Backend Engineer (Node.js)

Objective

The aim of this technical test is to evaluate the candidate’s ability to develop and implement backend solutions in a **Node.js** ecosystem with a focus on scalability, clean architecture, and efficient design. The test will allow us to assess the candidate’s proficiency in TypeScript, their understanding of key technologies and principles (Prisma, PostgreSQL, Docker, etc.), and their ability to design solutions that follow good software engineering practices such as SOLID principles and Domain-Driven Design (DDD).

Task Overview

You will be tasked with building a simplified REST API for managing athlete data and their performance metrics. This service will allow athletes to be registered, their performance data to be updated, and basic queries on their data to be executed.

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)

API Endpoints

- **POST** `/athletes`**:** Create a new athlete in the system.
- **POST** `/athletes/{id}/metrics`**:** Add a new performance metric for a specific athlete.
- **GET** `/athletes`**:** Retrieve a list of all athletes.
- **GET** `/athletes/{id}`**:** Get details and performance metrics for a specific athlete.
- **GET** `/athletes/{id}/metrics`**:** Retrieve the performance metrics for a specific athlete, with the option to filter by metricType and a date range.
- **PUT** `/athletes/{id}`**:** Update an athlete's information.
- **DELETE** `/athletes/{id}`**:** Delete an athlete and all their related performance metrics.
- **GET** `/athletes/{id}/metrics/aggregate`**:** Retrieve aggregate statistics for an athlete's performance metrics.
  - **Options:**
    - **metricType:** Filter aggregate calculations by a specific metric type (e.g., speed, strength).
    - **Operations to support:**
      - **Average** value for a given metric type
      - **Max** and **Min** values for a given metric type
      - **Total count** of recorded metrics for a given metric type
      - **Standard deviation** (as a bonus, if possible) for a metric type over time.

- **GET** `/metrics/leaderboard`**:** Retrieve a leaderboard of athletes ranked by the highest average value for a specified metric type.
  - **Options:**
    - **metricType:** (e.g., speed, strength)
    - **limit:** Restrict the number of athletes returned in the leaderboard (default: 10).

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.

Additional Requirements

- **Error Handling:** Ensure proper error handling for invalid requests (e.g., non-existent athlete IDs).
- **Testing:** Implement unit tests for critical parts of the functionality (e.g., creating athletes, retrieving metrics).
- **Testing:** Add integration / unitary tests (e.g., using **Jest** or **Vitest**) for critical parts of the functionality.
- **SOLID Principles:** Demonstrate adherence to SOLID principles in your implementation.
- **OOD Principles:** Demonstrate adherence to OOD principles in your implementation.

Bonus

- **Authentication:** Implement a simple **JWT-based authentication** system to restrict certain actions (e.g., creating, updating, deleting athletes).
- **GET** `/athletes/{id}/metrics/trends`**:** Retrieve the trend of an athlete's performance metrics over time. The response should provide a series of timestamped data points showing how the selected metric type changed over time. Include a simple linear regression model to predict future performance based on past data.
- **Options:**
  - **metricType:** Filter trend data for a specific metric type (e.g., speed, strength).
  - **dateRange:** Filter data by a specific time period (e.g., last 30 days, last 6 months).
- **Domain-Driven Design:** Apply **DDD** concepts where necessary, particularly in structuring services and entities.
- **Caching:** Add basic caching (using an in-memory cache like Redis) for repeated queries of athlete performance metrics.

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 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.
  - **Completion time and documentation**

We look forward to seeing your work!