Coding Challenge

The challenge completion time is one week.

Objective:

You've landed a backend development gig with a twist! Your task is to build the server-side logic for a slot machine game that ensures the house always has an edge. This is a backend-focused assignment, so there's no need to worry about frontend implementation.

Brief:

When a player starts a game session, they are allocated 10 credits. Pulling the machine lever (rolling the slots) costs 1 credit. The game has 1 row with 3 blocks and 4 possible symbols: cherry, lemon, orange, and watermelon, each offering different credit rewards. For players to win the roll, they have to get the same symbol in each block. However, the house always ensures it maintains an advantage.

Server-Side Requirements:

1. **Session Management:**
- Implement user session management where each new session starts with 10 credits.
- Each game session state should be stored and managed on the server.
- Implement a cash-out mechanism that transfers session credits to the user's account and closes the session.

1. **Game Logic:**
- Define a random roll logic with the following symbols and rewards:
- Cherry: 10 credits
- Lemon: 20 credits
- Orange: 30 credits
- Watermelon: 40 credits
- Implement a rolling mechanism that:
- Costs 1 credit per roll.
- Rewards the player if all three symbols in the roll match.

1. **House Advantage Logic:**
- If the player has fewer than 40 credits, rolls are truly random.
- If the player has between 40 and 60 credits:
- Implement a 30% chance that the server will re-roll a winning round before sending the result back to the client, thereby potentially avoiding a win.
- If the player has more than 60 credits:
- Increase the chance of re-rolling a winning round to 60%.
- Ensure that the re-rolling logic is integrated seamlessly and is non-deterministic.

1. **Cash-Out Logic:**
- Create an endpoint for cashing out that:

- Moves credits from the game session to the user's account.
- Ends the current game session.
- Include a mechanism to simulate potential failure or obstacles in the cash-out process to make it challenging:
- Introduce a 50% chance that a cash-out request might face a server-side delay or error (e.g., a timeout or intentional server error to handle on retry).

1. **Testing:**

- Write comprehensive unit tests for the game logic, session management, and cash-out mechanism.
- Implement integration tests to ensure the entire flow works as expected.
- Include edge cases in your tests, such as handling a user trying to play with insufficient credits or attempting multiple cash-outs in quick succession.

1. **Additional Complexity:**

- Add a logging mechanism to track each player's session, including all rolls, wins, losses, and cash-out attempts.
- Implement rate limiting on the roll and cash-out endpoints to simulate server load handling.
- Create a leaderboard mechanism to rank players based on their total accumulated credits, excluding any in-session credits.

Technical Requirements:

- Use TypeScript with the Express framework for the backend.
- Ensure RESTful API design.
- Use a database to store session data and user accounts (choose any SQL or NoSQL database).
- Implement error handling and data validation throughout the API.

Evaluation Criteria:

- **Completeness:** Did you complete all the backend features as briefed?
- **Correctness:** Does the solution perform logically and correctly?
- **Maintainability:** Is the code written cleanly and structured for easy maintenance?
- **Scalability:** Is the system designed to handle multiple concurrent sessions effectively?
- **Testing:** Is the system adequately tested, including edge cases and error conditions?

Submission:

Please organize, design, test, and document your code as if it were going into production. Work with Git and provide the solution as a link to the repository on GitHub, ideally on the main branch.