Table Descriptions

1. User table
   1. userID integer primary key autoincrement unique,
   2. username text unique not null,
   3. password text not null,
   4. email text unique not null,
   5. dob text not null,
   6. isonline bit not null,
   7. privacy bit not null,
   8. currLobbyID integer
2. Messages table
   1. messageID integer primary key autoincrement unique,
   2. senderID integer not null,
   3. receiverID integer not null,time datetime unique not null,
   4. message text not null
3. Lobbies table
   1. lobbyID integer primary key autoincrement unique,
   2. name text unique not null,
   3. gameType text not null,
   4. lat float unique not null,
   5. long float unique not null
4. Friends table
   1. id integer primary key autoincrement unique,
   2. user1ID integer not null,
   3. user2ID integer not null
5. Blocking table
   1. id integer primary key autoincrement unique,
   2. blockerID text not null,
   3. blockedID text not null
6. Statistics table
   1. userID integer primary key autoincrement unique,
   2. userid integer not null,
   3. pongWins integer not null

Server Call Documentation

This is a REST-ish API. All sent information will use json requests, even GET calls. ID is the row number in the database, cannot be changed and will be used as the de-facto identification of rows. If you just want to test the database without a client, use Insomnia REST client. Any calls that don’t involve sending information will respond with two strings, “success”, which is a Boolean, and “message”.

1. /users
   1. GET: search for users
      1. Send “username” to search for users by username
      2. Send “userID” to search for users by id, int
      3. Send “search” to search for usernames containing the string
      4. Send “lobby” to get all users in the lobby with id “lobby”, int
      5. Send nothing to get all users
      6. Example Query:

/users?userID=1

* + 1. Example Response:

[

{

"currGameID": null,

"dob": "10/10/1998",

"email": "asdf@asdf.com",

"userID": 2,

"isonline": 1,

"loggedIn": 1,

"password": "asdfasdf",

"privacy": 1,

"username": "asdf"

},

{

"currGameID": null,

"dob": "10/10/1998",

"email": "asdff@asdf.com",

"userID": 7,

"isonline": 1,

"loggedIn": 1,

"password": "asdfasdf",

"privacy": 1,

"username": "asd"

}

]

* 1. POST: Add user
     1. Must send all the fields below!
     2. “username”, username
     3. “password”, password
     4. “email”, email
     5. “dob”, date of birth
     6. “privacy”, social privacy setting, int
     7. Example Query:

{

"username": "asdf",

"password": "asdfasdf",

"email": "asdf@asdf.com",

"dob": "10/10/1998",

"privacy": 1

}

* + 1. Example Response:

{

"message": "Added new user",

"success": true

}

* 1. PUT: Update user field
     1. Must update each field one at a time, ID is required!
     2. “username”, username
     3. “password”, password
     4. “email”, email
     5. “dob”, date of birth
     6. “privacy”, social privacy setting, int
     7. “isOnline”, online status, int
     8. “currLobbyID”, current game id, can be null, int.
     9. Example Query:

{

"userID": 1,

"username": "bob"

}

* + 1. Example Response:

{

"message": "Updated username",

"success": true

}

* 1. DELETE: delete user
     1. Send “userID” to delete user with id, int
     2. Example Query:

DELETE /users/1

* + 1. Example Response:

{

"message": "Deleted user",

"success": true

}

1. /messages
   1. GET: get messages with receiverID. Messages are deleted from database once read
      1. “receiverID”, user id of receiver, int
      2. Example Query:

GET /messages?receiverID=7

* + 1. Message fields: senderID (int), receiverID (int), message (string), time (string)
    2. Example Response:

[

{

"messageID": 2,

"message": "Hey bob",

"receiver": "7",

"sender": "2",

"time": "2018-11-16 20:38:31.358376"

},

{

"messageID": 3,

"message": "Hey bob",

"receiver": "7",

"sender": "2",

"time": "2018-11-16 20:38:33.021672"

},

{

"messageID": 4,

"message": "Hey bob",

"receiver": "7",

"sender": "2",

"time": "2018-11-16 20:38:33.877553"

}

]

* 1. POST: send messages
     1. “senderID”, user id of the sender, int
     2. “receiverID”, user id of the receiver, int
     3. “message”, message to be sent
     4. Example Query:

{

"senderID": 2,

"receiverID": 7,

"message": "Hey bob"

}

* + 1. Example Response:

{

"message": "Sent message",

"success": true

}

1. /lobbies
   1. GET: get list of lobbies
      1. “gameType”, string of gameType
      2. Example Query:

GET /lobbies?gameType=pong

* + 1. Example Response:

[

{

"gameType": "pong",

"lobbyID": 1,

"latCoord": 12.829347928,

"longCoord": 11.231232323,

"name": "lobby1"

}

]

* 1. POST: Add lobby
     1. “name”, string of lobby name
     2. “gameType”, string of gameType
     3. “latCoord”, float of latitude, float
     4. “longCoord”, float of longitude, float
     5. Example Query:

{

"name": "lobby1",

"gameType": "pong",

"latCoord": 1.2829347928E+1,

"longCoord": 1.1231232323E+1

}

* + 1. Example Response:

{

"message": "Added lobby",

"success": true

}

* 1. DELETE: Delete lobby
     1. “lobbyID”, id of the lobby, int
     2. Example Query:

DELETE /lobbies/1

* + 1. Example Response:

{

"message": "Deleted lobby",

"success": true

}

1. /report
   1. GET: Get number of reports for userID
      1. “userID”, id of the user, int
      2. Example Query:

GET /report?userID=1

* + 1. Example Response:

{

“reports”: 3

}

* 1. POST: report user
     1. “userID”, id of the user, int
     2. Example Query:

{

“userID”: 1

}

* + 1. Example Response:

{

“success”: True,

“message”: “User has been reported”

}

1. /block
   1. GET: check if “blocker” has blocked “blocked”
      1. “blockerID”, userID of the blocker
      2. “blockedID”, userID of the user to check
      3. Example Query:

GET /block?blockerID=1&blockedID=2

* + 1. Example Response:

{

“result”: 1

}

* 1. POST: block user
     1. “blockerID”, userID of the blocker
     2. “blockedID”, userID of the user to be blocked
     3. Example Query:

{

“blockerID”: 1,

“blockedID”: 2

}

* + 1. Example Response:

{

“success”: True,

“message”: “User successfully blocked”

}

* 1. PUT: unblock user
     1. “blockerID”, userID of the blocker
     2. “blockedID”, userID of the user to be blocked
     3. Example Query:

{

“blockerID”: 1,

“blockedID”: 2

}

* + 1. Example Response:

{

“success”: True,

“message”: “Block successfully deleted”

}

1. /stats
   1. GET: Get stats for userID
      1. “userID”, userID for user, int
      2. Example Query:

GET /stats?userID=1

* + 1. Example Response:

{

“id”: 5,

“userID”: 1,

“pongWins”: 20

}

* 1. PUT: Update stats for userID
     1. Only updates one stat at a time!
     2. “userID”, userID for user, int
     3. “pongWins”, pongWins for user, int
     4. Example Query:

{

“userID”: 1,

“pongWins”: 10

}

* + 1. Example Response:

{

“success”: True,

“message”: "Updated pongWins"

}

1. /friends
   1. GET: check if “user1ID” is friends with “user2ID”
      1. “user1ID”, userID of user 1
      2. “user2ID”, userID of user 2, optional. Use only user1ID to return all friends of user1
      3. Example Query:

GET /friends?user1ID=1&user2ID=2

* + 1. Example Response:

{

“result”: 1

}

* 1. POST: add user as friend
     1. “user1ID”, userID of the user 1
     2. “user2ID”, userID of the user 2
     3. Example Query:

{

“user1ID”: 1,

“user2ID”: 2

}

* + 1. Example Response:

{

“success”: True,

“message”: “User successfully blocked”

}

* 1. PUT: remove user as friend
     1. “user1ID”, userID of the user 1
     2. “user2ID”, userID of the user 2
     3. Example Query:

{

“user1ID”: 1,

“user2ID”: 2

}

* + 1. Example Response:

{

“success”: True,

“message”: “Friend successfully deleted”

}