Back End Documentation

High-Level Structure

Express.js

Express.js is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. It facilitates the rapid development of Node-based web applications, offering features like routing, middleware support, and template engines.

Mongoose

Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js. It manages relationships between data, provides schema validation, and is used to translate between objects in code and the representation of those objects in MongoDB.

MongoDB Atlas

MongoDB Atlas is a fully-managed cloud database developed by the same people that build MongoDB. It automates the time-consuming database administration work, freeing you to focus on building and optimizing your applications.

MVC Pattern

MVC stands for Model-View-Controller. It is a design pattern that separates an application into three main logical components: Model, View, and Controller. Each of

these components is built to handle specific development aspects of an application. In the context of your Express.js API:

Model (Schemas): The Model represents the structure of your data. In your case, you're using Mongoose schemas to define the structure of the documents within your MongoDB database. This includes the types of data fields, default values, validators, etc. It defines and manipulates the data in your database.

View (Routes): In the context of an API, a "View" can be thought of as the end-user interface with the application, even though it isn't a visual interface. Here, it refers to the routes in your API, which determine how the API responds to client requests. Each route is associated with a specific URL pattern, HTTP method (like GET, POST, etc.), and function (controller) that handles the request.

Controller (Middleware Functions): The Controller acts as an intermediary between Model and View, processing all the business logic and incoming requests. In your Express application, the controller is where you define your middleware functions. Each function corresponds to a route and uses the models to interact with your database, and then sends the data back to the client.

Routes Implemented

Transaction Route: This route is designed to handle financial transactions related to user winnings and cash-outs. Players who win tournaments or bets can use this route to process their winnings and transfer them out of the platform. Additionally, the cashOut feature that was mentioned could be an endpoint in this route, allowing users to directly convert their winnings to cash. Not implemented fully

Auth Route: The Auth (authentication) route is a critical part of any application, and even more so for a gaming gambling platform. It manages all things related to user authentication, including account creation, log in, log out, password reset, and possibly two-factor authentication. Ensuring secure user authentication is crucial to maintain the integrity of user accounts and their associated funds.

Tournaments Route: The Tournaments route handles all aspects related to the gaming tournaments on the platform. This can include viewing and registering for upcoming tournaments, tracking ongoing tournaments, viewing past tournaments, and displaying the results and payouts of completed tournaments. Not implemented fully

Wagers Route: On a gaming gambling platform, the Wagers route is responsible for managing user bets on tournaments or matches. It handles the placement of wagers, calculating and updating odds, tracking the results, and managing the payout of winnings.

Dispute Route: The Dispute route is an important part of maintaining user trust in a gaming gambling platform. This route allows users to raise disputes or issues they encounter on the platform, particularly those related to transactions, wagers, tournaments, or potential glitches in games. The issues raised through this route can then be addressed and resolved by the platform's support or moderation team. Not implemented fully

Challenges

Similarly to the front end, tne challenging part of the project was dealing with the brackets. The way we imagined the brackets is similar to a reverse heap tree. By using the formulas following formulas we were able to get the winning teams that moved up a bracket:

Parent: floor((idx - 1) / 2)

Right: (idx * 2) + 1

Left: (idx * 2) + 2

Database Design

wagerSchema:

`name`: A String that represents the name of the wager. It is a required field.

`image`: A Buffer that stores image data, which is typically used to hold an image related to the wager. It's also a required field. The `contentType` is also stored for the image to understand its format.

`referee`: An ObjectId that references a 'User' document. This is likely the user who acts as the referee for the wager. This is a required field.

`isOpen`: A Boolean that determines whether the wager is currently open. It defaults to true, indicating that the wager is open by default when created.

`matchType`: A String that indicates the type of match. It can only be 'ONEVONE' or 'TEAMVTEAM'. This field is also required.

`betAmount`: A Number that represents the amount of money placed as a bet for this wager. It is a required field.

`startDate`: A Date representing the start date of the wager. This is also a required field.

`endDate`: A Date representing the end date of the wager. It's a required field as well.

`game`: A String that denotes the game associated with the wager. This field is required.

`gameMode`: A String to denote the mode of the game for the wager. It is a required field.

`winner`: An ObjectId that references a 'User' or 'Team' document depending on the match type. This field is likely updated when the wager is resolved.

`platforms`: A String that indicates the gaming platform(s) for the wager ('PlayStation', 'Xbox', 'PC', 'Console', or 'All'). This field is required.

`dateSubmitted`: A Date marking when the wager was submitted or created.

This schema represents a 'Wager' in the application. A 'Wager' is an event where two parties place a bet on a match (either one-on-one or team versus team). The match is overseen by a referee, and the schema contains information about the match, the game, the platform, and the bet amount. It also tracks the start and end dates for the wager and stores an image related to the wager. This information is used to manage and track wagers within the application.

userSchema:

`username`: A String that represents the username of the user. It is a required field, has to be between 12 and 32 characters, and it's included when queried.

`email`: A String that represents the user's email. It is unique, case-insensitive, and validated against an email regex.

`profilePicture`: A Buffer that contains the user's profile picture.

`isDisabled`: A Boolean that indicates whether the user is disabled. It defaults to false.

`role`: A String that represents the role of the user. Can be 'user', 'referee', or 'admin'. Defaults to 'user'.

`password`: A String that contains the user's password. It's required and must be at least 8 characters long.

`passwordConfirm`: A String that matches the password for confirmation. It's not stored in the database (as indicated by the pre-save hook `doNotSavePasswordConfirm`).

`intro`: A brief String describing the user. It should not exceed 150 words.

`startDate`: The date the user joined.

`dateDisabled`: The date the user was disabled.

`passwordChangedAt`: The date the user last changed their password.

`passwordResetToken`: A String used for password reset operations.

`passwordResetExpires`: The date the password reset token expires.

`active`: A Boolean to indicate whether the user is active.

`friends`: An Array containing the user's friends.

`tournamentHistory`: An Array containing the tournaments the user participated in.

`matchHistory`: An Array containing the matches the user participated in.

`privateKey`: A reference to a 'PrivateKey' document associated with the user.

`credits`: A Number representing the user's available credits.

`subscription`: A Number representing the user's subscription duration in months.

`riotAccount`, `psAccount`, `xboxAccount`, `steamAccount`: Strings representing the user's gaming accounts on various platforms.

`referees`: A list of 'User' documents who act as referees for the user.

`resolvedDisputes`: A list of resolved 'Dispute' documents related to the user.

`unresolvedDisputes`: A list of unresolved 'Dispute' documents related to the user.

`userSchema` includes methods to perform various operations such as changing passwords, creating a password reset token, and verifying a password. Virtual fields `hasGamingAccounts` and `hasSubscription` are derived from other fields in the schema.

Lastly, `userSchema` is used to create the 'User' model in Mongoose, which is then exported for use elsewhere in the application. This model represents a user in the application, storing their information, game history, friends, disputes, and more.

tournamentSchema:

`name`: A String that represents the name of the tournament. It is a required field and included when queried.

`image`: A Buffer that stores an image associated with the tournament.

`referee`: An ObjectId that references a 'User' document (the referee for the tournament).

`isOpen`: A Boolean indicating whether the tournament is open for registration or participation.

`matchType`: A String specifying the type of matches in the tournament - 'ONEVONE' or 'TEAMVTEAM'.

`teams`: An Array of ObjectId's referencing 'Team' documents participating in the tournament. This field is only selected if the `matchType` is 'TEAMVTEAM'.

`participants`: An Array of ObjectId's referencing 'User' documents participating in the tournament. This field is only selected if the `matchType` is 'ONEVONE'.

`bracketsID`: An ObjectId that references a 'Brackets' document associated with the tournament.

`prize`: A Number indicating the prize for the tournament.

`entryFee`: A Number indicating the entry fee for the tournament.

`startDate`: A Date indicating when the tournament starts.

`endDate`: A Date indicating when the tournament ends.

`limit`: A Number that represents the participant limit for the tournament. The possible values are 8, 16, or 32.

`game`: A String that specifies the game being played in the tournament.

`gameMode`: A String that specifies the game mode of the tournament.

`winner`: An ObjectId that references either a 'User' or 'Team' document, depending on the `matchType`.

`platforms`: A String indicating the gaming platform(s) for the tournament. The options are 'PlayStation', 'Xbox', 'PC', 'Console', or 'All'.

The 'Tournament' model is created using the `tournamentSchema` and exported for use elsewhere in the application. This model represents a tournament within the system, holding information about the teams, participants, prize, start and end dates, and more.

teamSchema:

`name`: A String that represents the name of the team. This is a required field and is selected during queries.

`privateKeys`: An ObjectId that references a 'PrivateKey' document. This field is mandatory and represents the unique private keys associated with each member of the team. It is selected during queries.

`image`: A Buffer that holds an image file. This is the team's logo and is a required field.

`description`: A String describing the team. The maximum length of this field is 100 characters.

The 'Team' model is created using the `teamSchema` and is exported for use elsewhere in the application. This model represents a team within the system, including information about the team name, logo, description, and the private keys of its members.

privateKey:

The `privateKeySchema` is a simple schema that represents the private keys of users or teams within the system. It consists of:

> **`value`**: A String field representing the value of the private key.

The 'PrivateKey' model is created using the `privateKeySchema` and is exported for use elsewhere in the application. This model is essential for maintaining secure user authentication and secure team communication within the application. It contains the value of a private key associated with a user or team in the system.

matchSchema:

The `matchSchema` is a complex schema representing a match within the system. It contains fields to describe the type of the match, its association with either a Tournament or a Wager, details about the participating parties, and the final winner. It comprises:

> `matchType`: This field represents the type of match. The enum restricts the values to either 'TOURNAMENT' or 'WAGER', with 'TOURNAMENT' as the default. This field is required.
>
> `tournamentID`: This field references the 'Tournament' model. It contains the ID of a tournament if the match type is a tournament. This field is selected only if the match type is 'TOURNAMENT'.

`bracketsID`: This field references the 'Brackets' model. It contains the ID of the brackets if the match type is a tournament. This field is selected only if the match type is 'TOURNAMENT'.

`bracketIndex`: This field represents the index of the bracket in a tournament. It's selected only when the match type is 'TOURNAMENT'.

`wagerID`: This field references the 'Wager' model. It contains the ID of a wager if the match type is a wager. This field is selected only if the match type is 'WAGER'.

`partyA`: This field references either the 'Team' model or the 'User' model, depending on whether the match type is a tournament or a wager. It contains the ID of the first party participating in the match. It's selected only when the match type is 'TOURNAMENT'.

`partyB`: This field, like 'partyA', references either the 'Team' model or the 'User' model, depending on the match type. It contains the ID of the second party participating in the match. It's selected only when the match type is 'TOURNAMENT'.

`winner`: This field references either the 'User' model or the 'Team' model, depending on the match type. It contains the ID of the winning party.

`dateSubmitted`: This field represents the date when the match was submitted.

The 'Match' model is created using the `matchSchema` and is exported for use elsewhere in the application. This model is central to understanding the dynamics of matches, either tournament matches or wager matches, within the application.

bracketsSchema:

The `bracketsSchema` represents the brackets within a tournament. It has fields to associate the bracket with a tournament, the type of matches that will be held in the tournament, and a list of matches associated with this bracket. It consists of:

`tournament`: This field references the 'Tournament' model and holds the ID of the associated tournament. This field is required and is always selected.

`matchType`: This field represents the type of match. The enum restricts the values to either 'ONEVONE' or 'TEAMVTEAM'. This field is required.

`matches`: This field is an array of IDs referencing the 'Match' model. It holds the IDs of the matches associated with this bracket. This field is always selected.

The 'Brackets' model is created using the `bracketsSchema` and is exported for use elsewhere in the application. This model is integral to manage and understand the progression of a tournament, as it houses all the matches under a particular tournament.

Final Notes:

- This API might be difficult to configure for hosting.