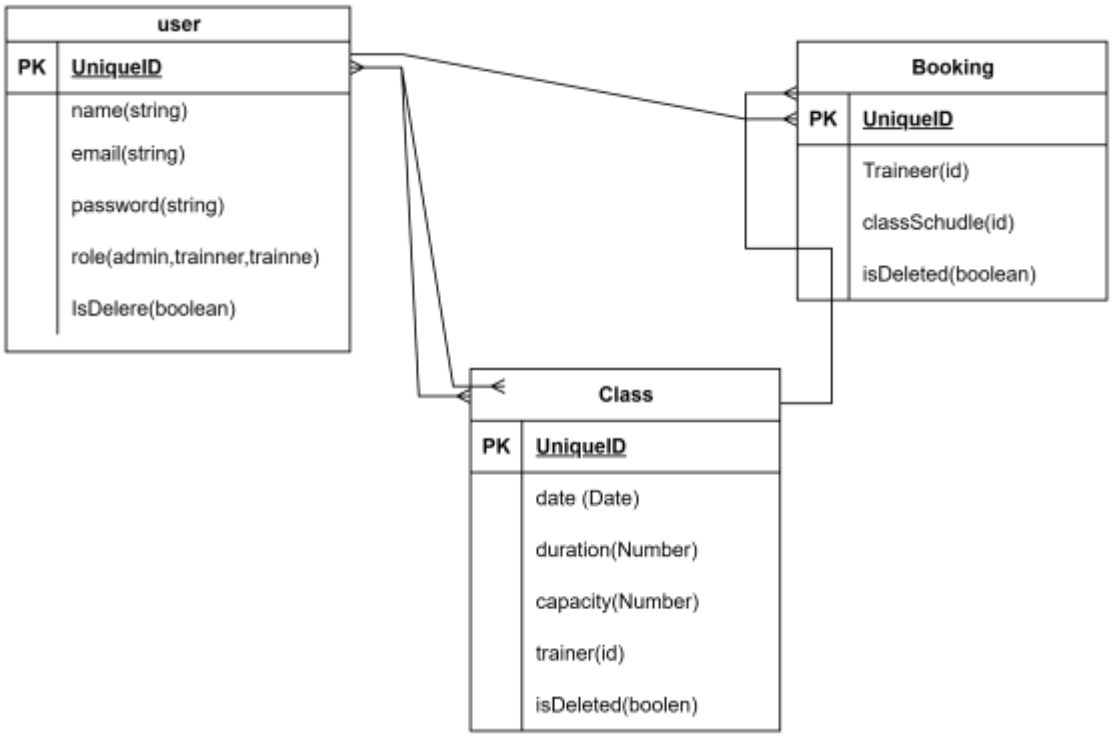
Gym Class Scheduling & Membership Management System

Project Overview

This Gym Class Scheduling and Membership Management System enables **admins** to manage classes, trainers, and trainees. **Trainees** can view and book classes, while **trainers** and **admins** can manage schedules and users. Key features include:

- JWT authentication
- Role-based access control (Admin, Trainer, Trainee)
- Soft delete for users, classes, and bookings
- Booking overlap prevention
- Capacity management for classes

Relational Diagram



Technology Stack

- **Language:** TypeScript

- **Framework:** Express.js
- **Database/ODM:** MongoDB with Mongoose
- **Authentication:** JWT (JSON Web Tokens)
- **Validation:** Zod + express-validator
- **Other Tools:** bcrypt, http-status, Vercel (deployment)

API Endpoints

Auth

1. Registration

- **Route:** POST /api/v1/auth/register
- **Body:**

json
CopyEdit
```
{
  "name": "John Doe",
  "email": "john@gmail.com",
  "password": "securePass123"
}
```

- **Response:**

```
{
  "statusCode": 201,
  "success": true,
  "message": "User registered successfully",
  "data": {
    "id": "689cdf241770f4b9ab7964e2",
    "name": "John Doe",
    "email": "john@gmail.com",
    "role": "TRAINEE"
  }
}
```

2. Login

- **Route:** POST /api/v1/auth/login
- **Body:**

```
{
  "email": "john@gmail.com",
  "password": "securePass123"
}
```

- **Response:**

```
{
```

```
  "statusCode": 200,
  "success": true,
  "message": "User logged in successfully",
  "data": {
    "accessToken": "<JWT_TOKEN>"
  }
}
```

User Management

1. Create User (Admin Only)

- **Route:** POST /api/v1/user/create
- **Body:**

```
json
CopyEdit
{
  "name": "trainer1",
  "email": "trainer1@gmail.com",
  "password": "securePass123",
  "role": "TRAINER"
}
```

2. Get Profile (All Roles)

- **Route:** GET /api/v1/user/profile
- **Header:** Authorization: Bearer <JWT_TOKEN>
- **Response:**

```
json
CopyEdit
{
  "statusCode": 200,
  "success": true,
  "message": "Profile fetched",
  "data": {
    "_id": "id",
    "name": "John Doe",
    "email": "john@gmail.com",
    "role": "TRAINEE"
  }
}
```

3. Get Single User (Admin Only)

- **Route:** GET /api/v1/user/single/:id

4. Get All Users (Admin Only)

- **Route:** GET /api/v1/user

5. Update User (Admin Only)

- **Route:** PUT /api/v1/user/update/:id
- **Body Example:**

json
CopyEdit
```
{
  "name": "perfecttrainer1223"
}
```

6. Delete User (Admin Only)

- **Route:** DELETE /api/v1/user/delete/:id

7. User Settings (Admin/Trainee)

- **Route:** GET /api/v1/user/setting/profile

Class Management

1. Create Class (Admin Only)

- **Route:** POST /api/v1/classes/create
- **Body:**

```
{
  "date": "2025-08-20T10:00:00.000Z",
  "duration": 120,
  "capacity": 10,
  "trainer": "trainer_id"
}
```

- **Response:**

```
{
  "statusCode": 200,
  "success": true,
  "message": "Classes created successfully",
  "data": {
    "date": "2025-08-20T10:00:00.000Z",
    "duration": 120,
    "trainer": "trainer_id"
  }
}
```

- **Error (5 classes/day exceeded):**

json
CopyEdit
```
{
```

```
  "success": false,
  "message": "Daily class schedule limit (5) reached."
}
```

2. Update Class (Admin Only)

- **Route:** PUT /api/v1/classes/update/:id

3. Delete Class (Admin Only)

- **Route:** DELETE /api/v1/classes/delete/:id

4. Get Single Class

- **Route:** GET /api/v1/classes/single/:id

Booking Management

1. Create Booking (Trainee Only)

- **Route:** POST /api/v1/bookings/create-bookings
- **Body:**

```
json
CopyEdit
{
  "classSchedule": "class_id"
}
```

- **Response:**

```
json
CopyEdit
{
  "statusCode": 201,
  "success": true,
  "message": "Booking created successfully",
  "data": {
    "trainee": "trainee_id",
    "classSchedule": "class_id"
  }
}
```

2. My Bookings (Trainee Only)

- **Route:** GET /api/v1/bookings/my-bookings

3. Cancel Booking (Trainee Only)

- **Route:** DELETE /api/v1/bookings/delete-bookings/:id

Database Schema

User

```
{
  name: string,
  email: string,
  password: string,
  role: "ADMIN" | "TRAINER" | "TRAINEE",
  isDeleted: boolean,
  timestamps: true
}
```

ClassSchedule

```
{
  date: Date,
  duration: number,
  capacity: number,
  trainer: ObjectId,
  isDeleted: boolean,
  timestamps: true
}
```

Booking

ts
CopyEdit

```
{
  trainee: ObjectId,
  classSchedule: ObjectId,
  isDeleted: boolean,
  timestamps: true
}
```

User Credentials

Trainee:

```
{
  "email": "john@gmail3.com",
  "password": "securePass123"
}
```

Admin:

```
{
  "email": "admin@example.com",
  "password": "Admin@123"
}
```

Trainer:

json

```
{
  "email": "trainer2@gmail.com",
  "password": "securePass123"
}
```

---

1. Create Trainers (Admin only)

- Login as Admin → create a trainer with name, email, password → should succeed.

---

2. Schedule Classes (Admin only)

- Login as Admin → create a class (date, time, trainer) → should succeed if under 5/day.
- Try to create more than 5 in one day → should fail.
- Try to give the same trainer 2 classes at the same time → should fail.

---

3. Book Classes (Trainee only)

- Login as Trainee → book a class with free seats → should succeed.
- Try to book when class is full (10 people) → should fail.
- Try to book the same class twice → should fail.

---

4. Role Permissions

- Admin can manage trainers and schedules.
- Trainer can only view own schedules.
- Trainee can only book and view classes.

---

5. Login & Security

- Correct login → can access features.
- Wrong/expired token → access denied.

Instructions to Run Locally

Gym Class Scheduling – Setup Guide

1. Clone the Repository

```
git clone https://github.com/abdulalimsujon/Gym-Class-Scheduling.git
cd Gym-Class-Scheduling
```

2. Install Dependencies

```
npm install
```

3. Configure Environment Variables

Create a .env file in the project root and add the following:

```
PORT=5000
NODE_ENV=production

# Database Configuration
DATABASE_URL=mongodb://127.0.0.1:27017/gym-management

# Security & Authentication
SALT_ROUND=10
JWT_SECRET=your_jwt_secret
JWT_EXPIRES_IN=1d                # Access token expiry
JWT_ACCESSTOKEN=gsdklgklsjdfl   # Optional: Additional secret for access token
JWT_REFRESH_TOKEN=fashfka        # Optional: Refresh token secret
```

4. Start the Development Server

```
npm run start:dev
```

5. Access the API

Once the server is running, you can access the API at:

```
http://localhost:5000/api/v1
```

Live Hosting Link

https://gym-scheduling-one.vercel.app/

Click

Postman Documentation Link

https://documenter.getpostman.com/view/42134880/2sB3BGJVqD
click here