GreenRun - Backend Developer Test

Using Node.js, create a backend - API REST named GreenRun - Sports, which will be used for a sportsbook application. Any backend framework can be used but **the use of Hapi will give extra points.**

Mandatory requirements:

- Use of **TypeScript**
- Use of relational **MySQL** database
- The backend should have support for different user roles and its corresponding access permissions depending on the endpoint.
- All of the authentication process and sessions should be handled with Json Web Tokens (**JWT**)
- Deploy the API and db in any cloud service (**AWS**, **GCP**, etc.)
- Provide swagger or postman collection with corresponding descriptions and documentation for its use.
- Use any repository service and provide corresponding access

General description:

This API and its corresponding database for storing the data consist of admins/users of a sportsbook application that offers bets on different sports and allows the users to create an account, deposit money, place bets, withdraw money. The admins can perform some internal monitoring and administrative operations for the plattform

1. The API should have two defined roles and corresponding endpoints to register users of any role and login. The roles are the following:

    a. Admin
    b. User

    - The login auth process can be handled directly in the backend or by an external service as **Firebase auth.**

2. User endpoints / permissions:

    a. Place a bet on a specific event
    b. Deposit money in its account (create corresponding transaction)
    c. Withdraw money (create corresponding transaction)
    d. Update user data
    e. Request its balance (calculate balance based on transactions)

f. Get its transactions (can be filtered by type deposit, withdraw, bet, winning)

3. Admin endpoints / permissions:

    a. List all of the bets (can be filtered by specific event or sport) (bets table)

    b. List all of the user transactions (can be also filtered by specific user and/or category)
    c. Request user balance
    d. Change a bet status (active / cancelled)
    e. Block a specific user (user state => active / blocked) (can’t block other admins)
    f. Settled bets results (won / lost)
    * This settlement should trigger payments for users that have a placed bet for the winning option in case of a won
    g. Update user data

Database structure suggestion: This is just a suggestion based on how we think that the database should be structured but any structure is accepted as long as it fits the requirements of the API requested. Also the proposed structure can contain any number of tables.

- **Users table:** This table should store users information

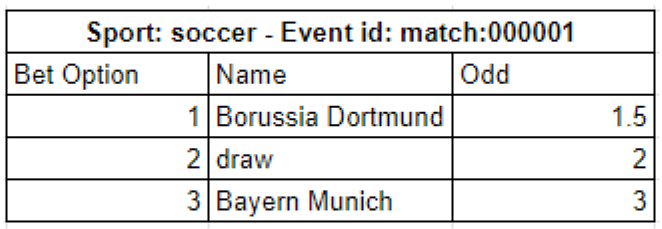
  id, role, first_name, last_name, phone, email, username, address, gender, birth_date, country_id, city, category, document_id, user_state, created_at, updated_at, deleted, deleted_at

- **Transactions table:** This table should store all of the transactions made by the users i.e. deposits, bets, withdraws

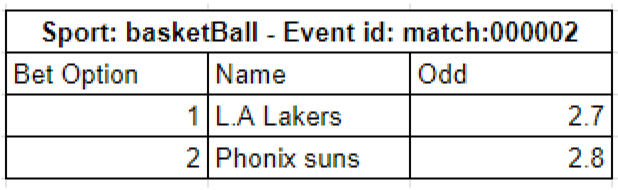
  id, user_id, amount, category (deposit, withdraw, bet, winning), status, created_at, updated_at, deleted, deleted_at, user_bet_id (if bet or winning)

  * for winning transactions the amount is calculated based on bet odd * user_bet amount

- **Bets table:** This table should store all of the available bets on the plattform. To be specific, you could use the money-line concept where each of the teams involved in a match have a bet option and also the draw if it's common in the sport. Please add at least bets for two sports. For example:



Sport: soccer - Event id: match:000001		
Bet Option	Name	Odd
1	Borussia Dortmund	1.5
2	draw	2
3	Bayern Munich	3



Sport: basketBall - Event id: match:000002		
Bet Option	Name	Odd
1	L.A Lakers	2.7
2	Phonix suns	2.8

id, bet_option, sport, status (active / cancelled / settled), name, event_id, odd, result (won/lost) created_at, updated_at, deleted, deleted_at

- **User bets:** This table should store all placed user bets

  id, user_id, bet_id, odd, amount, state (open/won/lost), created_at, updated_at, deleted, deleted_at

Expected Behavior:

- All the placed bets or withdraws should check if the user has enough amount to perform that action
- Place more than one bet at the same time should be allowed
- Admins can not block other admins
- Admins can update user data (not from another admins)
- Bets should have different status (active/cancelled/settled) taking into account the following requirements:
  - Users can not place bets on cancelled or settled bets
  - Already settled bets can not be cancelled
  - Cancelled bets can be activated again
- User balance should be calculated based on the user transactions.
- Admins can settle bet results for a specific event_id and each one of its bet options (won/lost) and then the endpoint should trigger payments for all of the bet transactions on the winning option and also update the state of the user_bets table. Again, for winning transactions the amount is calculated based on bet odd * user_bet amount

Extra points:

- Use **Hapi** framework
- Deploy in **AWS**

- Provide **swagger**
- Provide documentation
- Use of **Knex** for query building
- Clean code
- Good repository and branching management
- Use of **GITLab**
- Good definition of db interfaces and models
- Create CI/CD pipelines for automatic deployment to cloud server