Game backend with NodeJS

As a NodeJS developer you want to get rich by building a game **backend** for gambling.

Returning assignment

Once you are satisfied with the implementation, make an archive and upload it using the link provided in the email. Please do **not** upload task instructions or your solution publicly, for example to GitHub. You may return your submission any time in the given time limit, which is seven working days.

General instructions

- **Follow the specifications below carefully**
- Use TypeScript
- Use express or similar level tools, avoid “scaffolding” (code generation etc).
- Feel free to use any libraries (it is recommended to use Socket.IO for streaming the messages).

Thing that are relevant for making a good grade

- Provide working code
    - It's better to have less features implemented but what is implemented should be working as per specifications. It is possible to implement the full game in a few hours, but if you cannot, it's ok to respond with less features. Just make what you implement work exactly as specified.
- This is a POC
    - Don't focus on production grade concerns, such as logging, performance, or security. Be prepared to talk about those in the interview, however.
- Add a README
    - **Please write clearly what is missing from implementation wrt. spec**
- Style
    - Less code is better, aim for less code but with clear professional patterns
    - Avoid too many dependencies and bloat
- Submission
    - Avoid leaving unrelated/dead code
    - Don't include node_modules in submission

Games system requirements

The game system has rooms for users to play games in. Game rooms allow users to place bets and play a game of dice. The game is the same in all rooms: all users who bet play, and get to throw a single die. Highest die roll wins, and receives all the bets. Die roll happens in the backend to prevent cheating. All traffic between users using the game board and the backend is done with a streaming API.

To save time when implementing this

- Set of game rooms in the game can be fixed
- Set of users can be fixed. Create them up-front, and just identify with message property.
- Users keep their balance forever, and no real authentication or security is needed. Users have a fixed balance up-front. There is no way to add credits (reboot the app)

If you are short in time, suggested implementation tiers and their messages:

- Tier 1: Basic game
    - Connection management: JOIN and ROOM_STATUS
    - Game management: START, GAME_STARTED, ROLL, DIE_ROLLED
      (all users in room get to roll)
    - Winning games: GAME_COMPLETE
    - (allow free re-rolls, game will not finish before there is a winner)
- Tier 2: Playing with balance
    - Balance management: BET, BET_PLACED, BALANCE, INSUFFICIENT_FUNDS
      (limit games to betting users)
    - Require unique winner result: REROLL
      (full implementation, bonus: let users who tied bet more)

Messages between users and the backend:

- Client to server
    - JOIN
        - Each room can be joined by any user
        - If playing user leaves, he loses credits in any on going game (Tier 2)
        - Sending JOIN again on same room has no effect
        - Respond with ROOM_STATUS
    - BET (Tier 2)
        - Bet specified amount of credits
        - This can be done multiple times before game starts
        - If valid
            - Respond with BALANCE message and BET_PLACED broadcast
        - If funds not enough, respond with INSUFFICIENT_FUNDS
        - Otherwise can be silent or respond with an ERROR of some kind
    - START

        - A game can be started by users who are in a game room by any user
        - A game can be only be started
            - If there are minimum two users who have bets (Tier 2)
            - And when one is not in progress
        - User who bet participates, others spectate (receiving broadcasts)
        - Respond with GAME_STARTED broadcast if valid
        - Respond with some error otherwise
    - ROLL
        - Messages to roll die when no game is on-going are simply ignored
        - Otherwise roll is recorded for the user and used for choosing between win or tie after all users have rolled
        - Respond with DIE_ROLLED broadcast
        - If game completed, respond with GAME_COMPLETE broadcast
        - If die roll is tied, respond with REROLL to all tied users (Tier 2)
- Server to client broadcasts
    - ROOM_STATUS
        - Users in the room
    - GAME_STARTED
        - List of users in the game (Tier 1) and their bets (Tier 2).
        - Total amount that can be won (Tier 2).
    - BET_PLACED (Tier 2)
        - User and amount
    - GAME_COMPLETE
        - Total cash won (Tier 2) and the winner (Tier 1)
    - DIE_ROLLED
        - All die rolls, user and result
- Server to client messages (unicast) (Tier 2)
    - BALANCE
        - Total amount of credits remaining
    - REROLL
        - Request to roll again for user who tied
    - INSUFFICIENT_FUNDS
        - Tell user that bet cannot be placed

Other specifications:

- Die is six sided (has 6 faces, with values 1,2,3,4,5,6)
- All users in the specific game room who bet must participate in the game. No one else can participate.
- There is no way to stop a game
- There is no way to join an on-going game
- Disconnecting or joining a new room during a game or after betting makes you lose the game, even if you rolled already.

- Game finishes when
    - All users in room have rolled a die and there is a winning die roll
    - When there is only one connected user left in the game
- Messages are only broadcast to users who are in the same game room