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"
        }
}
```