Project: TAW_ForzaCaste

API documentation related to the connected four application developed by the ForzaCaste group for the TAW project

N.B. <text in this format> represents user input

🗀 Collection: User management

End-point: User login

Login endpoint that uses passport middleware to check user credentials and if it is everything ok a new JWT will be returned

If the login is successful, it will return a response with HTTP status code 200, with this body:

```
{ error: false, errormessage: "", token: token_signed }
```

where token field contains the jwt_token.

If the login fails, it will return a response with HTTP status code 500, with this body:

```
{ statusCode: 500, error: true, errormessage: "..." }
```

where errormessage field is evaluated:

- "Invalid user" when the username does not exist
- "Invalid password" when the password is not correct

Once obtained the JWT_TOKEN it is important to connect with socket.io to be notified of changes

Method: GET

> ```
> localhost:8080/login
> ```

Headers

Content-Type	Value
Content-Type	application/x-www-form-urlencoded

Body (**raw**)

```

```

🔑 Authentication basic

Param	value	Type

- -

End-point: Getting connected user

Get user information associated with the current JWT_TOKEN, in addition refreshes the JWT_TOKEN if the expire time is less than 5 minutes

It will response with HTTP status code 200 with this body: `{ error:false, errormessage: 'Logged in user is ${req.user.username}' }`.
If the token expire time is less then 5 minutes it will response with HTTP status code 200 with this body: `{error:false,errormessage:'Logged in user is ${req.user.username}', token: 'new_token'}`

Method: GET

>```
>localhost:8080/whoami
>```

🔑 Authentication bearer

Param	value	Type
token	{{JWT_TOKEN}}	string

- -

End-point: User sign up

Creation of a new user, this endpoint reset statistics and save user information into DB.

This method responds:

- HTTP code 400 with this body `{ statusCode: 400, errormessage: "Some field missing, sign up cannot be possible" }` if all attributes have not been specified in the body
- HTTP code 400 with this body `{ statusCode: 400, errormessage: "You cannot register yourself as cpu" }` if you try to register yourself as cpu
- HTTP code 404 with this body `{ statusCode: 404, errormessage: "User already exists" }` if the username is already taken
- HTTP code 401 with this body `{ statusCode: 401, errormessage: "DB error: " + reason.errmsg }` if an error occurs in the DB

- HTTP code 200 with this body `{ error: false, errormessage: "", id: data._id }` if everything ok

Below is an example of a body for this request, all specified fields are mandatory

Method: POST

```
localhost:8080/users
```

Body (**raw**)

```
{
    "username": "<username chosen by the user>",
    "name": "<name>",
    "surname": "<surname>",
    "avatarImgURL": "<user image URL>",
    "mail": "<user mail>",
    "password": "<user password>"
}
```

🔑 Authentication noauth

Param	value	Type

- -

End-point: Adding a new moderator

Creation of a new moderator, this request can be made only by another moderator.

When the moderator is created he is in the status of unregistered moderator and cannot make any request other than updating his profile to add his personal data.

This method responds:

- HTTP status code 400 with this body `{ statusCode: 400, errormessage: "Some field missing, signup cannot be possible" }` if all attributes have not been specified in the body
- HTTP status code 400 with this body `{ statusCode: 400, errormessage: "User already exists" }` if the username specified already exists
- HTTP status code 401 with this body `{ statusCode: 401, errormessage: "DB error: " + reason.errmsg }` if an error occurs in the DB
- HTTP status code 401 with this body `{ statusCode: 401, errormessage: "Operation not permitted" }` if the user who made the request does not have permission to do it
- HTTP status code 200 with this body `{ error: false, errormessage: "", id: modCreated._id }` if everything ok

Below is an example of a body for this request, all specified fields are mandatory

Method: POST

```
localhost:8080/users/mod
```

Headers

Content-Type	Value
Content-Type	application/json

Body (**raw**)

```
{
    "username" : "<username>",
    "password" : "<password>"
}
```

Authentication bearer

Param	value	Type
token	{{JWT_TOKEN}}	string

- -

End-point: Update user information

This request allows the user to modify his profile. This is also used by moderators who must update the profile before they can browse the API.

Not registered moderator after this request becomes moderator.

This method responds:

- HTTP status code 401 with this body `{ statusCode: 401, errormessage: 'Wrong password, you aren't allowed to do it' }` if the old password does not match
- HTTP status code 400 with this body `{ statusCode: 400, errormessage: "Old password is missing" }` if the user has not specified the old password
- HTTP status code 400 with this body `{ statusCode: 400, errormessage: "Some field are missing" }` if the user is a not registered moderator who did not specify all fields
- HTTP status code 401 with this body `{ statusCode: 401, errormessage: 'DB error: ${err}'}` if an error occurs in the DB
- HTTP status code 200 with this body `{ error: false, errormessage: "" }` if everything ok

If the client is listening on the event *updateUser* it will receive the user object just saved into DB

Below is an example of a body for this request

Not registered moderator must specify the following fields:

- name
- surname
- mail
- avatarImgURL
- password

Other users can specify only the fields they want to modify, the only constraint is that if they want to change the password then they must also specify the old one.

Method: PUT

> ```
> localhost:8080/users
> ```

Body (**raw**)

```
{
    "name": "<name>",
    "surname": "<surname>",
    "avatarImgURL": "<user image URL>",
    "password": "<new password>",
    "old password": "<old password>",
    "mail" : "<user mail>"
}
```

🔑 Authentication bearer

Param	value	Type
token	{{JWT_TOKEN}}	string

- -

End-point: Deletion of standard players

This request allows the moderator to logically delete a standard player specified within param. It also removes his friendships.

This method responds:

- HTTP status code 401 with this body `{ statusCode: 401, errormessage: "You cannot delete a mod" }` if the user you want to delete is a moderator
- HTTP status code 401 with this body `{ statusCode: 401, errormessage: 'DB error: ${err}'}` if an error occurs in the DB
- HTTP status code 401 with this body `{ statusCode: 401, errormessage: "You cannot do it, you aren't a mod!" }` if the user who make the request is not a moderator

- HTTP status code 200 with this body `{ error: false, errormessage: "" }` if everything ok

Method: DELETE

```
localhost:8080/users/<:username>
```

Body (**raw**)

```

```

🔑 Authentication bearer

Param	value	Type
token	{{JWT_TOKEN}}	string

- -

End-point: Getting user information

This request allows the user to obtain information about another user by specifying the username.

This method responds:

- HTTP status code 401 with this body `{ statusCode: 401, errormessage: 'DB error: ${err}'}` if an error occurs in the DB
- HTTP status code 200 with this body

  ```
  {   "username": "username",   "name": "name",   "surname": "surname",
  "avatarImgURL": "user image URL",   "mail": "user mail",   "statistics": {
  "nGamesWon": 1,       "nGamesLost": 21,       "nGamesPlayed": 42,
  "nTotalMoves": 292,       "ranking": 125,       "_id": "statistics id"   },
  "friendList": [       {           "_id": "friend 1 id",
  "username": "friend 1 username",           "isBlocked": false       },
  {           "_id": "friend 2 id",           "username": "friend 2 username",
  "isBlocked": true       }   ],   "role": "User role"}
  ```

Method: GET

```
localhost:8080/users/<:username>
```

Body (**raw**)

```

```

🔑 Authentication bearer

Param	value	Type
token	{{JWT_TOKEN}}	string

End-point: Getting list of all users

This request allows the user to get all the useful information about the players who are registered in the platform.

This method responds:

- HTTP status code 401 with this body `{ statusCode: 401, errormessage: 'DB error: ${err}'}` if an error occurs in the DB
- HTTP status code 401 with this body `{ statusCode: 401, errormessage: "You cannot get user list" }` if you are a non registered moderator
- HTTP status code 200 with this body

```
{
  "error": false,
  "errormessage": "",
  "userlist": [
    {
      "_id": "user 1 id",
      "roles": "user 1 role",
      "username": "username 1",
      "name": "name 1",
      "surname": "surname 1"
    },
    {
      "_id": "user 2 id",
      "roles": "user 2 role",
      "username": "username 2",
      "name": "name 2",
      "surname": "surname 2"
    },
    {
      "_id": "user 3 id",
      "roles": "user 3 role",
      "username": "username 3",
      "name": "name 3",
      "surname": "surname 3"
    },
    ...
    ]
}
```

Method: GET

> ```
> localhost:8080/users
> ```

🔑 Authentication bearer

Param	value	Type
token	{{JWT_TOKEN}}	string

End-point: Getting list of online players

This request allows you to get all online users with a web socket (specifically socket.io) registered in the application at the moment of the request.

This method responds:

- HTTP status code 401 with this body `{ statusCode: 401, errormessage: 'You cannot do it' }` if the user who requested this resource is not authorized to receive it
- HTTP status code 401 with this body `{ statusCode: 401, errormessage: 'DB error: ${err}'}` if an error occurs in the DB
- HTTP status code 200 with this body

```
{
    "error": false,
    "errormessage": "",
    "onlineuser": [
        "player1",
        "player2"
    ]
}
```

Method: GET

> ```
> localhost:8080/users/online
> ```

🔑 Authentication bearer

Param	value	Type
token	{{JWT_TOKEN}}	string

- -

End-point: Getting ranking history of logged user

This request allows the logged in user to get the history of his ranking, it is based on the ranking he had at the time of the game requests he made

This method responds:

- HTTP status code 401 with this body `{ statusCode: 401, errormessage:'You cannot do it' }` if the user logged in is a non registered moderator
- HTTP status code 401 with this body `{ statusCode: 401, errormessage: 'DB error: ${err}'}` if an error occurs in the DB
- HTTP status code 200 with this body

```
{
  "error": false,
  "errormessage": "",
  "matchmakingList": [
      {
          "_id": "ranking id",
          "ranking": 0
      },
      {
          "_id": "ranking id",
          "ranking": 30
      },
      {
          "_id": "ranking id",
          "ranking": 70
      },
      {
          "_id": "ranking id",
          "ranking": 60
      }
  ]
}
```

Method: GET

> ```
> localhost:8080/rankingstory
> ```

🔑 Authentication bearer

Param	value	Type
token	{{JWT_TOKEN}}	string

End-point: Getting ranking history of the specified user

This request allows the logged in user to get the ranking history of the user specified in the request parameters with the username, it is based on the ranking he had at the time of the game requests he made

This method responds:

- HTTP status code 401 with this body `{ statusCode: 401, errormessage:'You cannot do it' }` if the user logged in is a non registered moderator
- HTTP status code 401 with this body `{ statusCode: 401, errormessage: 'DB error: ${err}'}` if an error occurs in the DB
- HTTP status code 200 with this body

```
{
  "error": false,
  "errormessage": "",
  "matchmakingList": [
      {
          "_id": "ranking id",
          "ranking": 0
      },
      {
          "_id": "ranking id",
          "ranking": 60
      },
      {
          "_id": "ranking id",
          "ranking": 120
      },
      {
          "_id": "ranking id",
          "ranking": 100
      },
      {
          "_id": "ranking id",
          "ranking": 80
      },
      {
          "_id": "ranking id",
          "ranking": 110
      },
      {
          "_id": "ranking id",
          "ranking": 125
      }
  ]
  }
```

Method: GET

> ```
> localhost:8080/rankingstory/<:username>
> ```

Authentication bearer

Param	value	Type
token	{{JWT_TOKEN}}	string

Collection: Game

End-point: Play game against random user

Description

This endpoint allows the user to send a request to play a game against a random user.

If there isn't another request created before, a new will be created and the user will be put on hold until another user send a request of random game.

Instead if a request already exist, the matchmaking is done and both the players will be redirect to the match page. The player1 or the player that will play the first move is randomly determined and are also created the Socket.IO rooms used by the server to notify to the clients all events.

The random matchmaking works that if there are several users waiting to play, the users matching is done based on their ranking: similarly ranked users are matched. However if the players that want to play are only two and their rank is very different, there is a mechanism for matching the two players anyway.

Available listener

- ***gameReady*** event, user receive `{ 'gameReady': true, 'opponentPlayer': ""}` to inform both clients that game requests have been accepted and the match will start soon
- ***move*** event, player receive `{ 'yourTurn': bool }` that inform the user if it's his turn (`true`) or if it's opponent turn (`false`)
- ***gameStatus*** event, received by the observer of the match, that receive `{playerTurn: ""}` that informs the observators which player will do the first move

Possible responses

This method respond:

- HTTP code 404 with this body `{statusCode: 404, errormessage: "Match creation error"}` if an error occurs during the creation of the match document
- HTTP code 404 with this body `{ statusCode: 404, errormessage: "Match request update error" }` if an error occurs during the update of the notification document
- HTTP code 404 with this body `{ statusCode: 404, errormessage: "Match request already exists" }` if already exists a request for a match against a random user from the same user
- HTTP code 404 with this body `{ statusCode: 404, errormessage: "Random game request creation error" }` if an error occurs during the creation of the notification document
- HTTP code 400 with this body `{ statusCode: 400, errormessage: "Invalid request" }` if the value of he field `"type"` is incorrect
- HTTP code 400 with this body `{statusCode: 400, errormessage: "Bad Request"}` if `type` is not included in the request body
- HTTP code 404 with this body `{statusCode: 404, errormessage: "DB error"}` if an operation on the DB raises an error
- HTTP code 401 with this body `{ statusCode: 401, errormessage: "Unauthorized" }` if the user that made the request is not authorized for that operation
- HTTP code 200 with this body `{ error: false, message: "Match request has been created correctely" }` if the match has been created correctely. It means that the game document has been created and the notification document for the game request has been updated.
- HTTP code 200 with this body `{ error: false, errormessage: "Match has been created correctely" }` if the match have been created so the players will play the match soon

Below is an example of the body for this request, the specified field and the value are mandatory

Method: POST

```
localhost:8080/game
```

Body (**raw**)

```
{
    "type": "randomMatchmaking"
}
```

Authentication bearer

Param	value	Type
token	{{JWT_TOKEN}}	string

End-point: Create friendly game request

Descrption

This endpoint allows the user to send a request to a friend to play a game against him. If the friend is not online then will be raised an error, otherwise if the friend is online the user will be put on hold for the matchmaking.

Available listener

- ***gameRequest*** event, the friend who receive the request, receive `{type: "friendlyGame", player: ""}` where username contains the username of the user that send the request

Possible responses

This method responds:

- HTTP code 404 with this body `{ statusCode: 404, errormessage: "The opposite player is not a friend" }` if the user with the username specified in the body is not a friend
- HTTP code 404 with this body `{ statusCode: 404, errormessage: "The opposite player is not online" }` if the friend specified in the body by the username is not online at the moment
- HTTP code 404 with this body `{ statusCode: 404, errormessage: "Friendly game request creation error" }` if an error occurs during the creation of the notification document
- HTTP code 400 with this body `{ error: true, errormessage: "Invalid request" }` if the value of he field `"type"` is incorrect
- HTTP code 400 with this body `{statusCode: 400, errormessage: "Bad Request"}` if `type` is not included in the request body
- HTTP code 404 with this body `{statusCode: 404, errormessage: "DB error"}` if an operation on the DB raise an error
- HTTP code 401 with this body `{ statusCode: 401, errormessage: "Unauthorized" }` if the user that made the request is not authorized for that operation
- HTTP code 404 with this body `{ statusCode: 404, errormessage: {statusCode:404, errormessage: "Friendly game request already exist"}` if a game request to that friend from the same user already exists
- HTTP code 200 with this body `{ error: false, message: "Request sended to friend" }` if everything is ok

Below is an example of the body for this request, all specified fields are mandatory

Method: POST

```
localhost:8080/game
```

Body (**raw**)

```
{
    "type" : "friendlyMatchmaking",
    "oppositePlayer" : "<username of the friend>"
}
```

🔑 Authentication bearer

Param	value	Type
token	{{JWT_TOKEN}}	string

- -

End-point: Enter in a game as observator

Description

This endpoint allows the user to enter in a game as observator. The request works that the client speciies the username of one of the two player in the body. This because a user can play only one match at the moment, so it is possible to retrieve the current match that the specified player is playing.

The Socket.IO socket of the new observator join the match rooms, where will receive all the notification, such as the performed move or the game chra messagges.

Available listeners

- ***enterGameWatchMode*** event, the observator user receive `{ 'playerTurn': "", playground: "" }` that informs the observator of the status of the game, in particular the username of the player that will play the next move and the status of the playground at the moment

Possible responses

This method responds:

- HTTP code 404 with this body `{ statusCode: 404, errormessage: "SocketIO client is not connected" }` if the Socket.IO client socket is not connected to the server
- HTTP code 200 with this body `{ error: false, message: "Match joined as observator" }` if everything is ok
- HTTP code 404 with this body `{ statusCode: 404, errormessage: "The specified match does not exist" }` if there is no match with the specified user as player
- HTTP code 400 with this body `{ statusCode: 400, errormessage: "Invalid request" }` if the value of he field `"type"` is incorrect
- HTTP code 400 with this body `{statusCode: 400, errormessage: "Bad Request"}` if type is not included in the request body
- HTTP code 404 with this body `{statusCode: 404, errormessage: "DB error"}` if an operation on the DB raise an error

- HTTP code 401 with this body `{ statusCode: 401, errormessage: "Unauthorized" }` if the user that made the request is not authorized for that operation

Below is an example of the body for this request, all specified fields are mandatory

Method: POST

```
localhost:8080/game
```

Body (**raw**)

```
{
    "type" : "watchGame",
    "player" : "<username>"
}
```

🔑 Authentication bearer

Param	value	Type
token	{{JWT_TOKEN}}	string

- -

End-point: Respond to friendly game request

Descrption

This endpoint allows the user to accept or refuse a request of a friend to play a game against him.

If the `"accept"` field is true, the match request is accepted, otherwise if it is false is refued. It he first case the two players will redirected to the game view, where they will play to the match. Furthermore the match rooms will be created and the players will be added. Instead in the second case, so if the user refuse the match request, the sender will notified, using Socket.IO, that the game request has been refused.

Available listener

- ***gameReady*** event, user receive `{ 'gameReady': true, 'opponentPlayer': ""}` to inform both clients that game request have been accepted correctely and the match will start soon
- ***move*** event, player receive `{ 'yourTurn': bool }` that inform the user if it's his turn (`true`) or if it's opponent turn (`false`)
- ***gameStatus*** event, received by the observer of the match, that receive `{'playerTurn': ""}` that inform the observators which player will do the first move

Possible responses

This method respond:

- HTTP code 404 with this body `{statusCode: 404, errormessage: "Match creation error"}` if an error occurs during the creation of the match document
- HTTP code 404 with this body `{ statusCode: 404, errormessage: "Match request update error" }` if an error occurs during the update of the notification document
- HTTP code 404 with this body `{ statusCode: 404, errormessage: "Match request does not exists" }` if there is no game request by the user specified in the body
- HTTP code 200 with this body `{ error: false, errormessage: "Match has been created correctely" }` if the `"accept"` field is `true` so the match have been created correctely and the players will start the match soon
- HTTP code 200 with this body `{ error: false, message: "Request refused" }` if the `"accept"` field is `false` so the request is refused
- HTTP code 400 with this body `{statusCode: 400, errormessage: "Bad Request"}` if `accept` is not included in the request body
- HTTP code 404 with this body `{statusCode: 404, errormessage: "DB error"}` if an operation on the DB raise an error
- HTTP code 401 with this body `{ statusCode: 401, errormessage: "Unauthorized" }` if the user that made the request is not authorized for that operation
- HTTP code 404 with this body `{ statusCode: 404, errormessage: "Match request update error" }` if an error occurs while updating the math request

Below is an example of the body for this request, all specified fields are mandatory

Method: PUT

```
localhost:8080/game
```

Body (**raw**)

```
{
    "sender" : "<username of the friend who sent the request>",
    "accept": bool
}
```

🔑 Authentication bearer

Param	value	Type
token	{{JWT_TOKEN}}	string

- -

End-point: Delete a match if the player is playing it, otherwise is deleted the game request

Description

This endpoint allows the user to quit and delete a match he is playing. Otherwise, if he is not playing any match, it deletes the notification for a match against a random player or a friend. If not even any match request exists, an error will be reported.

Available listeners

- ***result*** event, the opponent player and the observators receive `{'winner': "", 'message': "Opposite player have left the game"}` that informs the player have left the game

Possible responses

This method responds:

- HTTP code 404 with this body `{ statusCode: 404, errormessage: "Match cancellation error" }` if an error occurs while deleting a match document from the DB
- HTTP code 200 with this body `{ error: false, message: "The match has been deleted correctely" }` if the match have been deleted correctely from the DB
- HTTP code 404 with this body `{ statusCode: 404, errormessage: "Notification cancellation error" }` if an error occurs while deleting the notification document from the DB
- HTTP code 200 with this body `{ error: false, message: "The match request has been deleted correctely" }` if the notification have been deleted correctely from the DB
- HTTP code 404 with this body `{ statusCode: 404, errormessage: "Match request does not exists" }` if a match request sent by this user doesn't exist
- HTTP code 404 with this body `{statusCode: 404, errormessage: "DB error"}` if an operation on the DB raise an error
- HTTP code 401 with this body `{ statusCode: 401, errormessage: "Unauthorized" }` if the user that made the request is not authorized for that operation

This request have no body because can be only one match or match request active at any moment

Method: DELETE

```
localhost:8080/game
```

🔑 Authentication bearer

Param	value	Type
token	{{JWT_TOKEN}}	string

- -

End-point: Get all in progress matches

Description

This endpoint allows the client to retrieve all the in progress matches.

Possible responses

This method responds:

- HTTP code 200 with this body `{ error: false, matches: matches }` where `matches` is an array of matches
- HTTP code 404 with this body `{ statusCode: 404, errormessage: "Error getting matches" }` if an error occurs in the query to get the in progress matches from the DB
- HTTP code 404 with this body `{statusCode: 404, errormessage: "DB error"}` if an operation on the DB raise an error
- HTTP code 401 with this body `{ statusCode: 401, errormessage: "Unauthorized" }` if the user that made the request is not authorized for that operation

```
{
    "error": false,
    "matches": [
        {
            "playground": [
                [
                    "/",
                    "/",
                    "/",
                    "/",
                    "/",
                    "/",
                    "/"
                ],
                [
                    "/",
                    "/",
                    "/",
                    "/",
                    "/",
                    "/",
                    "/"
                ],
                [
                    "/",
                    "/",
                    "/",
                    "/",
                    "/",
                    "/",
                    "/"
                ],
                [
                    "/",
                        "/",
                        "/",
                        "/",
                        "/",
                        "/",
                        "/"
                    ],
                    [
                        "/",
                        "/",
                        "/",
                        "/",
                        "/",
                        "/",
                        "/"
                    ],
                    [
                        "/",
                        "/",
                        "/",
                        "/",
                        "/",
                        "/",
                        "/"
                    ]
                ],
                "_id": "61eac3f7f83ae200290de9a4",
                "inProgress": true,
                "player1": "",
                "player2": "",
                "winner": null,
                "chat": [],
                "nTurns": 1,
                "__v": 0
            }

        ]
    }
```

Method: GET

```
localhost:8080/game
```

Authentication bearer

Param	value	Type
token	{{JWT_TOKEN}}	string

End-point: Create a game against CPU

Description

This endpoint allows the user to play a game against the CPU.

Possible responses

This method respons:

- HTTP code 403 with this body `{ statusCode: 401, errormessage: "Unauthorized" }` the user that send the HTTP request haven't necessary roles
- HTTP code 404 with this body `{ statusCode: 404, errormessage: "Socket.IO error" }` if occurs an error while joining the match room
- HTTP code 200 with this body `{ error: false, errormessage: "Single player match has been created" }` if everything is ok
- HTTP code 404 with this body `{ statusCode: 404, errormessage: "Match saving error" }` if an error occurs while saving the match into the DB
- HTTP code 404 with this body `{statusCode: 404, errormessage: "DB error"}` if an operation on the DB raise an error

The body for this request is empty

Method: POST

>```
>localhost:8080/game/cpu
>```

🔑 Authentication bearer

Param	value	Type
token	{{JWT_TOKEN}}	string

End-point: Ask AI what is the best move

This request allows the logged-in user who is playing a game to ask an artificial intelligence what is the best move to play.
The AI will return the number of the column in which it thinks to make the move with also a number that is the weight that move has on the game, the higher the number the greater the probability of winning.

N.B. Only user with ranking greater that 100 can ask for help

This method responds:

- HTTP code 403 with this body `{statusCode: 403, errormessage:"You cannot do it" }` if the user is a non registered moderator
- HTTP code 403 with this body `{statusCode: 403, errormessage:"You have to improve your ranking before doing that!" }` if the user has a ranking lower than 100
- HTTP code 404 with this body `{statusCode: 404, errormessage: "Match not found" }` if the user is not playing a match
- HTTP code 401 with this body `{ statusCode: 401, errormessage: "DB error: " + reason.errmsg }` if an error occurs in the DB
- HTTP code 200 with this body `{ error: false, errormessage: "", move: {"0": 4, "1": 9} }` where `move[0]` represent the suggested column, while `move[1]` represent the weight that this move has on the game

Method: GET

```
localhost:8080/move
```

🔑 Authentication bearer

Param	value	Type
token	{{JWT_TOKEN}}	string

- -

End-point: Play the turn making a move

Description

This method allows a user who is logged in and playing a game against another person to play their turn, by inserting the disk in the desired column.
This method handles all the game logic, checking if a move is invalid or one of the two players wins the game, in this case it takes care of updating the statistics by assigning a score, which will be added to the ranking, based on the "skill" of the player.
It also takes care of notifying the listeners of any event that is happening during the game.

Available listener

If the client is listening on:

- ***move*** event, player receive `{ "error": true, "codeError": 3, "errorMessage": "Wrong turn" }` if the player is trying to play even if it is not his turn
- ***move*** event, player receive `{ "error": false, "codeError": null, "errorMessage": null }` if everything ok
- ***move*** event, player receive `{ "error": true, "codeError": 1, "errorMessage": "The column is full" }` or `{ "error": true, "codeError": 2, "errorMessage": "Move not allowed, out of playground" }` if the player is trying to insert an invalid move

- ***move*** event, opponent player receive `{ move: index }` which represents the move played by other player in which *index* is the chosen column
- ***gameStatus*** event, match observers, who are joining the match room identified by the string made up of player1's username concatenated to the string Watchers*(i.e. username1Watchers),* receive `{ player: username, move: index, nextTurn: otherPlayer }` which represents the player who inserted disk into column index and the player who has to play next turn
- ***result*** event, players receive `{ "winner": null }` if the game ended up in a draw
- ***result*** event, the winner receive `{ winner: true }`
- ***result*** event, the loser receive `{winner:false}`
- ***result*** event, players receive `{ "rank": rank }` representing the points they have lost or gained after this match
- ***result*** event, match observers, who are joining the match room, receive `{ winner: 'username of the player who won'}`

Possibile responses

This method responds:

- HTTP status code 400 with this body `{ statusCode: 400, errormessage: "Bad request, you should pass your move" }` if the request is uncorrecly formed
- HTTP status code 400 with this body `{ statusCode: 400, errormessage: "Wrong turn" }` if the player is trying to play even if it is not his turn
- HTTP status code 404 with this body `{ statusCode: 404, errormessage: "Match does not exists" }` if the user is not playing a match
- HTTP status code 403 with this body `{ statusCode: 403, errormessage: "You cannot do it" }` if the user is a non registered moderator
- HTTP status code 400 with this body `{ statusCode: 400, errormessage: "This column is full, choose another one" }` or `{ statusCode: 400, errormessage: "Move not allowed, out of playground, choose another one" }` if the player is trying to insert an invalid move
- HTTP status code 200 with this body `{ error: false, errormessage: "added move" }`

Below is an example of the body for this request, all specified fields are mandatory

Method: POST

```
localhost:8080/move
```

Body (**raw**)

```
{
    "move": 1
}
```

Authentication bearer

Param	value	Type
token	{{JWT_TOKEN}}	string

- -

End-point: Play the turn vs AI making a move

Description

This method, is very similar to *Play the turn making a move*, in fact it allows a user who is logged in and playing a game against a CPU to play their turn, by inserting the disk in the desired column.
This method handles all the game logic, checking if a move is invalid or one of the two players wins the game, in this case the statistics are not updated.
It also takes care of notifying the listeners of any event that is happening during the game.

The AI logic is implemented using Minimax algorithm.

Available listener

If the client is listening on:

- ***gameStatus*** event, match observers, who are joining the match room identified by the string made up of player's username concatenated to the string Watchers*(i.e. usernameWatchers),* receive `{ player:'player username or cpu', move: index, nextTurn: 'player username or cpu' }` which represents the player who inserted disk into column index and the player who has to play next turn
- ***result*** event, players receive `{ "winner": null }` if the game ended up in a draw
- ***result*** event, the winner receive `{ winner: true }`
- ***result*** event, the loser receive `{winner:false}`
- ***result*** event, players receive `{ "rank": rank }` representing the points they have lost or gained after this match
- ***result*** event, match observers, who are joining the match room, receive `{ winner: 'username of the player who won'}`

Possibile responses

This method responds:

- HTTP status code 401 with this body `{ statusCode: 401, errormessage: "Bad request, you should pass your move" }` if the request is uncorrecly formed
- HTTP status code 400 with this body `{ statusCode: 400, errormessage: "Wrong turn" }` if the player is trying to play even if it is not his turn
- HTTP status code 404 with this body `{ statusCode: 404, errormessage: "Match does not exists" }` if the user is not playing a match
- HTTP status code 403 with this body `{ statusCode: 403, errormessage: "You cannot do it" }` if the user is a non registered moderator

- HTTP status code 400 with this body `{ statusCode: 400, errormessage: "This column is full, choose another one" }` or `{ statusCode: 400, errormessage: "Move not allowed, out of playground, choose another one" }` if the player is trying to insert an invalid move
- HTTP status code 200 with this body:
  - `{ error: false, errormessage: "The match ended up in a draw" }` if the game ended up in a draw
  - `{ error: false, errormessage: "Correctly added move", cpu: index }` which represents that the game continues and informs the player about the move made by the AI
  - `{ error: false, errormessage: "Correctly added move", cpu: index, winner: username }`If the game is over, winner contains the player who wins

Below is an example of the body for this request, all specified fields are mandatory

Method: POST

```
localhost:8080/move/cpu
```

Body (**raw**)

```
{
    "move":0
}
```

Authentication bearer

Param	value	Type
token	{{JWT_TOKEN}}	string

- -

End-point: Send a message into the game chat

Description

This endpoint allows the user, that have joined a match as observator, to send a message in the chat to al other observator.

The match chat works that if the sender of a message is a player or a modertor, the message will be received by everybody, otherwise if the message is sent by an observator it will be received only by the others observators and moderators.

Available listeners

- ***gameChat*** event, all the users involved in the match, so the players and the observators, receive `{'_id' : "", 'content' : "", 'sender' : "", 'receiver' : "", 'timestamp' : {"$date" : ""}}` that correspond to the match document in the DB.

Possible responses

This method responds:

- HTTP code 404 with this body `{ statusCode: 404, errormessage: "SocketIO client is not connected" }` if the Socket.IO client socket is not connected to the server
- HTTP code 200 with this body `{ error: false, message: "Message have been send and saved correctely" }` if everything is ok
- HTTP code 404 with this body `{ statusCode: 404, errormessage: "Send message error" }` if an error occurs while saving the message into the DB
- HTTP code 404 with this body `{ statusCode: 404, errormessage: "SocketIO client is not in the match room" }` if the user that send the HTTP request is not an observator
- HTTP code 404 with this body `{ statusCode: 404, errormessage: "Match not found" }` if the match with the player specified in the body does not exist
- HTTP code 400 with this body `{statusCode: 400, errormessage: "Bad Request"}` if `player` username or `message` fields are not included in the request body
- HTTP code 404 with this body `{statusCode: 404, errormessage: "DB error"}` if an operation on the DB raise an error
- HTTP code 401 with this body `{ statusCode: 401, errormessage: "Unauthorized" }` if the user that made the request is not authorized for that operation

Below is an example of the body for this request, all specified fields are mandatory

Method: POST

```
localhost:8080/gameMessage
```

Body (**raw**)

```
{
    "player" : "<username>",
    "message" : "<message>"
}
```

🔑 Authentication bearer

Param	value	Type
token	{{JWT_TOKEN}}	string

Collection: Notification management

End-point: Send a friend request

Description

This endpoint allows the user to send a friend request to another user, who is not already friend.

Available listeners

- ***newNotification*** event, the recipient of the request receive `{ sender: "", type: "friendRequest" }` that inform the destination user of a new friend request

Possible responses

The method respond:

- HTTP code 404 with this body `{ statusCode: 404, errormessage: "You are already friend" }` if the the two users are already friends
- HTTP code 404 with this body `{ statusCode: 404, errormessage: "Request already exist" }` if a friend request for the specified user already exists
- HTTP code 200 with this body `{ error: false, message: "Request forwarded" }` if everything is ok
- HTTP code 404 with this body `{ statusCode: 404, errormessage: "Notification creation error"}` if an error occurs while creating the notification into the DB
- HTTP code 404 with this body `{ statusCode: 404, errormessage: "Specified user does not exist" }` if the user specified in the body does not exist
- HTTP code 404 with this body `{ statusCode: 404, errormessage: "Type of the notification not accepted" }` if the value of the filed `"type"` is not valid. In this case the only valid type is "friendRequest"
- HTTP code 400 with this body `{statusCode: 400, errormessage: "Bad Request"}` if type is not included in the request body
- HTTP code 404 with this body `{statusCode: 404, errormessage: "DB error"}` if an operation on the DB raise an error
- HTTP code 401 with this body `{ statusCode: 401, errormessage: "Unauthorized" }` if the user that made the request is not authorized for that operation

Below is an example of the body for this request, all specified fields are mandatory

Method: POST

>```
>localhost:8080/notification
>```

Body (**raw**)

```
{
    "receiver": "<username of user to send the request>",
    "type": "friendRequest"
}
```

Authentication bearer

Param	value	Type
token	{{JWT_TOKEN}}	string

End-point: Get all friends of an user

Description

This endpoint allows the client to ask the server all the inbox notification of the user.

Possible responses

This method respond:

- HTTP code 404 with this body `{statusCode: 404, errormessage: "DB error"}` if an operation on the DB raise an error
- HTTP code 401 with this body `{ statusCode: 401, errormessage: "Unauthorized" }` if the user that made the request is not authorized for that operation
- HTTP code 200 with this body `{ error: false, notification: n }` where notification contains all the notification, so friend request and match request, received by the user

```
{
    "error": false,
    "errormessage": "",
    "friendlist": [
        {
            "_id": "61cb32503f80cd00c65aa19b",
            "username": "",
            "isBlocked": false
        }
    ]
}
```

Method: GET

```
localhost:8080/friend
```

```
```

Body (**raw**)

```
```

Authentication bearer

Param	value	Type
token	{{JWT_TOKEN}}	string

End-point: Answer to a friend request

Description

This endpoint allows a user to accept or refuse a friend request. In particular this request accept if the `accepted` field is `true`, otherwise it refuse, a friend request sent by the user specified in the `sender` field.

Available listeners

- ***request*** event, the creator of the request receive `{newFriend: ""}` that inform that the other user has accepted the friend request. If the `newFriend` field is `null` it means that the friend request has been refused

Possible responses

This method respond:

- HTTP code 404 with this body `{ statusCode: 404, errormessage: "Notification does not exist" }` if the notification that you want to accept or refuse does not exist
- HTTP code 404 with this body `{ statusCode: 404, errormessage: "Friend addition error" }` if an error occur adding the user to the friend list of the other user in the DB
- HTTP code 200 with this body `{ error: false, errormessage: "Data saved successfully" }` if everything is ok
- HTTP code 404 with this body `{ statusCode: 404, errormessage: "Notification update error" }` if an error occurs while updating the notification in the DB
- HTTP code 400 with this body `{statusCode: 400, errormessage: "Bad Request"}` if `accpeted` field is not included in the request body
- HTTP code 404 with this body `{statusCode: 404, errormessage: "DB error"}` if an operation on the DB raise an error
- HTTP code 401 with this body `{ statusCode: 401, errormessage: "Unauthorized" }` if the user that made the request or the user that received the request are not authorized for that operation

Below is an example of the body for this request, all specified fields are mandatory

Method: PUT

```
localhost:8080/notification
```

Body (**raw**)

```
{
    "sender": "<username of the user that sent the notification>",
    "accepted": bool
}
```

Authentication bearer

Param	value	Type
token	{{JWT_TOKEN}}	string

- -

End-point: Delete a friend

Description

This endpoint allows the user to remove a friend from his friend list.

The user to be removed must be specified in the URL as parameter using the username.

Available listeners

- ***friendDeleted*** event, the friend of the user receive `{deletedFriend: ""}` that inform the user that he has lost a friend

Possible response

The method respond:

- HTTP code 200 with this body `{ error: false, errormessage: "Friend removed" }` if the friend is deleted from the user friend list correctely
- HTTP code 404 with this body `{statusCode: 404, errormessage: "The user is not friend"}` if the specified user is not a friend
- HTTP code 400 with this body `{statusCode: 400, errormessage: "Bad Request"}` if the `username` parameter is not included in the request body
- HTTP code 404 with this body `{statusCode: 404, errormessage: "DB error"}` if an operation on the DB raise an error

- HTTP code 401 with this body `{ statusCode: 401, errormessage: "Unauthorized" }` if the user that made the request is not authorized for that operation

Method: DELETE

```
localhost:8080/friend/<username>
```

Authentication bearer

Param	value	Type
token	{{JWT_TOKEN}}	string

End-point: Block or unblock a friend

Description

This endpoint allows the user to block or unblock a friend (a user in his friend list).

If the isBlocked field is `true` the friend will be blocked, otherwise if it is `false`, it will be unblocked.

The friend on which the operation is performed, is specified using the `username` field in the body.

Available listeners

- ***friendBlocked*** event, the friend that has been blocked or unblocked receive `{blocked: bool}` that inform if he has been blocked if it is true, otherwise if it is false it means he has been unblocked

Possible responses

This method respond:

- HTTP code 200 with this body `{ error: false, errormessage: "User blocked" }` if the friend has been blocked correctely
- HTTP code 200 with this body `{ error: false, errormessage: "User unblocked" }` if the friend has been unblocked correctely
- HTTP code 400 with this body `{statusCode: 400, errormessage: "Bad Request"}` if both `username` and `isBlocked` fields are not included in the body of the request
- HTTP code 404 with this body `{statusCode: 404, errormessage: "DB error"}` if an operation on the DB raise an error
- HTTP code 401 with this body `{ statusCode: 401, errormessage: "Unauthorized" }` if the user that made the request is not authorized for that operation

Below is an example of the body for this request, all specified fields are mandatory

Method: PUT

```
localhost:8080/friend
```

Body (**raw**)

```
{
    "username": "<username of the user to block or unblock>",
    "isBlocked": bool
}
```

🔑 Authentication bearer

Param	value	Type
token	{{JWT_TOKEN}}	string

End-point: Get all user notifications

This method allows the user to get all the notifications they have received.
In addition, two parameters are available, one for filtering which returns only unread messages and the other for making these notifications read.

N.B. The two parameters can be concatenated to take advantage of both features

This method responds:

- HTTP status code 404 with this body `{ statusCode: 404, errormessage: "DB error: " + reason }` if an operation on the DB raise an error
- HTTP status code 200 with this body(*request forwarded without params*)

```
{
    "error": false,
    "errormessage": "",
    "notification": [
        {
            "inpending": true,
            "_id": "61d564dac6c39d00633e4243",
            "type": "friendRequest",
            "text": "New friend request by user1.",
            "sender": "user1",
            "receiver": "currentUser",
            "deleted": false,
            "__v": 0
        },
        {
            "inpending": false,
            "_id": "61d564dac6c39d00633e4243",
            "type": "friendRequest",
            "text": "New friend request by user2.",
            "sender": "user2",
            "receiver": "currentUser",
            "deleted": true,
            "__v": 0
        }
    ]
}
```

Method: GET

> ```
> localhost:8080/notification
> ```

Query Params

Param	value
inpending	true
makeNotificationRead	true

Authentication bearer

Param	value	Type
token	{{JWT_TOKEN}}	string

Collection: Message management

End-point: Send a message to a user

Description

This endpoint allows the user to send a message to a friend. The payload of the message and the destination user must be specified in the body.

Available listeners

- ***message*** event, the target user of the message receive `{'_id' : "", 'content' : "", 'sender' : "", 'receiver' : "", 'timestamp' : {"$date" : ""}}` that correspond to the match document in the DB.

Possible responses

This method responds:

- HTTP code 404 with this body `{ statusCode: 404, errormessage: "Message creation error" }` if an error occurs while saving the message in the DB.
- HTTP code 200 with this body `{ error: false, errormessage: "Message has been saved correctely" }` if the message has been saved and sent correctely
- HTTP code 404 with this body `{ statusCode: 404, errormessage: "The user is not a friend" }` if the user specified in the body is not a friend
- HTTP code 404 with this body `{ statusCode: 404, errormessage: "The user does not exist" }` if the user specified in the body does not exist
- HTTP code 400 with this body `{statusCode: 400, errormessage: "Bad Request"}` if both `receiver` and `message` fields are not included in the body of the request
- HTTP code 404 with this body `{statusCode: 404, errormessage: "DB error"}` if an operation on the DB raise an error
- HTTP code 401 with this body `{ statusCode: 401, errormessage: "Unauthorized" }` if the user that made the request is not authorized for that operation

Below is an example of the body for this request, all specified fields are mandatory

Method: POST

> ```
> localhost:8080/message
> ```

Body (**raw**)

```
{
    "receiver" : "<username>",
    "message" : "<message text>"
}
```

🔑 Authentication bearer

Param	value	Type
token	{{JWT_TOKEN}}	string

End-point: Notify that messages from a user have been read

Description

This endpoint is used by the user to notify to the server that the inpending message from the specifed user are read.

Possible responses

This method responds:

- HTTP code 400 with this body `{ statusCode: 400, errormessage: "Bad Request" }` if the `sender` field is not specified
- HTTP code 404 with this body `{ statusCode: 404, errormessage: "Message update error" }` if an error occurs while updating the messages
- HTTP code 200 with this body `{ error: false, errormessage: "Messages have been updated" }` if everything is ok
- HTTP code 404 with this body `{ statusCode: 404, errormessage: "There are no messages to be update" }` if there are no messages to read
- HTTP code 404 with this body `{statusCode: 404, errormessage: "DB error"}` if an operation on the DB raise an error
- HTTP code 401 with this body `{ statusCode: 401, errormessage: "Unauthorized" }` if the user that made the request is not authorized for that operation

Below is an example of the body for this request, sender is mandatory.

If you set modMessage to true you will work only with moderator message.

Method: PUT

> ```
> localhost:8080/message
> ```

Body (**raw**)

```
{
    "sender" : "<username of the sender of the message>",
    "modMessage": "<bool>"
}
```

Authentication bearer

Param	value	Type
token	{{JWT_TOKEN}}	string

- -

End-point: Send moderator message to user

Description

This endpoint is used to send moderator messages to another user, so one of them must be a moderator, otherwise this request cannot be completed.

Available listeners

- ***message*** event, the target user of the message receive `{'_id' : "", 'content' : "", 'sender' : "", 'receiver' : "", 'timestamp' : {"$date" : ""}, inpending: '', isAModMessage: true}` that correspond to the message document in the DB.

Possible responses

This method responds:

- HTTP code 400 with this body `{ statusCode: 400, errormessage: 'You should send receiver and message body' }` if the `receiver` or the `message` field is missing
- HTTP code 200 with this body `{ error: false, errormessage: "Message from admin has been saved correctely" }` if the messahe has been saved and sent correctely
- HTTP code 404 with this body `{ statusCode: 404, errormessage: "The opposite player is not a friend" }` if the receiver with the username specified in the body is not a friend
- HTTP code 404 with this body `{ statusCode: 404, errormessage: "The user does not exist" }` if the friend specified in the body by the username does not exist
- HTTP code 404 with this body `{ statusCode: 404, errormessage: "Message creation error" }` if during the creation of the message into the database there was an error
- HTTP code 401 with this body `{ statusCode: 401, errormessage: "Unauthorized" }` if the user that made the request is not authorized for that operation

Below is an example of the body for this request, all specified fields are mandatory

Method: POST

> ```
> localhost:8080/message/mod
> ```

Body (**raw**)

```
{
    "receiver": "<username>",
    "message": "<message text>"
}
```

Authentication bearer

Param	value	Type
token	{{JWT_TOKEN}}	string

End-point: Get moderator message

Description

This endpoint returns to the user all the in pending messages and all messages, in two different arrays.

You can filter messages using ModMessage=true to obtain only moderator messages

Possible response

This method respond:

- HTTP code 200 with this body `{ error: false, inPendingMessages: inPendingMessages, allMessages: allMessages }` if everything is ok
- HTTP code 404 with this body `{ statusCode: 404, errormessage: "Error getting messages from DB"}` if an error occurs while getting messages from the DB
- HTTP code 401 with this body `{ statusCode: 401, errormessage: "Unauthorized" }` if the user that made the request is not authorized for that operation
- HTTP code 404 with this body `{statusCode: 404, errormessage: "DB error"}` if an operation on the DB raise an error

Method: GET

```
localhost:8080/message?ModMessage=<bool>
```

Query Params

Param	value
ModMessage	

🔑 Authentication bearer

Param	value	Type
token	{{JWT_TOKEN}}	string

End-point: Getting endpoints list

List of available endpoints in specified version

This will return, a response with HTTP status code 200, with this body:

```
{
  api_version: "1.0",
  endpoints: ["/", "/login", "/users", "/game","/gameMessage", "/notification",
"/friend", "/message", "/move", "/whoami",]
}
```

Method: GET

> ```
> localhost:8080
> ```

- -

Powered By: postman-to-markdown