**Board Engine API v1**

Http API

boarge/API/v1

**/games/<Game ID>**

* HttpGET (GameCriteria criteria)
  + Parameters
    - Criteria (GameCriteria) - used to filter a search for all open games.
    - user (int) – ID of user to search for within list of all games
    - team (int) – ID of team to search for within list of all games
    - turns (boolean) - true if should return list of all games where it is user or team’s (whichever spec.) turn.
    - games (boolean) – true if should return list of all games user or team (whichever specified) is active in.
  + Description
    - Always returns a list of games.
    - Priority

1. GameID specified end of url, search games table for associated id
2. GameID not present in url, GameCriteria specified. Search games table for a list of all open games that match the criteria.
3. GameID not present in url, GameCriteria not specified, user is specified and turns is true. Search games table for all games where it is the user’s turn.
4. GameID not present in url, GameCriteria not specified, user is specified and games is true. Search games table for all games that user is active in.
   * Return
     + getGame(gameID) – if game id is specified at end of url
     + getOpenGames(GameCriteria) – if gameID is not specified in url and GameCriteria is specified
     + getAllTurns(userID) – if gameID and GameCriteria are not specified , user is specified, and turns is true
     + getAllGames(userID) - if gameID and GameCriteria are not specified , user is specified, and games is true
   * Example
     + <http://boarge/API/v1/games?user=9999&turns=true> will return a list of games where its user 9999’s turn

* HttpPOST
  + Parameters
    - CreateGameCriteria – criteria used to create game.
    - usersToInvite – all users to invite to game upon successful creation
  + Description
    - Response is a Game created based on the CreateGameCriteria specified. If ranked game type, ignore the usersToInvite list. Else (unranked) send game invite to all users within the list using invitePlayers() api call.
  + Return
    - createGame(userUniqueID, criteria, usersToInvite)
* HttpPUT
  + Parameters
    - UserID/TeamID – id of player(s) making move
    - GameState – GameState of game after making the move.
  + Description
    - Overwrites the GameState pointed to by gameID. The new game state encompasses the user’s submitted move for his/her turn.
  + Return
    - submitMove(gameID, userID, GameState)

**/games/join/<GameID>**

* HttpPOST (post me to that game)
  + Parameters
    - UserID or TeamID trying to join this game
  + Description
    - Response is the updated game after a successful join.
  + Return
    - joinGame(gameID, joiningID)

**/games/invite/<GameID>**

* HttpPOST (post me to that game)
  + Parameters
    - UserID or TeamID trying to join this game
  + Description
    - Response is the updated game after a successful join.
  + Return
    - invitePlayers(userID, int gameID, String[] usersToInvite)

Function list called by server

**joinGame**(userUniqueID, int gameID)

* Parameters
  + userUniqueID
  + gameID (int) – the id of the Game to join.
* Description
  + Attempt to add user to specified Game. (**how support team joining**).
  + If successful, send all players in game a notification with updated game state.
* Return
  + If successful, the updated state of the game that was joined (Game state may have changed if players have joined since last time user made get request). Null if could not join the game.
* Note
  + As soon as user has decided to join a Game, the screen should immediately update based on the Game that was clicked (plus user now joined). When a response returns from the server with the updated game state, make sure to update the UI with any differences.

**getOpenGames**(GameCriteria criteria)

* Parameters
  + userUniqueID
  + criteria (GameCriteria) – the criteria used to filtering the Game set.
* Description
  + Fetch all open games that fall under the set of game criteria specified by the user
* Return
  + The set of all open games that fit the game criteria specified by the user. Null if set of games is empty.

**getGame** (int gameID)

* Parameters
  + gameID – id of game trying to get
* Description
  + Fetch the game associated with gameID
* Return
  + The game associated with gameID. Null if set of games is empty.

**createGame**(userUniqueID, CreateGameCriteria criteria, usersToInvite)

* Parameters
  + userUniqueID -
  + criteria (CreateGameCriteria) - criteria used to restrict access to the game.
  + usersToInvite () – List of user accounts to invite to created game.
* Description
  + Creates a game based on the CreateGameCriteria specified. If ranked game type, ignore the usersToInvite list. Else (unranked) send game invite to all users within the list using invitePlayers() api call.
  + Note that when a game is private, it is automatically an unranked match. If public, a game may be ranked or played for fun.
* Return
  + The Game created. Null if unsuccessful.

**invitePlayers**(userID, int gameID, String[] usersToInvite)

* Parameters
  + userUniqueID
  + gameID (int) – Game id to invite users to.
  + usersToInvite – List of user accounts to invite to game.
* Description
  + Fetch game and send game-invite notifications to all users within usersToInvite list. If the client accepts the game invite, a new call to joinGame() should be made.
* Return
  + List of users that were unable to be invited. Null if successful.

**submitMove**(userUniqueID, int gameID, GameState gameState)

* Parameters
  + userUniqueID
  + gameID (int) – id of game to overwrite
  + gameState (GameState) – current state of the game that is being submitted.
* Description
  + Overwrites the GameState pointed to by gameID. The new game state encompasses the user’s submitted move for his/her turn.
* Return
  + Success/Fail code.

**getAllTurns**(userUniqueID)

* Parameters
  + userUniqueID
* Description
  + Fetch all games that are waiting on this user to make a turn.
* Return
  + A list of all games that are waiting on this user to make a turn. Null if empty.

**getAllGames**(userUniqueID)

* Parameters
  + userUniqueID
* Description
  + Fetch all games user is active in.
* Return
  + A list of all games that the user is active in. Null if empty.

Table Objects:

**Users**

* Table of user Accounts

**Teams**

* Table of teams

|  |  |  |
| --- | --- | --- |
| Unique\_id | Team\_name | Array of users |

**Games**

* Table all active games

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Unique\_id | GameCriteria (should just spell out in table?) | GameState JSON | Team array joined | Team up | LastUpdated |

GameCriteria

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| private | ranked | difficulty | Num\_teams | Num\_players\_per\_team | Time\_limit\_per\_move | TurnStrat |