Last Updated: 9/20/21

Cave Firebase (Backend) Documentation

Last Updated: 9/20/21

Callable Cloud Function Information

1. Create Bet

HTTPS Callable	createBet()

```
REQUEST BODY
{
 description : description of bet,
 name : name of bet,
 odds : odds line of bet,
 visibility : PUBLIC or PRIVATE,
 isFreePlay : true | false,
 wagerAmount : amount wagered in this bet,
 wagers : [
     {
         game : gameid | SEASON,
         sport : sport,
         team : teamid (only for GAME_PROP),
         player : playerid (only for PLAYER_PROP),
         amount : 3 (SPREAD) | 44.5 (O/U) | 200 (PROP), must be a number
         statComparison : EQUAL, OVER, UNDER
         stat : wins, passyards, rushyards, etc.,
         wagerType : MONEYLINE | SPREAD | OVERUNDER | GAME_PROP | PLAYER_PROP
     }
 ]
}
```

```
RESPONSE BODY
{
 success: true,
 betId: id of the new bet
}
```

Last Updated: 9/20/21

Follow/Favorite Request Functions

User A wants to follow User B

1. Check User B's user document and see if it is public or private
2. If private, then bets and other info should be hidden and the *Follow* box should say *Request Follow*
3. After pressing the Follow/Request Follow button, then the app should call **Submit Request** with requestType=FOLLOW and recipient=User B's uid
4. If User B is public, then the request is accepted automatically
5. If User B is private, then they must accept the request with **Accept Request**

User A wants to favorite User B

1. Check that User A is following User B (by looking at User B followers doc)
    a. If not following, don't show the star symbol.
2. After pressing the star button to favorite, then the app should call **Submit Request** with requestType=FAVORITE and recipient=User B's uid
    a. At this point favorite requests are automatically accepted

View/Accept Inbound Requests

1. Get user's requests document (check requests schema for more info)
2. Separate and show the *followInbound* (FOLLOW request user ids) and *favoriteInbound* (FAVORITE request user ids) with the option to accept any of them (using the requestor's uid and the **AcceptRequest** function)

View/Cancel Outbound Requests

1. Get user's requests document (check requests schema for more info)
2. Separate and show the *followInbound* (FOLLOW request user ids) and *favoriteInbound* (FAVORITE request user ids) with the option to cancel any of them (using the recipient's uid and the **Cancel Request** function)

2. Submit Request

HTTPS Callable	submitRequest()

```
REQUEST BODY
{
 requestType: FOLLOW | FAVORITE,
 recipient: id of the user receiving the request,
}
```

```
RESPONSE BODY
{
 success: true
}
```

3. Accept Request

HTTPS Callable	acceptRequest()

```
REQUEST BODY
{
 requestType: FOLLOW | FAVORITE,
 requestor: id of the user that sent the request,
}
```

```
RESPONSE BODY
{
 success: true
}
```

4. Cancel Request

HTTPS Callable	cancelRequest()

```
REQUEST BODY
{
 requestType: FOLLOW | FAVORITE,
 recipient: id of the user that would be receiving the request,
}
```

```
RESPONSE BODY
{
 success: true
}
```

User Profile Functions

Database Query to see if a handle exists:

```
db.collection("users").whereField("handle",isEqualTo:"{testVal}").limit(to:1)
```

5. Create User

HTTPS Callable	createUser()

Create User Flow:

1. **Sign user up in Firebase Authentication**
2. **Present the Create User screen to get fields specified in request**
3. **After user inputs their handle/username, use the Database Query above to check if a user exists with that handle**
    a. **If user already exists, the field should turn red with a message that the handle is taken (button to Create User should be disabled until the user enters a new handle that is not taken)**
4. **If user does not exist, then call Create User with the fields from the screen**
5. **Make sure the Create User call succeeds, then proceed to the main menu (do not update the user document, don't call updateName or updateHandle)**

```
REQUEST BODY
{
 handle: handle of the user,
 name: name of the user
 bio : the user's bio/about section
 website : user's website,
 photoUrl : url of the user's profile picture
}
```

```
RESPONSE BODY
{
 success: true
}
```

6. Update Name

HTTPS Callable	updateName()

Updates user's name

```
REQUEST BODY
{
 name: new name of the user
}
```

```
RESPONSE BODY
{
 success: true
}
```

7. Update Handle

HTTPS Callable	updateHandle()

Updates user's handle (if it's not taken). Throws an error if the handle is taken.

```
REQUEST BODY
{
 handle: new handle of the user
}
```

```
RESPONSE BODY
{
 success: true
}
```

Last Updated: 9/20/21

Push Notification Functions

These functions are used to keep a record of Firebase Cloud Messaging Registration Tokens for a given user. When a user creates an account or signs in, the registration token of that device/instance should be added using the **Add Notification Token** function
Info on getting the registration token

8. Add Notification Token

HTTPS Callable	addNotificationToken()

```
REQUEST BODY
{
 token: string, push notification device token
}
```

```
RESPONSE BODY
{
 success: true
}
```

9. Remove Notification Token

HTTPS Callable	removeNotificationToken()

```
REQUEST BODY
{
 token: string, push notification device token
}
```

```
RESPONSE BODY
{
 success: true
}
```

Last Updated: 9/20/21

Miscellaneous Queries

Search by handle:

```
db.collection("users").whereField("handle",isEqualTo:"{searchVal}"+"\uf8ff")
```

Search by name:

```
db.collection("users").order(by:"searchableIndex.{searchVal}").limit(to: 5)
```

Get ranking of favorites:

Start with some limit. Show more if the user clicks “View More”.

```
db.collection("subscribers")
      .whereField("subscribers", arrayContains: "{logged-in-user-id}")
      .order(by: "dailyPercentage" | "weeklyPercentage")
      .limit(to: 5)
```

Get ranking of following:

Start with some limit. Show more if the user clicks “View More”.

```
db.collection("followers")
      .whereField("followers", arrayContains: "{logged-in-user-id}")
      .order(by: "dailyPercentage" | "weeklyPercentage")
      .limit(to: 5)
```

Last Updated: 9/20/21

Bet and Game Management (Backend Only)

1. TODO Open Items

- **[updateName]** Update chats with a user's name in title when name changes
- **[refreshTeams]** Create cumulative player/team stats for the year
- **[refreshPlayers]** Get position and number for players
- **[refreshPastBoxScore]** Implement refreshPastBoxScore
- **[refreshPregameOdds/refreshLiveStats]** Update schedule triggers to be every min. Also need to fix some bugs
- **[Push Notifications]** Implement Push Notifications functions
- **[season-ended]** Handle what happens at the end of the season (clean up bets, record stats, etc)
- **[getBetOutcome]** Handle special type bets (FUTURE, OTHER)
- **[All functions]** Clean up awaits, use transactions and batched writes wherever possible to speed up performance

2. Getting Sports Data from Goalserve

a. refreshTeams (refreshTeamsForSport)

Scheduled	**Every Day 1am EST**

Loops through sports and calls *refreshTeamsForSport* for each of them. For each sport, go through the standings and get all the teams. Create or Update a document for each team with the team name and sport.

TODO:

- **Loop through the team's game documents to generate cumulative stats**

b. refreshPlayers (refreshPlayersForSport)

Scheduled	**Every Day 1:30am EST**

Loops through sports and calls *refreshPlayersForSport* for each of them.
For each sport, get all teams from Firestore for the sport.
For each team, get the team roster and Create or Update a document for each player with player name, team, sport, etc.

TODO:

- **Need to get the position and number for each player**

c. refreshPregameOdds (refreshPregameOddsForSport)

Scheduled	**PROD: Every minute** **Current: Every 15min**

Loops through sports and calls *refreshPregameOddsForSport* for each (start_date= today | end_date= today + 7 days).
Get pregame odds from Goalserve for each sport. Create or Update documents for each game with game ID, status, pregame odds, and other game info.

d. refreshLiveStats (refreshLiveStatsForSport)

Scheduled	**PROD: Every minute** **Current: Every 15min**

Loops through sports and calls *refreshLiveStatsForSport* for each sport.
Get all games from Firestore that are not COMPLETE. Collect a list of all of their dates.
For each date, get scores for every game from Goalserve for that date.
For each game on the date, update the game document with latest stats and status. If the game has ended, publish a message to game-ended, otherwise publish a message to game-updated.

e. TODO refreshPastBoxScore()

Scheduled	**Every Day 3am EST**

TODO: Need to get scores for date=yesterday, date=2 days ago, date=3 days ago, and update the bets for any stats/outcomes that changed.

3. TODO Push Notifications

Unimplemented, need to create push notification messages.
Need to come up with and include triggers.

a. **TODO sendPushNotificationToUser()**
b. **TODO subscribeUserToTopic()**
c. **TODO unsubscribeUserFromTopic()**

4. Stats Management

a. resetDailyRecord

Scheduled	**Every Day 4am EST**

Resets every user's daily record to be a blank stats object (0-0-0).

b. resetWeeklyRecord

Scheduled	**Every Tuesday 4:05am EST**

Resets every user's daily record to be a blank stats object (0-0-0). Also resets each user's *games* list in their followers and subscribers documents.

5. Pubsub Data Management

a. Game Updated [game-updated]

Pubsub Message	game-updated

If the game is live, update all Bet Options for this game and update their *currentAmount* and *onTrack* fields along with a timestamp. Get all bets that include this game that are UPCOMING and set the status to LIVE.

```
MESSAGE PAYLOAD
{
 gameId: string, ID of the game that has been updated
}
```

b. Game Ended [game-ended]

Pubsub Message	game-ended

Find all Bet Options for this game and determine their *outcome* and update their status to COMPLETE. Publish the bet-option-completed message with a list of all bet options changed to COMPLETE.

```
MESSAGE PAYLOAD
{
 gameId: string, ID of the game that has ended
}
```

c. Bet Option Completed [bet-option-completed]

Pubsub Message	bet-option-completed

Loop through completed bet options in payload. Find bets that contain these bet options: update status of wagers within bet (to outcome of Bet Option) and determine if overall bet status can be made COMPLETE (if there is a LOSS or all bet options are COMPLETE). For Bets that are done and marked as COMPLETE, publish one message to bet-completed with a list of completed bets.

```
MESSAGE PAYLOAD
{
 betOptions: list of {id: string, outcome: string} completed bet option ids
}
```

d. Bet Completed [bet-completed]

Pubsub Message	bet-completed

Loop through completed bet IDs in payload. Update user's stats based on bet outcome. Update win percentages (for ranking) in user's followers and subscribers documents. Update the *nextBet* and *nextBetId* fields in the user's subscribers document.

```
MESSAGE PAYLOAD
{
 betIds: list of strings (completed bet option ids)
}
```

e. TODO Reset Season Record [season-ended]

Pubsub Message	season-ended

Resets each user's seasonRecord to a blank record for the next season of the sport sent in the payload. Saves the previous season record.

TODO:

- **Need to come up with a trigger for this**
- **Need to close out bets that had bet options for the SEASON**

```
MESSAGE PAYLOAD
{
 sport: string [NFL | NBA | NHL],
 nextSeason: string [2022, 2023, etc.]
}
```

Last Updated: 9/20/21

Firestore Database Schema

Collections:

1. bets

```
Bet Document
{
    "name": "Name of bet",
    "description": "Description of bet",
    "odds": 100, // number, positive for + and negative for - odds
    "visibility": PUBLIC | PRIVATE,
    "wagerAmount": 50,
    "games": [
        "gameid123",
        "gameid456"
    ],
    "outcome": WIN | LOSS | PUSH,
    "startDate": 1605575700, // Number, unix timestamp
    "status": COMPLETE | UPCOMING | LIVE,
    "user": "uid",
    "wagers": [
        {
             "betOption" : "12345",
             "outcome" : WIN | LOSS | PUSH,
             "sport" : NBA | NFL | MLB | NHL,
             "status" : COMPLETE | UPCOMING | LIVE
        }
    ],
    "betOptions": [
        "12345",
        "67890"
    ],
    "toWin": 50,
    "isFreePlay": true | false
}
```

2. chats

```
Chat Document
{
    "lastMessage" : {
        "message" : "content of message",
        "sender" : "sender-uid",
        "timestamp" : November 27th, 200 at 12:36:37 AM UTC-8 // Firebase timestamp
    },
    "members" : [
        "uid123",
        "uid456"
    ],
    "title" : "Title of chat"
}
```

a. messages (subcollection of chat)

```
Message Document
{
    "message" : "Content of message",
    "sender" : "uid123",
    "timestamp" : Firebase timestamp,
    "bet" :"betid123" // Optional, bet ID of attached bet
}
```

3. followers

```
Followers Document
{
    "name": "User Name",
    "yearlyWinPercentage": 0,
    "alltimeWinPercentage": 0,
    "weeklyWinPercentage": 0,
    "dailyWinPercentage": 0,
    "followers": [
        "uid123",
        "uid456"
    ]
}
```

4. requests

```
Requests Document
{
    "followInbound": ["uid123"],
    "followOutbound": ["uid456"],
    "favoriteInbound": ["uid789"],
```

```
    "favoriteOutbound": ["uid000"]
}
```

5. settings

```
Settings Document
{
    "publicProfile": true | false,
    "publicBetAmount": true | false
}
```

6. stats

```
Stats Document
{
    "overall" : {
        "allTimeRecord" : {
            "losses" : 0,
            "wins" : 0,
            "ties" : 0,
            "stringValue" : "0-0-0",
            "winnings" : 0,
            "public" : true
        },
        "yearlyRecord" : {
            "losses" : 0,
            "wins" : 0,
            "ties" : 0,
            "stringValue" : "0-0-0",
            "winnings" : 0,
            "public" : true,
            "year" : "2021"
        },
        "weeklyRecord" : {
            "losses" : 0,
            "wins" : 0,
            "ties" : 0,
            "stringValue" : "0-0-0",
            "winnings" : 0,
            "public" : true,
            "week" : "03.03.2021" // Tuesday that the week started on
        },
        "dailyRecord" : {
            "losses" : 0,
            "wins" : 0,
            "ties" : 0,
            "stringValue" : "0-0-0",
            "winnings" : 0,
            "public" : true,
            "date" : "03.03.2021"
        },
```

```
        "streak" : {
            "numberValue" : 0,
            "stringValue" : "-",
            "public" : true
        }
    },
    "NFL" : {...}, // Same as "overall" but no "winnings"  fields
    "NBA" : {...}, // Same as "overall" but no "winnings"  fields
    "MLB" : {...}, // Same as "overall" but no "winnings"  fields
    "NHL" : {...}  // Same as "overall" but no "winnings"  fields
}
```

7. subscribers

```
Subscribers Document
{
    "name": "User Name",
    "nextBet": null,
    "yearlyWinPercentage": 0,
    "alltimeWinPercentage": 0,
    "weeklyWinPercentage": 0,
    "dailyWinPercentage": 0,
    "subscribers": [
        "uid123",
        "uid456"
    ]
}
```

8. users

```
User Document
{
    "bio": "",
    "name": "User Name",
    "favoriteTeams": [
        "teamid123",
        "teamid456"
    ],
    "notificationTokens": ["12345678"],
    "numFollowing": 0,
    "website": "",
    "handle": "@Username"
    "searchableIndex": {
        "u" : true,
        "us" : true,
        "use" : true,
        "user" : true,
        "n" : true,
        "na" : true,
        "nam" : true,
        "name" : true
```

```
        "user n" : true,
        "user na" : true,
        "user nam" : true,
        "user name" : true
    }
}
```

Read Only Collections:

1. betOptions

```
BetOption Document
{
    "amount" : 6,
    "currentAmount" : 5,
    "game" : "90958",
    "onTrack" : true | false,
    "outcome" : WIN | LOSS| PUSH,
    "statComparison" : EQUAL | OVER | UNDER,
    "wagerType" : MONEYLINE | SPREAD | OVERUNDER | GAME_PROP | PLAYER_PROP,
    "sport" : NBA | NFL | MLB | NHL,
    "team" : "teamid123", // If GAME_PROP
    "player" : "playerid123", // If PLAYER_PROP
    "timestamp" : November 27th, 200 at 12:36:37 AM UTC-8 // Firebase timestamp of
latest currentAmount/onTrack
}
```

2. constants

a. enums (/constants/enums)

```
{
    "betType" : [
        "MONEYLINE",
        "OVERUNDER",
        "SPREAD",
        "GAME_PROP",
        "PLAYER_PROP"
    ],
    "gameStatus" : [
        "UPCOMING",
        "LIVE",
        "COMPLETE"
    ],
    "outcomes" : [
        "WIN",
        "LOSS",
        "PUSH"
    ],
```

```
    "sport" : [
        "NFL",
        "NBA",
        "MLB",
        "NHL"
    ],
    "statComparison" : [
        "OVER",
        "UNDER"
    ]
}
```

b. help (/constants/help)

```
{
    "faq": [
        {
            "question": "question",
            "answer": "answer"
        }
    ]
}
```

c. privacypolicy (/constants/privacypolicy)

```
{
    "title" : "Privacy Policy",
    "text" : "Sample privacy policy text"
}
```

d. stats_{SPORT} (ex. /constants/stats_NBA)

3. games

```
Game Document
{
    "awayTeam": "1219",
    "awayTeamName" : "Indiana Pacers",
    "formattedDate" : "3.03.2021",
    "homeTeam" : "1183",
    "homeTeamName" : "Cleveland Cavaliers",
    "sport" : NBA | NFL | MLB | NHL,
    "startDate" : 1614812400,
    "status" : UPCOMING | LIVE | COMPLETE,
    "winner" : "",
    "odds" : {
        "timestamp" : 1614758539057,
        "moneyline" : {
            "1219" : -230,
            "1183" : 190
```

```
        },
        "overunder" : {
            "amount" : 216,
            "over_line" : -110,
            "under_line" : -110
        },
        "spread" : {
            "1183" : {
                "amount" : 6,
                "line" : -105
            },
            "1219" : {
                "amount" : -6,
                "line" : -105
            }
        }
    },
    "stats" : {
        "player" : {
            "playerid123" : {
                "stat_name" : stat_val
            }
        },
        "team" : {
            "teamid123" : {
                "stat_name" : stat_val
            }
        }
    }
}
```

4. players

```
Player Document
{
    "name": "Cam Newton",
    "team": "teamid123",
    "sport": NFL | NBA | MLB | NHL,
}
```

5. teams

```
Team Document
{
    "fullName": "Carolina Panthers",
    "sport": NFL | NBA | MLB | NHL,
}
```

Last Updated: 9/20/21