
DaCodes.

Node.js Developer Challenge: "Time It Right" Game

⚙️ Technical Requirements

- **Tech Stack:** Node.js, TypeScript, Express.
- **Persistence Layer:** Use an in-memory database (Redis, or a simple object to track users and scores). If you're comfortable, you can use a lightweight DB like SQLite.
- **JWT Authentication:** Use JWT tokens for authenticating users.
- **API Documentation:** Use tools like Postman or Insomnia to create a collection with test cases that cover the happy path and edge cases. Share this as part of your deliverable.

🏁 Objective

Build a **game timer system** with the following mechanics:

- Users authenticate to start a game session.
- After authentication, they can **start a timer** (sending a request).
- The user will then **stop the timer** by sending another request. The goal is to time the stop request **exactly 10 seconds after the start request**.
- The game should track how close the user's stop request was to the target (10 seconds).

- Users with the most accurate timing should be ranked on a **leaderboard** that displays the top 10 users based on their average deviation from 10 seconds.

🚀 Challenge Requirements

1. **Authentication:**
    - Create an authentication endpoint for users to log in (e.g., username/password, or any simple form of authentication).
    - Upon successful authentication, return a **JWT token** for the user.
2. **Game Session:**
    - **Start Timer Endpoint:**
      `POST /games/:userId/start`
        - This starts a game session and stores the time the request was received.
        - Return a response containing a **session token** that expires in 30 minutes.
    - **Stop Timer Endpoint:**
      `POST /games/:userId/stop`
        - This ends the game session. It should calculate the **difference between the stop request and the start request** in **milliseconds**.
        - If the difference is **within 10 seconds (±500 ms)**, the player gets a point (or another score metric).
3. **Leaderboard:**
    - Endpoint to retrieve the **top 10 users** with the **smallest average time difference** across multiple rounds:
      `GET /leaderboard`
        - Rank users by the **average time deviation** from the target (10 seconds), from lowest to highest.
        - **Leaderboard Time Format:** For both the leaderboard display and the game session results, display the time difference in **milliseconds** format. For example:
            - Target time: 10,000 ms (10 seconds).
            - Player time: 9,950 ms (50 ms deviation).

Display the results with the following structure:
`UserID | Total Games | Average Deviation (ms) | Best Deviation (ms)`

4. **Analytics:**
    - Each successful game session (start/stop) should be logged and used for calculating **score deviation** for each user.

🚀 **Architecture Plan:**

System Architecture Overview:

- Before coding, provide a high-level architecture for the backend system.

Deliverable:

- A simple diagram (draw.io, Lucidchart, or even Markdown ASCII) that includes:
    - Auth flow
    - Game timer logic
    - Leaderboard
    - Storage (in-memory/Redis/SQLite)

Explain how components interact (brief description in the README)

Scalability & Improvement Plan

Describe, in a few paragraphs or bullets:

- What would need to change to scale the system to 10,000+ users
- One improvement you would prioritize for production-readiness (e.g., add metrics, swap DB, refactor for modularity)

💻 Implementation Details

1. **Authentication:**
    - Implement simple authentication (JWT tokens) for each user.
    - Assume the game will have one endpoint for user login (`POST /auth/login`) to generate JWT tokens.
2. **Game Mechanics:**
    - For simplicity, store session data in-memory (in a dictionary or database, e.g., Redis, SQLite).
    - For each game session, store the **start time** and **stop time** in **milliseconds**.
    - Calculate the **deviation** from the target time (10 seconds or 10,000 milliseconds) on each stop.
    - Ensure that the **stop time** is calculated in **milliseconds** and that the deviation is displayed as the difference from 10 seconds.
3. **Leaderboard:**
    - Maintain a leaderboard ranking based on the **average deviation** (in milliseconds) for each player.
    - Store the **user scores** in a persistent layer (e.g., in-memory database, or local file) for simplicity.
    - Display the leaderboard in the format described above, showing both **average deviation** and **best deviation**in **milliseconds**.

📦 **Deliverables**

1. **System Architecture Diagram:** Include a high-level diagram showing how the game backend is structured.
2. **In your README, include:** A short explanation of your architecture and tech choices
3. A brief plan for how you would scale the system for higher load or prepare it for production
4. **Code:**
    - Push your solution to a Git repository.
    - Structure your application with clear separation of concerns (authentication, game session logic, leaderboard).
    - Comment on your code and explain your thought process when necessary.
5. **Postman Collection or Swagger Documentation:**
    - Include a Postman for testing the authentication, start/stop game timers, and fetching the leaderboard.
    - Include **edge cases** in your collection (e.g., invalid sessions, multiple requests).
6. **README:**
    - Provide a simple README with instructions on how to:
        - Set up and run the application.
        - Test the APIs using Postman/Insomnia.
        - Describe the logic behind your leaderboard calculation.
7. **Deployment:** *(Bonus)*
    - It is **preferred** that your solution is already **deployed and ready to consume** (e.g., on Heroku, DigitalOcean, AWS, etc.).
    - If possible, **deploy** the solution to a public URL and share the link in your submission.
8. **Frontend:** *(Bonus)*
    - **Even better** would be a **basic frontend** showcasing the game mechanics and leaderboard. You can use any frontend framework or just HTML/JS. The frontend should demonstrate:
        - Login and session management.
        - The start/stop game functionality.
        - Displaying the leaderboard.

    - If you provide a frontend, make sure it is also deployed and accessible.
9. **Real-Time Leaderboard:** *(Bonus)*
    - Implement a **real-time leaderboard connection** using **WebSockets** or **Server-Sent Events (SSE)**.
    - As users complete more sessions, the leaderboard should update dynamically in real-time.
    - Provide the connection to the frontend, so users can see the leaderboard refresh without needing to reload the page.

⏰ Evaluation Criteria

- **Code quality:** Clean, modular code with appropriate use of TypeScript and async/await.
- **JWT Authentication:** Correct use of JWTs for session management.
- **Leaderboard Logic:** Accurate ranking of users based on average deviation in **milliseconds**.
- **Performance Considerations:** Ensure the leaderboard calculation is efficient.
- **Testing:** Comprehensive testing (unit tests for the logic and integration tests for APIs).
- **Documentation:** Clear documentation of your APIs and architecture.
- **Edge cases:** Handling errors and edge cases (e.g., invalid requests, multiple starts/stops, session timeouts).
- **Deployment:** Is the solution deployed and accessible? If a frontend is provided, is it functional and integrated?

💡 Using ChatGPT for Assistance

- **Feel free to use ChatGPT or other online resources** for any assistance you may need. For example, using ChatGPT to generate boilerplate code, search for best practices, or get guidance on complex concepts is encouraged. However:
    - **Any code generated by ChatGPT (or similar tools)** should be **documented in your code comments**, clearly indicating the portion of the code that was AI-generated (e.g., `// generated-with-GPT`).

    - **Ensure the solution meets senior-level standards.** While using AI can help, the final solution should reflect your understanding and ability to integrate AI suggestions with your expertise.
- **We expect you to adapt any AI-generated solutions** and refine them to fit the requirements and context of the challenge. Simply relying on AI for the entire challenge might not showcase your full capabilities.