FullStack Engineer (Node.js + React)

Objective

The aim of this technical test is to assess the candidate's proficiency in building full-stack web applications using **Node.js** (backend) and **React** (frontend) with **TypeScript**. This test will focus on your ability to design and develop scalable, maintainable, and well-structured web applications, covering both backend and frontend challenges. You are expected to demonstrate a solid understanding of best practices, performance optimization techniques, and clean architecture principles.

Task Overview

You will be tasked with creating a **Performance Tracking Dashboard** for athletes, which will allow users (athletes or coaches) to view and manage athlete profiles and their performance metrics.

Core Requirements

Entity Definitions and Structure

- **Athletes:** Each athlete should have the following fields:
  - id (UUID)
  - name
  - age
  - team
- **Performance Metrics:** Each performance metric should have the following fields:
  - id (UUID)

- athleteId (UUID, foreign key to Athletes)
- metricType (e.g., speed, strength, stamina)
- value (float)
- unit (e.g., kg, meters/second)
- timestamp (datetime)

Backend

Build a REST API with the following endpoints:

- **POST** `/athletes`**:** Create a new athlete profile.
- **GET** `/athletes`**:** Retrieve a list of all athletes.
- **GET** `/athletes/{id}`**:** Get details and performance metrics for a specific athlete.
- **PUT** `/athletes/{id}`**:** Update athlete information.
- **POST** `/athletes/{id}/metrics`**:** Add a new performance metric (e.g., speed, strength) to an athlete.
- **GET** `/athletes/{id}/metrics`**:** Get all metrics for an athlete, with the option to filter by metricType (e.g., speed, endurance).
- **DELETE** `/athletes/{id}`**:** Delete an athlete and their performance metrics.

Frontend

Build a **React** web application where users can:

- **View all athletes** in a table with basic information (e.g., name, age, team).
- **View detailed performance metrics** for a specific athlete.
- **Create or edit athlete profiles** using forms.
- **Add new performance metrics** for an athlete.
- **Delete athlete profiles**.
- Add basic **form validation** for the creation/editing of athletes and metrics.

Stack

- **TypeScript:** The project must be developed using **Node.js with TypeScript**.
- **Docker:** The application should be containerized using **Docker**. Provide a Dockerfile and docker-compose configuration that sets up the app and a PostgreSQL instance.
- **PostgreSQL**: Use **PostgreSQL** as the database.
- **Prisma**: Use **Prisma** as the ORM for database interaction.
- **Hono**: Use Hono as http framework for building the API.
- Ionic: use Ionic for structuring the frontend app and routing.
- **PandaCSS**: Use **PandaCSS** (or an alternative CSS framework) for styling.
- **React Query**: Use **React Query** for data fetching and state management.

Additional Requirements

- **Monorepo Setup:** Organize the backend and frontend projects in a **monorepo** structure using a tool like **Nx** or **Turborepo**.
- Provide **error handling** for backend and frontend interactions (e.g., invalid form submissions, API errors).

Bonus

- Implement performance optimizations (e.g., **code splitting**, **lazy loading** components).
- **Domain-Driven Design:** Apply **DDD** concepts where necessary, particularly in structuring services and entities.
- **Caching:** Use an in-memory cache like **Redis** to optimize repetitive requests on the backend.
- **CI/CD Integration:** Set up a basic CI/CD pipeline (e.g., using GitHub Actions) to automatically test and deploy the application.
- **Authentication:** Implement a simple **JWT-based authentication** system to restrict certain actions (e.g., creating, updating, deleting athletes).
- **Testing:** Add integration tests (e.g., using **Jest** or **Cypress**) for key parts of the frontend and backend.

What We Expect

- **Functionality:** Ensure the core features work as expected.
- **Code quality:** Your code should be well-structured, modular, and follow best practices for both React and Node.js applications.
- **Documentation:** Provide a README.md file with clear instructions on how to set up and run the project, as well as explanations for your architectural decisions.
- **Timeframe:** The test should be completed before 2 weeks. Focus on implementing a fully functional application that demonstrates your strengths while covering all core requirements.

Submission

- Please submit your project as a GitHub repository (or similar platform) with detailed instructions on how to set it up and run.
- We will review the following aspects:
  - **Functionality**: Does it meet the requirements?
  - **Code quality**: Is the code clean, modular, and maintainable?
  - **Adherence to best practices**: Use of SOLID, clean architecture, and testing practices.
  - **Performance optimizations**: Have you considered performance in both frontend and backend design?
  - **Completion time and documentation**

We look forward to seeing your work!