Gym Class Scheduling and Membership Management System Task

Assignment Requirements:

Note: If you are frontend-focused, keep your focus on the frontend part. Show your creativity in design. Try to use animation to make the website visually appealing. And also try to make a reusable Component.

Technology Stack (frontend):

- **Programming Language**: JavaScript/TypeScript (recommended)
- **Web Framework**: Next.js
- **Styling Framework**: Tailwind CSS / CSS
- **State Management**: Context API / React hooks/Redux Toolkit for data fetching(recommended)
- **Authentication**: JWT (client-side validation)

Frontend Features (Simplified with Redux):

1. Create a Home Page.

2. Authentication Pages:

- **Login Page**: Email and password fields with JWT-based authentication. Use Redux for handling the authentication state.
- **Registration Page** (Trainees only): Fields for Email, Password, and Full Name. Dispatch actions to register new users.

3. Admin Dashboard:

- **Manage Trainers**:
  - Fetch, add, update, and delete trainers using Redux actions and API calls.
  - Display a list of trainers with edit/delete actions.
- **Class Scheduling**:
  - Fetch existing class schedules and use forms to create new ones with Redux actions.
  - Ensure validation (max 5 schedules per day, 2-hour class duration) using Redux state.

4. Trainer Dashboard:

- **View Classes**:
    - Use Redux to fetch and display the trainer's assigned classes from the API.
    - Display basic information about each class (date, time, trainees).

6. API Integration with Redux:

- Use Redux Toolkit to fetch and manage data such as trainer lists, class schedules, and bookings.
- Dispatch actions to make API requests (`fetch` or `Axios`) for login, registration, trainer management, and class bookings.
- Handle responses, loading states, and errors via Redux slices.

7. Mobile Responsiveness:

- Ensure all pages and components are mobile-friendly using Tailwind CSS or CSS media queries.

Technology Stack (Backend):

- **Programming Language:** Javascript/Typescript(recommended)
- **Web Framework (backend):** Express.js
- **ORM or ODM:** Prisma/Mongoose
- **Database:** MongoDB/PostgreSQL
- **Authentication:** JWT (JSON Web Tokens)

Note: If you are backend-focused, keep your focus on the backend. you must follow a pattern like Modular pattern or MVC. Modular patterns will get priority. And please draw a relational diagram.

Else If you are frontend-focused you can work with the normal pattern……

Project Discussion: (please read carefully)

The **Gym Class Scheduling and Membership Management System** is designed to manage gym operations efficiently. The system defines three roles: **Admin**, **Trainer**, and **Trainee**, each with specific permissions. Admins are responsible for creating and managing **trainers**, scheduling **classes**, and assigning **trainers** to these **schedules**. Each day can have a maximum of **five** class schedules, with each class lasting two hours. Trainers conduct the

classes and can view their assigned class schedules but cannot create new schedules or manage trainee profiles. Trainees can create and manage their own profiles and book class schedules if there is availability, with a maximum of ten trainees per schedule.

The system enforces several business rules to ensure smooth operations. Admins are limited to scheduling a maximum of 5 classes per day, and each schedule can accommodate no more than 10 trainees. If a class schedule reaches its capacity, trainees will be prevented from making further bookings, and admins cannot create additional schedules beyond the daily limit. JWT-based authentication is implemented to control access, ensuring that only authorized users can perform specific actions, such as booking classes or managing trainers. Robust error handling is integrated throughout the system, addressing issues such as unauthorized access, validation errors, and booking limit violations. This system provides an organized and flexible solution for managing gym class scheduling and membership, with well-defined roles and responsibilities.

Business Rules:

1. **Authentication:**
    - All users must authenticate using JWT tokens to perform actions within the system.
    - Trainees can create and manage their own profiles.
    - Only admins can create and manage trainers.
2. **Roles and Permissions:**
    - **Admin**: Responsible for creating trainers, managing class schedules, and assigning trainers to classes. Admins cannot create or manage trainee profiles.
    - **Trainer**: Trainers can view their assigned class schedules but cannot create new schedules or manage trainee profiles.
3. **Class Scheduling:**
    - Each day is limited to a **maximum of 5 class schedules**.
    - Each class schedule lasts for **2 hours**.
    - The system enforces a **maximum of 10 trainees** per class schedule. Once the limit is reached, no additional bookings can be made for that schedule.
    - Admins are responsible for scheduling classes and assigning trainers.

Error Handling:

Implement proper error handling throughout the application. Use global error handling middleware to catch and handle errors, providing appropriate error responses with status codes and error messages.

- **Unauthorized Access:** Users who attempt actions without proper authentication or authorization will receive an "Unauthorized access" error.
- **Validation Errors:** Input validation will ensure that all required fields are provided, and if any data is invalid (e.g., incorrect email format), appropriate validation errors will be returned.
- **Class Booking Limit Exceeded:** If a trainee attempts to book a class that has already reached the maximum of 10 attendees, the system will return an error message indicating that the schedule is full.
- **Schedule Limit:** Admins attempting to create more than 5 schedules per day will receive an error.

Sample Error Response:

- **For Validation Errors:**

```
{
   "success": false,
   "message": "Validation error occurred.",
   "errorDetails": {
      "field": "email",
      "message": "Invalid email format."
   }
}
```

Unauthorized Access:

```
{
   "success": false,
   "message": "Unauthorized access.",
   "errorDetails": "You must be an admin to perform this action."
}
```

Booking Limit Exceeded:

```
{
   "success": false,
   "message": "Class schedule is full. Maximum 10 trainees allowed per schedule."
}
```

Success Response:

```
{
   "success": true,
   "statusCode": 201,
   "message": "Class booked successfully"
     "Data":[display the response data]
}
```

Submission Process:

DEADLINE: 01 December 2024, 11:59 pm

To ensure your submission meets the requirements, please carefully follow the steps outlined below:

1. Host the Server:

- Deploy your server on a live platform like **Vercel**, **Heroku**, etc.
- Ensure the server and all API endpoints are accessible via a live link.

2. Create a Git Repository:

- Push all your code to a **public GitHub** (or GitLab/Bitbucket) repository.
- Include a **detailed README.md** file that explains the project.

3. README.md File Must Include:

- **Project Overview:** Brief description of the system.
- **Relation Diagram:** Draw a Relational Diagram for the backend and add the link or image.
- **Technology Stack:** List of technologies used (e.g., TypeScript, Express.js, Prisma/Mongoose, PostgreSQL/MongoDB, JWT).
- **API Endpoints:** Document API methods, parameters, and responses.
- **Database Schema:** Include model definitions.
- **Admin Credentials:** Provide login details for admin access.
- **Instructions to Run Locally:** Step-by-step guide with commands for setup, installation, and starting the server.
- **Live Hosting Link:** Provide the live link to your deployed project.

---

4. Testing Instructions:

- Provide **admin login credentials** for testing.
- Explain how to test key features (e.g., creating trainers, scheduling classes, booking).

---

5. Submit the Project:

- Submit the following to the Google Form: https://forms.gle/vaSW89kA6XqqJzJZA

Important:

- **Ensure all steps are completed**—missing any details (live link, credentials, API documentation) may result in **disqualification**.
- Test all functionality thoroughly before submitting.

By following these steps, you'll ensure a complete and successful project submission.

1. Host the Server:

- **Platform Options: Deploy your server on a live platform such as Vercel, Heroku, or Laravel Forge.**
- **Ensure Accessibility: All API endpoints and the server should be fully accessible via a live link for testing purposes.**