API DOCUMENTATION FOR DOI

Server URL: https://doi-apis.fly.dev/

1. USER REGISTRATION WITH DEVICE ID:

   This creates a new user account with the unique device ID

   Route: /user/deviceID/registration

   Body:{

   "deviceID" (required),

   "username" (required),,

   "country" (required),,

   "avatar" (required),

   }

   Request: ”POST”,

   Response: {

   status:200 ,

   data:{

   type:"registrationsuccess" ,
   msg:jwtToken

   }

   }

2. USER AUTH WITH DEVICE ID:

   This returns the userID

   Route: /user/deviceID/auth

   Body:{

   "deviceID" (required),,

   }

Request: ”POST”,

Response: {

status:200 ,

data:{

type:"IDAuthSuccess" ,
msg:jwtToken

}

}

3. USER ACCOUNT EMAIL SYNC:

This attaches an Email and password to users account to enable later syncing

Route: /user/sync/signup

Body:{

"email" (required),,

"password" (required),,

"jwtToken" (required),

}

Request: ”POST”,

Response: {

status:200,

data:{

type:"syncSuccessful" ,

msg:"User account synced"

}

}

4. USER ACCOUNT SYNC LOGIN:

   This retrieves userID from synced accounts

   Route: /user/sync/login

   Body:{

   "email" (required),

   "password" (required),

   "deviceID" (required)//needed to update the deviceID for new device

   }

   Request: "POST",

   Response: {

   status:200 ,

   data:{

   type:"loginSuccessful" ,

   msg:jwtToken

   }

   }

5. ADD USERNAMES TO LIST OF USERNAME

   This is used to add username to user's list of usernames

   Route: /user/add/username

   Body:{

"username" (required),,

"jwtToken" (required),

}

Request: ”POST”,

Response: {

status:200 ,

data:{

type:"nameAddSuccessful" ,

msg:updated array of existing usernames

}

}

6. UPDATE USER COUNTRY

This is used to simply update user’s country

Route: /user/update/country

Body:{

"jwtToken" (required),,

"country" (required),

}

Request: ”PUT”,

Response: {

status:200 ,

data:{

type:"countryUpdateSuccessful" ,

msg:" country updated successfully"

}

}

7. UPDATE USER GAME AVATAR

This is used to update user's avatar,(the avatar is stored as a text value , which could be a representation of the name or path of the avatar

Route: /user/update/avatar

Body:{

"jwtToken" (required),

"avatar" (required)

}

Request: "PUT",

Response: {

status:200 ,

data:{

type:"avatarUpdateSuccessful",

msg:"avatar updated successfully"

}

}

8. HOST CREATE GAME SESSION

This Endpoint enables host create game sessions, if a session already exist, it will be updated as all players can only host a single game at a time

Route: /host/multiplayer/createGame

Body:{

jwtToken (required),

gameDuration (required),

playersCount (required),

gameType (required),

guessDigitCount (required)

}

Request:"POST",

Response: {

status:200 ,

data:{

type:"gameSessionCreateSuccess" ,

msg:{

code ,

link ,

sessionInfo:(updated session or created a new session)

}

}

}

9. PLAYERS JOIN GAME SESSION

This endpoint is used for inviting players, the generated invite code returned on the createGame endpoint will be sent to invited players, after which they will pass the inviteCode together with other auth arguments, in order to get the game session ID

Route: /player/multiplayer/authInvite

Body:{

jwtToken (optional) ,

inviteCode (required),

anonymous (optional)

}

For users to play anonymously , the **anonymous** argument must be set to true, after which a temp jwtToken will be generated for the anonymous user , and this token will be used for their auth throughout the gameplay

Request:"POST"

Response: {

ok:true ,

data:{

type:"gameIdQuerySuccess" ,

msg:{

gameID,

tempID (returned only when anonymous = true)

}

}

}

10. PLAYERS UPLOAD SECRET CODE

This endpoint enables players to upload their secret code to game session before the game starts

Route: /player/multiplayer/updateSecretCode

Body:{

jwtToken (required),

gameID (required),

secretCode (required)

}

Request:"POST"

Response: {

ok:true ,

data:{

type:"secretCodeUpdateSuccess" ,

msg:"secret code uploaded successfully"

}

}

11. HOST CREATE TOURNAMENT SESSION

This endpoint enables players to create and customise tournament sessions, automatically hosted on the game server.

Route: /host/tournament/create

Body:{

"jwtToken", (required = String) (Host Auth)

"name", (required = String)

"gameMode", (optional = String) (“1v1” or “gv1”)

"type", (required = String)

"entrySettings", (required = Object) (info like entry fee and the likes)

"startDate_time", (required = String) (Date set for tournament to start)

"pricePool", (required = Object) (payment for winners)

"matchTimeLimit", (required = Object)

"allowSpectators", (required = Boolean)

"autoStart", (required = Boolean)

"pricePoolType", (required = String) (Auto calculated or not)

"playersCount", (required = Number)

“tMode” (optional = String) (Defaults to singleElimination)

“guessDigitCount” (optional) (Defaults to 4)

}

Request:”POST”

Response: {

ok:true ,

data:{

type:"tournamentCreationSuccess" ,

msg:"tournament created successfully"

}

}

}

12. PLAYER JOIN TOURNAMENT SESSION

    This endpoint enables players join a tournament with regAlive = true

    Route: "/player/tournament/join"

    Body:{

        "jwtToken", (required = String)

        "tID" (required = String)

    }

    Request:"POST"

    Response: {

        ok:true ,

        data:{

            type:"joinTournamentSuccess" ,

            msg:`successfully joined ${tournamentSessionExist.name} tournament`

        }

    }

13. HOST DELETE TOURNAMENT SESSION.

    This endpoint is used by host to delete existing tournament sessions

    Route: "/host/tournament/delete"

    Body:{

        "jwtToken", (required = String)

        "tID", (required = String)

}

Request: “POST”

Response:{

ok:true ,

data:{

type:"deleteTournamentSuccess" ,

msg:`successfully deleted ${tournamentSessionExist.name} tournament`

}

}

14. GET GAME LEADERBOARD

This endpoint fetches the game leaderboard information , as well as players ranks and their XP

Route: “/game/leaderboard”

Body: {

“searchObj” (required = Object) (searchObj.mode === “global” , then global leader board, else returns leader board based on searchObj.country)

}

Request: “GET”

Response:{

ok:true ,

data:{

type:"leaderBoardFetchSuccess" ,

msg: [{username , userID , XP , country}……] (Top 10 players)

}

}

15. GET PLAYER STATS

This endpoint returns the stats of players in the game

Route: "/player/stats"

Body: {

“playerID”, (required = String)

}

Request: “GET”

Response:{

ok:true ,

data:{

type:"statsFetchSuccess" ,

msg:{

“userID”,

“username”

“dailyStreak”

“matchPlayed”

“shortestGameTime”

“rank”

“country”

“dateJoined”

“achievements”

}

}

}

WEBSOCKET ENDPOINTS

WsServerUrl: wss://doi-apis.fly.dev/

1. Multiplayer Game Route:

   This endpoint contains sub-routes for various actions during multiplayer games for both 1v1 mode and group gameplay(gv1)

   Route: parsed.type = “multiplayer”

   Sub-Route:

   1) Game connection sub-route:
   This request is sent whenever a player newly connects to a game

```
type:”multiplayer”,
post:”connection”,
body:{
        jwtToken (required),
        gameID (required)
}
```

Initiator response (Private response to person sending request)

```
Response: {
        type:”multiplayer”,
        post:”connection”,
        data:{
                type:”wsConnectionSuccess”,
                msg:{
                        generalPlayerInfo (username and avatar of game players),
                        playerPosition (player’s position in game [“player1,player2”]),
                        connectedPlayersCount (players connected),
                        gameMode (1v1 game or gv1 game)
                }

        }

}
```

General response (General response to all players EXCEPT initiator)

Response:{
type:"multiplayer",
post:" serverAuto",
data:{
type: "newGameConnection",
msg:{
generalPlayerInfo (username and avatar of game players),
}
}
}

2) Player Guess sub-route:

This request is sent whenever a player enters a new guess

type:"multiplayer",
post:" playerGuess",
body:{
jwtToken (required),
gameID (required),
guess (required),
clockTimer (required),
gameMode (required)
}

Initiator response (Private response to person sending request)

Response:{
type:"multiplayer",
post:"playerGuess",
data:{
type:"yourCalls",
msg:{
calls (opponent calls from guess e.g [2D 1I]),
dead (opponent dead points e.g 3),
injured (opponent injured points e.g 1)
}

```
        }
}
```

General response (General response to all players)

```
Response:{
        type:"multiplayer",
        post:" nextGameRound",
        data:{
                playerTimeSync (Timer of previous player)
                previousPlayer (Previous player e.g player1),
                playerTurn (player whose turn it is to play e.g player2)
        }
}
```

3) Player TimeOut sub-route:

This request is sent whenever a player runs out of time during game play

```
type:"multiplayer",
post:" playerTimeOut",
body:{
        jwtToken (required),
        gameID (required)
}
```

Initiator response (Private response to person sending request)

```
Response:{
        type:"multiplayer",
        post:" serverAuto",
        data:{
                type:"gameEnd",
                gameWinner (game winner e.g player1)
                status (status e.g winner / loser)
        }
}
```

General response (General response to all players EXCEPT initiator)

```
Response:{
        type:"multiplayer",
        post:"serverAuto",
        data:{
```

type:"gameEnd",
gameWinner (game winner e.g player1)
status (status e.g winner / loser)
}
}

4) Game Reconnection sub-route:
This request is sent whenever a player newly connects to a game

type:"multiplayer",
post:" re_connect",
body:{
jwtToken (required),
gameID (required)
}

Initiator response (Private response to person sending request)
Response: {
type:"multiplayer",
post:" re_connect",
data:{
type:" wsReconnectionSuccess",
msg:{
generalPlayerInfo (username and avatar of game players),
playerPosition (player's position in game ["player1,player2"]),
connectedPlayersCount (players connected),
playersTimer (timer record of all players for syncing),
yourCalls (Record of all your previous calls),
gameMode (gameMode of the game)
}

}

}

General response (General response to all players)
Response:{

type:"multiplayer",
post:" serverAuto",
data:{
type: "reconnectGame",
msg:{
reconnectPlayer (username and avatar of game players),
reconnectPlayerTimer (last recorded timer of the reconnected player),
playState (play to continue game),
playerTurn (players whose turn it is to play)
}
}
}

SERVER AUTO RESPONSE

This response are sent automatically when a server condition is fulfilled

Start Game:

Response:{
type:"multiplayer",
post:" serverAuto",
data:{
type: "gameStart",
msg:{
generalPlayerInfo (username and avatar of game players),
playerTurn (player whose turn it is to play :default=player1)
}
}
}

End Game:

Response:{

```
        type:"multiplayer",
        post:" serverAuto",
        data:{
                type: "gameEnd",
                msg:{
                        gameWinner (Winner of the game e.g player1),
                        status (status of player at the end of game e.g
winner or loser)
                }
        }
}
```