

Simone Bozzolan
878352

Battleship Report (TAW Project)

System Architecture	2
Data Model	4
Endpoints	7
Authentication	34
Front-end	35
Typical Workflow	37

1. System Architecture

On a macro level the system is divided into three components: the front-end, the back-end and the database. The database is a non relational database, more specifically it's a MongoDB database, and it communicates only with the back-end. The database stores all of the necessary data for the application to run correctly, to learn more about what exactly is stored and how the data is organized head over to section 2. The back-end is written in TypeScript, runs in NodeJS and uses Mongoose to communicate with the database. It's a REST-style back-end that offers a handful of endpoints that are then used by the front-end. Therefore, it offers a RESTful API and its endpoints are routed and managed using Express. For a complete list of all of the endpoints, together with their parameters and what data is exchanged, go to section 3. The front-end is written using the Angular framework and only communicates with the back-end, through the API. For a complete list describing all of the Angular components and services go to section 5.

The back-end consists of routes and models. The models describe how and what data is stored in the database together with the functions needed to manage such data. For example, in the user model there will be the user schema, so the description of what is stored in the database for each user (name, surname, username, mail, role, friend list...), together with the functions needed to hash the password for a newly created user, check the password when a user logs in and every other function needed to manage the user's data. The models are then used by Mongoose to communicate with the MongoDB database. The routes, instead, are basically the functions that are executed for each endpoint. For each endpoint there is a list of middlewares associated with it, some are in common between all endpoints, and when the back-end receives an HTTP request on a certain endpoint all of the middlewares associated with that endpoint are executed. A middleware is essentially a function, so for example when the back-end receives a request in the /user endpoint all of the middlewares associated with the /user route are executed. In this case, first there would be a CORS check, then the body of the request would be parsed, then the request would be printed in the console and then if the method is POST a new user would be added to the database, and if everything is fine an OK response is sent, otherwise an error is thrown. Hence, for each endpoint there is a list of middlewares (functions) associated with it, and the routing of each request, so which middlewares to execute when receiving a certain request, is done using Express. Finally, because normally it's not possible to send a request from the server to the client, in order to notify the clients when something happens socket.io is used. For example, when a user (A) sends a message to another user (B), A sends the message to the server that saves the message in the database and notifies B, using socket.io, that a new message is available. Socket.io uses the WebSocket protocol and makes it possible to have a bidirectional event-based communication between server and client. Thus, some endpoints emit socket.io events together with the normal HTTP response.

The front-end consists of components and services. Services are used to communicate with the back-end, both for normal HTTP requests and for socket.io events. They are a collection of functions that are used to send HTTP requests, receive HTTP responses and listen to socket.io events. Components, instead, are the graphical elements of the application. They manage the user interface, so they can receive data from the services and they contain all of the functions needed to manage the user interaction and to format and display the data. For

example, if a user wants to see his/her friends list, the friends component asks the friends service to retrieve the user's friends list from the back-end (through an HTTP request) and when the data is retrieved the friends component formats it and prints it as a list on the user's screen. Another example would be if the user clicks on the "delete friend" button, then a function in the friends component will be triggered and this function will ask the friends service to send a delete friend request to the back-end. Components will also ask the socket.io service to listen to certain events if needed, for example the friends component will listen to the "new friend request" event so that when a new friend request arrives it is displayed on the screen for the user to see.

2. Data Model

The database is made up of five collections: chats, matches, messages, notifications, users. An example of a user document is the following:

```
{ "_id": {"$oid": "6293bd929f3a4c036633adf2"},
  "name": "simone",
  "surname": "rosada",
  "username": "simo",
  "mail": "simo.ross@gmail.com",
  "role": "MODERATOR",
  "friends_list": [{" $oid": "6293bde29f3a4c036633adf8"}, {" $oid": "62949d4c01b28e1289a15d2d"} ],
  "friend_requests": [],
  "match_invites": [],
  "temporary": false,
  "salt": "4182f77a405591be669091de6c2edf36",
  "digest": "05e41ac94d49629581a2634853c4a182625ba88fcab877dd0c550e872a320846d421d92b7545e5159cc04066e327276a4454a7fc9de79c9ccf751137c09083d9",
  "__v": 0 }
```

Both `_id` and `username` are unique, `friends_list` is a list of all of the users' ids that are in the user's friends list, `friend_requests` is a list of all of the users' ids that have sent the user a friend request and he hasn't accepted nor rejected yet, `match_invites` is a list of all of the users' ids that have sent the user a match invite and he hasn't accepted nor rejected yet, `temporary` is true if the user's account has been created by a moderator, so he has to change password and fill in his data on first login, and the `salt` and `digest` are used to hash and check the password.

An example of a message document is the following:

```
{ "_id": {"$oid": "6293c57e1bb6639f9328c2a5"},
  "owner": {"$oid": "6293bd929f3a4c036633adf2"},
  "owner_username": "simo",
  "content": "vediamo",
  "visibility": true,
  "createdAt": {"$date": "2022-05-29T19:11:58.434Z"},
  "updatedAt": {"$date": "2022-05-29T19:11:58.434Z"},
  "__v": 0 }
```

`Owner` and `owner username` are the id and username of the user that created that message, `content` is the message itself, and `visibility` is set to false only if the message was written by a user observing a match, so that it is invisible to the players. `createdAt` and `updatedAt` are the timestamps, managed by the database.

An example of a matches document is the following:

```
{
  "_id": {"$oid": "6294ba20514de672ab8d70d6"},
  "playerOne": {"$oid": "6293bde29f3a4c036633adf8"},
  "playerTwo": {"$oid": "6293bd929f3a4c036633adf2"},
  "gridOne": [
    ["s", "s", "s", "s", "s", "b", "b", "s", "s", "s"],
    ["s", "b", "b", "b", "s", "s", "s", "s", "b", "s"],
    ["b", "s", "s", "s", "b", "s", "s", "s", "s", "s"],
    ["s", "s", "b", "s", "b", "s", "h", "b", "b", "b"],
    ["b", "s", "b", "s", "s", "s", "s", "s", "s", "s"],
    ["b", "s", "s", "s", "s", "s", "s", "s", "b", "b"],
    ["b", "s", "s", "s", "b", "b", "s", "b", "s", "s", "s", "b", "b"],
    ["b", "s", "s", "s", "s", "b", "s", "s", "s", "b", "s", "s", "s", "b", "b"],
    ["b", "b", "s", "s", "s", "b", "b", "b", "b", "b"],
    ["s", "s", "b", "s", "s", "s", "s", "s", "s", "s"],
    ["b", "s", "s", "s", "s", "b", "b", "b", "s", "b", "s"],
    ["s", "s", "s", "s", "s", "s", "s", "s", "s", "b", "s"],
    ["b", "b", "b", "b", "s", "s", "s", "s", "b", "s", "s"],
    ["s", "s", "s", "s", "b", "b", "b", "s", "b", "s", "s"],
    ["s", "b", "b", "s", "s", "s", "s", "s", "b", "b"],
    ["b", "s", "s", "s", "s", "s", "s", "s", "s", "b", "b"],
    ["b", "s", "s", "s", "s", "s", "s", "s", "s", "s", "s", "b", "b"]
  ],
  "moves": ["G5"],
  "result": "0-0",
  "chat": {"$oid": "6294ba20514de672ab8d70d4"},
  "startingPlayer": {"$oid": "6293bd929f3a4c036633adf2"},
  "createdAt": {"$date": "2022-05-30T12:35:44.476Z"},
  "updatedAt": {"$date": "2022-05-30T12:41:20.909Z"},
  "__v": 0
}
```

PlayerOne and playerTwo are the ids of the two players, gridOne and gridTwo are the two grids, moves is a list of the moves that have been made, result is the result of the match (it will be either 1-0 or 0-1 once the match is finished, 0-0 if the match hasn't finished yet), chat is the id of the chat associated with this match and startingPlayer is the id of the player that made the first move, and is randomly selected.

An example of a chat document is the following:

```
{
  "_id": {"$oid": "6293c57b1bb6639f9328c29f"},
  "participants": [
    {"$oid": "6293bd929f3a4c036633adf2"},
    {"$oid": "6293bde29f3a4c036633adf8"}
  ],
  "messages": [
    {"$oid": "6293c57e1bb6639f9328c2a5"},
    {"$oid": "6293c5ea1bb6639f9328c2c2"}
  ],
  "type": "friend",
  "__v": 0
}
```

Participants is a list of all of the users' ids that have joined the chat, messages is a list of all of the messages' ids that belong to this chat and type is the type of the chat, which can either be "friend" for the friends chat, "match" for the match chat and "moderator" for the chat between a user and a moderator.

An example of a notification document is the following:

```
{
  "_id": {"$oid": "62949d4c01b28e1289a15d30"},
  "user": {"$oid": "62949d4c01b28e1289a15d2d"},
  "friend_request": false,
  "match_invite": false,
  "friend_request_accepted": true,
  "friend_messages": [],
  "moderator_messages": []
}
```

```
"match_alerts":[],  
"__v":0}
```

The notifications collection is used so that if the user receives a notification while he's offline it is stored in the database and next time the user logs in he can see all of his notifications. friend_request is a new friend request notification, match_invite is a new match invite notification, friend_request_accepted is a new friend added to the friends list notification, friend_messages is a list of all of the chats' ids where I received a new message (only chats of type "friend"), moderator_messages is a list of all of the chats' ids where I received a new message (only chats of type "moderator") and match_alerts is a list of all of the matches' ids where something new happened in that match.

3. Endpoints

POST '/user'

Add a new user to the database.

Request Headers:

Content-type: application/json

Request Body:

```
{
  name: "federico"
  surname: "verdi"
  username: "fed"
  mail: "fede.verdi@gmail.com"
  password: "secretpassword"
}
```

Response Headers:

Content-type: application/json

Response Body:

```
{
  error: false,
  errormessage: "",
  id: "62962327653ecdc9339c9575"
}
```

GET '/login'

Login a user, for more information on how the authentication is handled go to section 4.

Request Headers:

Authorization: Basic bmFubmk6YmVsbGE=

Request Body:

none

Response Headers:

Content-type: application/json

Response Body:

```
{
  error: false,
  errorMessage: "",
  token:
    "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6Im5hbm5pliwiImFtZSI6Ikdwb3ZhbW5pliwic3VybmFtZSI6Im1hbmVudGUiLCJtYWwIjoiZ2lvLm1hbkbNbWFpYm5jb20iLCJyb2xlIjoiaVNFUilslmkljoij5NDIkNGMwMWlyOGUxMjg5YTE2ZDJkIiwidGVtcG9yYXJ5IjpmYWxzZSwiaWF0IjoxNjU0MDA3MDcwLCJleHAiOiJlE2NTQwMjUwNzB9.i2XwFp140Ar7ZFER7w3ArJzLjRroWJ6fPQI8Ejrc3Xw",
  temporary: false
}
```

GET '/users/username/:username'

Retrieve the user from the username.

Request Headers:

Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6Im5hbm5pliwbmFtZSI6Ikdwb3ZhbW5pliwic3VybmFtZSI6Im1hbmVudGUIlCJtYWIsIjoziZlVlM1hbkBnbWFpbC5jb20iLCJyb2x1IjoivVNFIiIsImklkjoijNjI5NDIkNGMwMWIyOGUxMjg5YTE1ZDJkIiwidGVtcG9yYXJ5IjpmYWxzZSwiaWF0IjoxNjU0MDA3MDcwLCJleHAiOiJlE2NTQwMjUwNzB9.i2XwFp140Ar7ZFER7w3ArJzLjRroWJ6fPQI8Ejrc3Xw

Request Body:

none

Response Headers:

Content-type: application/json

Response Body:

```
{
  "_id": "6293bd929f3a4c036633adf2",
  "name": "simone",
  "surname": "rosada",
  "username": "simo",
  "mail": "simo.ross@gmail.com",
  "role": "MODERATOR",
  "friends list": [
```



```
        "6293bde29f3a4c036633adf8",
        "62949d4c01b28e1289a15d2d"
    ],
    "friend_requests": [],
    "match_invites": [],
    "temporary": false,
    "__v": 0
}
```

GET '/users/id/:userid'

Retrieve the user from the id.

Request Headers:

Authorization: Bearer

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6Im5hbm5pliwbmFtZSI6Ikdpb3ZhbW5pliwic3VybmFtZSI6Im1hbmVudGUiLCJtYWlsIjoiz2lvbm1hbkBnbWFpbiC5jb20iLCJyb2xlIjoivVNFUilsImkljoiNjI5NDIkdGMwMWlyOGUxMjg5YTE1ZDJKliwidGVtcG9yYXJ5IjpmYWxzZSwiaWF0IjoxNjU0MDA3MDcwLCJleHAiOjE2NTQwMjUwNzB9.i2XwFp140Ar7ZFER7w3ArJzLjRroWJ6fPQI8Ejrc3Xw
```

Request Body:

none

Response Headers:

Content-type: application/json

Response Body:

```
{
  "_id": "6293bd929f3a4c036633adf2",
  "name": "simone",
  "surname": "rosada",
  "username": "simo",
  "mail": "simo.ross@gmail.com",
  "role": "MODERATOR",
  "friends_list": [
    "6293bde29f3a4c036633adf8",
    "62949d4c01b28e1289a15d2d"
  ],
  "friend_requests": [],
  "match_invites": [],
  "temporary": false,
  "__v": 0
}
```

```
}
```

GET '/users/stats/:userid'

Retrieve the user's stats from the id.

Request Headers:

Authorization: Bearer

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6Im5hbm5pliwbmFtZSI6Ikdpb3ZhbW5pliwic3VybmFtZSI6Im1hbmVudGUiLCJtYWlsIjoiz2lvLm1hbkBnbWFpbC5jb20iLCJyb2xlljoiVFNFUilsImkljoiNjI5NDIkdGMwMWlyOGUxMjg5YTE1ZDJKliwidGVtcG9yYXJ5IjpmYWxzZSwiaWF0IjoxNjU0MDA3MDcwLCJleHAiOjE2NTQwMjUwNzB9.i2XwFp140Ar7ZFER7w3ArJzLjRroWJ6fPQI8Ejrc3Xw
```

Request Body:

none

Response Headers:

Content-type: application/json

Response Body:

```
{
  "_id": null,
  "wins": 0,
  "losses": 0
}
```

POST '/users/moderator'

Add a new moderator user to the database.

Request Headers:

Content-type: application/json

Authorization: Bearer

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6Im5hbm5pliwbmFtZSI6Ikdpb3ZhbW5pliwic3VybmFtZSI6Im1hbmVudGUiLCJtYWlsIjoiz2lvLm1hbkBnbWFpbC5jb20iLCJyb2xlljoiVFNFUilsImkljoiNjI5NDIkdGMwMWlyOGUxMjg5YTE1ZDJKliwidGVtcG9yYXJ5IjpmYWxzZSwiaWF0IjoxNjU0MDA3MDcwLCJleHAiOjE2NTQwMjUwNzB9.i2XwFp140Ar7ZFER7w3ArJzLjRroWJ6fPQI8Ejrc3Xw
```

Request Body:

```
{
  username: "fede"
  password: "secretpassword"
}
```

Response Headers:

Content-type: application/json

Response Body:

```
{
  error: false,
  errormessage: "",
  id: "62962327653ecdc9339c9575"
}
```

PATCH '/users/:username'

Modify a user.

Request Headers:

Content-type: application/json

Authorization: Bearer

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6Im5hbm5pliwiFtZSI6Imdpb3ZhbW5pliwiFtZSI6Im1hbmVudGUlLCJtYWlsIjoZ2lvLm1hbkBnbWFpbC5jb20iLCJyb2xlIjoVVFUilSI6ImkljoiNjI5NDIkdGMwMWlyOGUxMjg5YTE1ZDJkIiwidGVtcG9yYXJ5IjpmYWxzZSwiaWF0IjoxNjU0MDA3MDcwLCJleHAiOjE2NTQwMjUwNzB9.i2XwFp140Ar7ZFER7w3ArJzLjRroWJ6fPQl8Ejrc3Xw

Request Body:

```
{
  name: "federico"
  surname: "verdi"
  username: "fede"
  mail: "fede.verdi@gmail.com"
  password: "secretpassword"
}
```

Response Headers:

Content-type: application/json

Response Body:

```
{
  error: false,
  errorMessage: "",
  id: "62962327653ecdc9339c9575"
}
```

DELETE '/users/:username'

Delete a user.

Request Headers:

Authorization: Bearer

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6Im5hbm5pliwbmFtZSI6Ikdwb3ZhbW5pliwic3VybmFtZSI6Im1hbmVudGUiLCJtYWlsIjoiz2lvLm1hbkbNnbWFpbC5jb20iLCJyb2xlljoivVNFUiIsImkljoijNj15NDIkNGMwMWlyOGUxMjg5YTE1ZDJkIiwidGVtcG9yYXJ5IjpmYWxzZSwiaWF0IjoxNjU0MDA3MDcwLCJleHAiOiE2NTQwMjUwNzB9.12XwFp140Ar7ZFER7w3ArJzLjRroWJ6fPQI8Ejrc3Xw

Request Body:

none

Response Headers:

Content-type: application/json

Response Body:

```
{
  error: false,
  errorMessage: "",
  id: "62962327653ecdc9339c9575"
}
```

POST '/friends/request'

Send, accept or reject a friend request. The database will be updated and the corresponding socket.io events will be emitted. Action field in the body can be either “send”, “accept” or “reject” and the username field is the username of the target user.

Request Headers:

Content-type: application/json

Authorization: Bearer

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6Im5hbm5pliwbmFtZSI6Ikdwb3ZhbW5pliwc3VybmFtZSI6Im1hbmVudGUlLCJtYWlsIjoiz2ZlLm1hbkbNbnWFpbC5jb

20iLCJyb2xlljoiVVNFUilsImkljoiNjl5NDIkNGMwMWlyOGUxMjg5YTE1ZDJkIiwidGVtcG9yYXJ5IjpmYWxzZSwiaWF0IjoxNjU0MDA3MDcwLCJleHAiOjE2NTQwMjUwNzB9.i2XwFp140Ar7ZFER7w3ArJzLjRroWJ6fPQI8Ejrc3Xw

Request Body:

```
{
  username: "fede"
  action: "send"
}
```

Response Headers:

Content-type: application/json

Response Body:

```
{
  "_id": "62962327653ecdc9339c9575",
  "name": "federico",
  "surname": "verdi",
  "username": "fede",
  "mail": "fede.verdi@gmail.com",
  "role": "USER",
  "friends_list": [],
  "friend_requests": [],
  "match_invites": [],
  "temporary": false,
  "__v": 0
}
```

POST '/friends/play'

Invite, accept or reject a match request. The database will be updated and the corresponding socket.io events will be emitted. Action field in the body can be either "invite", "accept" or "reject" and the username field is the username of the target user.

Request Headers:

Content-type: application/json

Authorization: Bearer

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6Im5hbm5pliwiFtZSI6Imkdpb3ZhbW5pliwiFtZSI6Im5hbmVudGUlLCJtYWlsIjoiZ2lvLm1hbkBnbWFpbC5jb20iLCJyb2xlljoiVVNFUilsImkljoiNjl5NDIkNGMwMWlyOGUxMjg5YTE1ZDJkIiwidGVtcG9yYXJ5IjpmYWxzZSwiaWF0IjoxNjU0MDA3MDcwLCJleHAiOjE2NTQwMjUwNzB9.i2XwFp140Ar7ZFER7w3ArJzLjRroWJ6fPQI8Ejrc3Xw

Request Body:

```
{
  username: "fede"
  action: "invite"
}
```

Response Headers:

Content-type: application/json

Response Body:

```
{
  "_id": "62962327653ecdc9339c9575",
  "name": "federico",
  "surname": "verdi",
  "username": "fede",
  "mail": "fede.verdi@gmail.com",
  "role": "USER",
  "friends_list": [],
  "friend_requests": [],
  "match_invites": [],
  "temporary": false,
  "__v": 0
}
```

DELETE '/friends/:username'

Delete a friend from the friends list.

Request Headers:

Authorization: Bearer

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6Im5hbm5pliwiFtZSI6Imdpb3ZhbW5pliwiFtZSI6Im1hbmVudGUlLCJtYWlsIjoiz2lvLm1hbkbWpC5jb20iLCJyY2xldjoiVjVNFUilImkljoiNjI5NDIkdGMwMWlyOGUxMjg5YTE1ZDJKliwidGVtcG9yYXJ5IjpmYWxzZSwiaWF0IjoxNjU0MDA3MDcwLCJleHAiOjE2NTQwMjUwNzB9.i2XwFp140Ar7ZFER7w3ArJzLjRroWJ6fPQl8Ejrc3Xw

Request Body:

none

Response Headers:

Content-type: application/json

Response Body:

```
{
  "_id": "62962327653ecdc9339c9575",
  "name": "federico",
  "surname": "verdi",
  "username": "fede",
  "mail": "fede.verdi@gmail.com",
  "role": "USER",
  "friends_list": [],
  "friend_requests": [],
  "match_invites": [],
  "temporary": false,
  "__v": 0
}
```

POST '/matches/grid/:matchid'

Submit your grid. The database will be updated (if the grid is valid) and the corresponding socket.io events will be emitted.

Request Headers:

Content-type: application/json

Authorization: Bearer

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6Im5hbm5pliwbmFtZSI6Ikdpb3ZhbW5pliwc3VybmFtZSI6Im1hbmVudGUlLCJtYWlsIjoiz2lvLm1hbkBnbWFpC5jb20iLCJyb2xlIjoivVNFUilsImkljoiNjI5NDIkNGMwMWIyOGUxMjg5YTE1ZDJkIiwidGVtcG9yYXJ5IjpmYWxzZSwiaWF0IjoxNjU0MDA3MDcwLCJleHAiOjE2NTQwMjUwNzB9.i2XwFp140Ar7ZFER7w3ArJzLjRroWJ6fPQI8Ejrc3Xw
```

Request Body:

```
{
  "grid": [
    ["s", "s", "s", "b", "s", "s", "s", "s", "s", "s"],
    ["s", "s", "s", "b", "s", "s", "s", "s", "s", "b"],
    ["b", "s", "s", "b", "s", "b", "b", "b", "s", "b"],
    ["b", "s", "s", "s", "s", "s", "s", "s", "s", "b"],
    ["b", "s", "b", "b", "s", "b", "b", "s", "s", "s"],
    ["b", "s", "s", "s", "s", "s", "s", "s", "s", "s"],
    ["s", "s", "s", "b", "b", "b", "b", "b", "s", "b"],
    ["s", "b", "b", "s", "s", "s", "s", "s", "s", "b"],
    ["s", "s", "s", "s", "s", "s", "b", "b", "s", "s"],
    ["s", "s", "b", "b", "b", "b", "s", "s", "s", "s"]
  ]
}
```

```
}
```

Response Headers:

Content-type: application/json

Response Body:

```
{
  "_id": "6294ba20514de672ab8d70d6",
  "playerOne": "6293bde29f3a4c036633adf8",
  "playerTwo": "6293bd929f3a4c036633adf2",
  "gridOne": [
    ["s", "s", "s", "b", "s", "s", "s", "s", "s", "s"],
    ["s", "s", "s", "b", "s", "s", "s", "s", "s", "b"],
    ["b", "s", "s", "b", "s", "b", "b", "b", "s", "b"],
    ["b", "s", "s", "s", "s", "s", "s", "s", "s", "b"],
    ["b", "s", "b", "b", "s", "b", "b", "s", "s", "s"],
    ["b", "s", "s", "s", "s", "s", "s", "s", "s", "s"],
    ["s", "s", "s", "b", "b", "b", "b", "b", "s", "b"],
    ["s", "b", "b", "s", "s", "s", "s", "s", "s", "b"],
    ["s", "s", "s", "s", "s", "s", "b", "b", "s", "s"],
    ["s", "s", "b", "b", "b", "b", "s", "s", "s", "s"]
  ],
  "gridTwo": [],
  "moves": [],
  "result": "0-0",
  "chat": "6294ba20514de672ab8d70d4",
  "startingPlayer": "6293bd929f3a4c036633adf2",
  "createdAt": "2022-05-30T12:35:44.476Z",
  "updatedAt": "2022-05-30T12:41:20.909Z",
  "__v": 0
}
```

POST '/matches/move/:matchid'

Submit your move. The database will be updated and the corresponding socket.io events will be emitted. It will also determine if a match is finished and update the database accordingly, together with emitting the right socket.io events.

Request Headers:

Content-type: application/json

Authorization: Bearer

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6Im5hbm5pliwiFtZSI6Ikdwb3ZhbW5pliwiFtZSI6Im5hbmVudGUlLCJtYWlsIjoizZlVlM1hbkBnbWFpYjB20iLCJyb2xlIjoivVNFUilslmkljoiNjI5NDIkNGMwMWlyOGUxMjg5YTE1ZDJkIiwidGVtcG9
```


yYXJ5ljpmYWxzZSwiaWF0IjoxNjU0MDA3MDcwLCJleHAiOjE2NTQwMjUwNzB9.i2XwFp140Ar7ZFER7w3ArJzLjRroWJ6fPQl8Ejrc3Xw

Request Body:

```
{  
  "move": "B6"  
}
```

Response Headers:

Content-type: application/json

Response Body:

```
{  
  "_id": "6294ba20514de672ab8d70d6",  
  "playerOne": "6293bde29f3a4c036633adf8",  
  "playerTwo": "6293bd929f3a4c036633adf2",  
  "gridOne": [  
    ["s", "s", "s", "b", "s", "s", "s", "s", "s", "s"],  
    ["s", "s", "s", "b", "s", "s", "s", "s", "s", "b"],  
    ["b", "s", "s", "b", "s", "b", "b", "b", "s", "b"],  
    ["b", "s", "s", "s", "s", "s", "s", "s", "s", "b"],  
    ["b", "s", "b", "b", "s", "b", "b", "s", "s", "s"],  
    ["b", "s", "s", "s", "s", "s", "s", "s", "s", "s"],  
    ["s", "s", "s", "b", "b", "b", "b", "b", "s", "b"],  
    ["s", "b", "b", "s", "s", "s", "s", "s", "s", "b"],  
    ["s", "s", "s", "s", "s", "s", "b", "b", "s", "s"],  
    ["s", "s", "b", "b", "b", "b", "s", "s", "s", "s"]  
  ],  
  "gridTwo": [  
    ["s", "s", "s", "b", "s", "s", "s", "s", "s", "s"],  
    ["s", "s", "s", "b", "s", "s", "s", "s", "s", "b"],  
    ["b", "s", "s", "b", "s", "b", "b", "b", "s", "b"],  
    ["b", "s", "s", "s", "s", "s", "s", "s", "s", "b"],  
    ["b", "s", "b", "b", "s", "b", "b", "s", "s", "s"],  
    ["b", "s", "s", "s", "s", "s", "s", "s", "s", "s"],  
    ["s", "s", "s", "b", "b", "b", "b", "b", "s", "b"],  
    ["s", "b", "b", "s", "s", "s", "s", "s", "s", "b"],  
    ["s", "s", "s", "s", "s", "s", "b", "b", "s", "s"],  
    ["s", "s", "b", "b", "b", "b", "s", "s", "s", "s"]  
  ],  
  "moves": ["B6"],  
  "result": "0-0",  
  "chat": "6294ba20514de672ab8d70d4",  
  "startingPlayer": "6293bd929f3a4c036633adf2",  
  "createdAt": "2022-05-30T12:35:44.476Z",  
}
```

}

GET '/matches'

Retrieve the matches.

URI parameters:

- result (required): filter the games by result (get only ongoing game or finished games)

Request Headers:

Authorization: Bearer

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6Im5hbm5pliwiFtZSI6Imdpb3ZhbM5pliwiFtZSI6Im1hbmVudGUlLCJtYWlsIjoZ2lvLm1hbkBnbWFpbC5jb20iLCJyb2xlljoVNFUiSlmkljoijNj5NDIkNGMwMWlyOGUxMjg5YTE1ZDJkIiwidGVtcG9yYXJ5IjpmYWxzZSwiaWF0IjoxNjU0MDA3MDcwLCJleHAiOiE2NTQwMjUwNzB9.12XwFp140Ar7ZFER7w3ArJzLjRroWJ6fPQI8Ejrc3Xw

Request Body:

none

Response Headers:

Content-type: application/json

Response Body:

```
[
{
  "_id": "6294ba20514de672ab8d70d6",
  "playerOne": "6293bde29f3a4c036633adf8",
  "playerTwo": "6293bd929f3a4c036633adf2",
  "gridOne":
    [
      ["s", "s", "s", "s", "s", "b", "b", "s", "s", "s"],
      ["s", "b", "b", "b", "s", "s", "s", "s", "b", "s"],
      ["b", "s", "s", "s", "b", "s", "s", "s", "b", "s"],
      ["b", "s", "s", "s", "b", "s", "s", "s", "s", "s"],
      ["s", "s", "b", "s", "b", "s", "h", "b", "b", "b"],
      ["b", "s", "b", "s", "s", "s", "s", "s", "s", "s"],
      ["b", "s", "s", "s", "s", "s", "s", "s", "s", "b"],
      ["b", "s", "s", "s", "s", "b", "s", "s", "s", "b"],
      ["b", "s", "b", "b", "s", "b", "s", "s", "s", "b"]
    ]
}
```

```

        ["b","s","s","s","s","b","s","s","s","b"]
    ],
    "gridTwo":
    [
        ["b","b","s","s","s","b","b","b","b","b"],
        ["s","s","b","s","s","s","s","s","s","s"],
        ["b","s","b","s","s","s","s","s","s","s"],
        ["b","s","s","s","b","b","b","s","b","s"],
        ["s","s","s","s","s","s","s","s","b","s"],
        ["b","b","b","b","s","s","s","s","b","s"],
        ["s","s","s","s","b","b","b","s","b","s"],
        ["s","b","b","s","s","s","s","s","s","b"],
        ["b","s","s","s","s","s","s","s","s","b"],
        ["b","s","s","s","s","s","s","s","s","b"]
    ],
    "moves":["G5"],
    "result":"0-0",
    "chat":"6294ba20514de672ab8d70d4",
    "startingPlayer":"6293bd929f3a4c036633adf2",
    "createdAt":"2022-05-30T12:35:44.476Z",
    "updatedAt":"2022-05-30T12:41:20.909Z",
    "__v":0
  }
]

```

GET '/matches/id/:matchid'

Retrieve a match from its id.

Request Headers:

Authorization: Bearer

```

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6Im5hbm5pliwbmFtZSI6Ikdpb3ZhbW5pliwic3VybmFtZSI6Im1hbmVudGUIiLCJtYWlsIjoiz2lvLm1hbkbNbnWFpbC5jb20iLCJyb2xlIjoivVNFUilslmlkljoiNjI5NDIkdNGMwMWlyOGUxMjg5YTE1ZDJKliwidGVtcG9yYXJ5IjpmYWxzZSwiaWF0IjoxNjU0MDA3MDcwLCJleHAiOjE2NTQwMjUwNzB9.i2XwFp140Ar7ZFER7w3ArJzLjRroWJ6fPQI8Ejrc3Xw

```

Request Body:

none

Response Headers:

Content-type: application/json

Response Body:

```

{
  "_id":"6294ba20514de672ab8d70d6",
  "playerOne":"6293bde29f3a4c036633adf8",
  "playerTwo":"6293bd929f3a4c036633adf2",
  "gridOne":
    [
      ["s","s","s","s","s","b","b","s","s","s"],
      ["s","b","b","b","s","s","s","s","b","s"],
      ["b","s","s","s","b","s","s","s","b","s"],
      ["b","s","s","s","b","s","s","s","s","s"],
      ["s","s","b","s","b","s","h","b","b","b"],
      ["b","s","b","s","s","s","s","s","s","s"],
      ["b","s","s","s","s","s","s","s","s","b"],
      ["b","s","s","s","s","b","s","s","s","b"],
      ["b","s","b","b","s","b","s","s","s","b"],
      ["b","s","s","s","s","b","s","s","s","b"]
    ],
  "gridTwo":
    [
      ["b","b","s","s","s","b","b","b","b","b"],
      ["s","s","b","s","s","s","s","s","s","s"],
      ["b","s","b","s","s","s","s","s","s","s"],
      ["b","s","s","s","b","b","b","s","b","s"],
      ["s","s","s","s","s","s","s","s","b","s"],
      ["b","b","b","b","s","s","s","s","b","s"],
      ["s","s","s","s","b","b","b","s","b","s"],
      ["s","b","b","s","s","s","s","s","s","b"],
      ["b","s","s","s","s","s","s","s","s","b"],
      ["b","s","s","s","s","s","s","s","s","b"]
    ],
  "moves":["G5"],
  "result":"0-0",
  "chat":"6294ba20514de672ab8d70d4",
  "startingPlayer":"6293bd929f3a4c036633adf2",
  "createdAt":"2022-05-30T12:35:44.476Z",
  "updatedAt":"2022-05-30T12:41:20.909Z",
  "__v":0
}

```

GET '/matches/mymatches'

Retrieve the logged in user's matches.

Request Headers:

Authorization: Bearer

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6Im5hbm5pliwbmFtZSI6Ikdpb3ZhbW5pliwic3VybmFtZSI6Im1hbmVudGUlLCJtYWlsIjoizZlvm1hbkBnbWpC5jb20iLCJyb2xlljoivVNFUilslmlkljoijNj15NDIkNGMwMWlyOGUxMjg5YTE1ZDJkIiwidGVtcG9yYXJ5IjpmYWxzZSwiaWF0IjoxNjU0MDA3MDcwLCJleHAiOiE2NTQwMjUwNzB9.12XwFp140Ar7ZFER7w3ArJzLjRroWJ6fPQI8Ejrc3Xw

Request Body:

none

Response Headers:

Content-type: application/json

Response Body:

```
[
{
  "_id":"6294ba20514de672ab8d70d6",
  "playerOne":"6293bde29f3a4c036633adf8",
  "playerTwo":"6293bd929f3a4c036633adf2",
  "gridOne":
    [
      ["s","s","s","s","s","b","b","s","s","s"],
      ["s","b","b","b","s","s","s","s","b","s"],
      ["b","s","s","s","b","s","s","s","b","s"],
      ["b","s","s","s","b","s","s","s","s","s"],
      ["s","s","b","s","b","s","h","b","b","b"],
      ["b","s","b","s","s","s","s","s","s","s"],
      ["b","s","s","s","s","s","s","s","s","b"],
      ["b","s","s","s","s","b","s","s","s","b"],
      ["b","s","b","b","s","b","s","s","s","b"],
      ["b","s","s","s","s","b","s","s","s","b"]
    ],
  "gridTwo":
    [
      ["b","b","s","s","s","b","b","b","b","b"],
      ["s","s","b","s","s","s","s","s","s","s"],
      ["b","s","b","s","s","s","s","s","s","s"],
      ["b","s","s","s","b","b","b","s","b","s"],
      ["s","s","s","s","s","s","s","s","b","s"],
      ["b","b","b","b","s","s","s","s","b","s"],
      ["s","s","s","s","b","b","b","s","b","s"],
      ["s","b","b","s","s","s","s","s","s","b"],
      ["b","s","s","s","s","s","s","s","s","b"],
      ["b","s","s","s","s","s","s","s","s","b"]
    ],
  "moves":["G5"],
}
```

]

GET '/matches/grid'

Generate a random valid grid.

Request Headers:

Authorization: Bearer

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6Im5hbm5pliwbmFtZSI6Ikdpb3ZhbW5pliwic3VybmFtZSI6Im1hbmVudGUiLCJtYWlsIjoizZlvm1hbkBnbWpC5jb20iLCJyb2xlljoivVNFUilslmlkljoijNj15NDIkNGMwMWlyOGUxMjg5YTE1ZDJKliwidGVtcG9yYXJ5IjpmYWxzZSwiaWF0IjoxNjU0MDA3MDcwLCJleHAiOiE2NTQwMjUwNzB9.12XwFp140Ar7ZFER7w3ArJzLjRroWJ6fPQI8Ejrc3Xw

Request Body:

none

Response Headers:

Content-type: application/json

Response Body:

```
{
  "grid":
  [
    ["s","s","s","s","s","b","b","s","s","s"],
    ["s","b","b","b","s","s","s","s","b","s"],
    ["b","s","s","s","b","s","s","s","b","s"],
    ["b","s","s","s","b","s","s","s","s","s"],
    ["s","s","b","s","b","s","h","b","b","b"],
    ["b","s","b","s","s","s","s","s","s","s"],
    ["b","s","s","s","s","s","s","s","s","b"],
    ["b","s","s","s","s","b","s","s","s","b"],
    ["b","s","b","b","s","b","s","s","s","b"],
    ["b","s","s","s","s","b","s","s","s","b"]
  ]
}
```

DELETE '/matches/retire/:matchid'

Forfeit an ongoing match. The corresponding socket.io events will be emitted.

Request Headers:

Authorization: Bearer

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6Im5hbm5pliwbmFtZSI6Imdpb3ZhbW5pliwic3VybmFtZSI6Im1hbmVudGUlLCJtYWlsIjoiz2lvLm1hbkBnbWFpbC5jb20iLCJyb2xlIjoivVNFUilsImklIjoibm91bnR5bW5kIj5NDIkNGMwMWlyOGUxMjg5YTE1ZDZkIiwidGVtcG9yYXJ5IjpmYWxzZSwiaWF0IjoxNjU0MDA3MDcwLCJleHAiOjE2NTQwMjUwNzB9.i2XwFp140Ar7ZFER7w3ArJzLjRroWJ6fPQl8Ejrc3Xw
```

Request Body:

none

Response Headers:

Content-type: application/json

Response Body:

```
{
  "_id": "6294ba20514de672ab8d70d6",
  "playerOne": "6293bde29f3a4c036633adf8",
  "playerTwo": "6293bd929f3a4c036633adf2",
  "gridOne":
    [
      ["s", "s", "s", "s", "s", "b", "b", "s", "s", "s"],
      ["s", "b", "b", "b", "s", "s", "s", "s", "b", "s"],
      ["b", "s", "s", "s", "b", "s", "s", "s", "b", "s"],
      ["b", "s", "s", "s", "b", "s", "s", "s", "s", "s"],
      ["s", "s", "b", "s", "b", "s", "h", "b", "b", "b"],
      ["b", "s", "b", "s", "s", "s", "s", "s", "s", "s"],
      ["b", "s", "s", "s", "s", "s", "s", "s", "s", "b"],
      ["b", "s", "s", "s", "s", "b", "s", "s", "s", "b"],
      ["b", "s", "b", "b", "s", "b", "s", "s", "s", "b"],
      ["b", "s", "s", "s", "s", "b", "s", "s", "s", "b"]
    ],
  "gridTwo":
    [
      ["b", "b", "s", "s", "s", "b", "b", "b", "b", "b"],
      ["s", "s", "b", "s", "s", "s", "s", "s", "s", "s"],
      ["b", "s", "b", "s", "s", "s", "s", "s", "s", "s"],
      ["b", "s", "s", "s", "b", "b", "b", "s", "b", "s"]
    ]
}
```

```

        ["s","s","s","s","s","s","s","s","b","s"],
        ["b","b","b","b","s","s","s","s","b","s"],
        ["s","s","s","s","b","b","b","s","b","s"],
        ["s","b","b","s","s","s","s","s","s","b"],
        ["b","s","s","s","s","s","s","s","s","b"],
        ["b","s","s","s","s","s","s","s","s","b"]
    ],
    "moves":["G5"],
    "result":"1-0",
    "chat":"6294ba20514de672ab8d70d4",
    "startingPlayer":"6293bd929f3a4c036633adf2",
    "createdAt":"2022-05-30T12:35:44.476Z",
    "updatedAt":"2022-05-30T12:41:20.909Z",
    "__v":0
}

```

POST '/chats'

Add a new chat to the database. The corresponding socket.io events will be emitted. Type field in the body can be either “friend”, “moderator” or “match”, while the participants field is a list of all of the users’ ids that will be part of the chat.

Request Headers:

Content-type: application/json

Authorization: Bearer

```

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6Im5hbm5pliwbmFtZSI6Ikdpb3ZhbW5pliwc3VybmFtZSI6Im1hbmVudGUlLCJtYWlsIjoiz2lvLm1hbkbNbnWFpbC5jb20iLCJyb2xlIjoivVNFUilsImklIjoibjI5NDIkdGMwMWlyOGUxMjg5YTE1ZDJKliwidGVtcG9yYXJ5IjpmYWxzZSwiaWF0IjoxNjU0MDA3MDcwLCJleHAiOjE2NTQwMjUwNzB9.i2XwFp140Ar7ZFER7w3ArJzLjRroWJ6fPQI8Ejrc3Xw

```

Request Body:

```

{
  participants: ["6293bd929f3a4c036633adf2"],
  type: "friend"
}

```

Response Headers:

Content-type: application/json

Response Body:

```

{
  "_id": "6293c57b1bb6639f9328c29f",

```



```

"participants": [
  "6293bd929f3a4c036633adf2",
  "6293bde29f3a4c036633adf8"
],
"messages": [],
"type": "friend",
"__v": 0
}

```

POST '/chats/:chatid/participants'

Add the logged in user to the participants of a chat.

Request Headers:

Authorization: Bearer

```

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6Im5hbm5pliwbmFtZSI6Ikdpb3ZhbW5pliwic3VybmFtZSI6Im1hbmVudGUiLCJtYWlsIjoiz2lvbm1hbkBnbWFpbC5jb20iLCJyb2xlljoiVVNFUilsImkljoiNjI5NDIkdGMwMWlyOGUxMjg5YTE1ZDJKliwidGVtcG9yYXJ5IjpmYWxzZSwiaWF0IjoxNjU0MDA3MDcwLCJleHAiOjE2NTQwMjUwNzB9.i2XwFp140Ar7ZFER7w3ArJzLjRroWJ6fPQI8Ejrc3Xw

```

Request Body:

none

Response Headers:

Content-type: application/json

Response Body:

```

{
  "_id": "6293c57b1bb6639f9328c29f",
  "participants": [
    "6293bd929f3a4c036633adf2",
    "6293bde29f3a4c036633adf8",
    "62962cb3c96d05d8dcd68df1"
  ],
  "messages": [],
  "type": "friend",
  "__v": 0
}

```

GET '/chats/chat/:chatid'

Retrieve a chat from its id.

Request Headers:

Authorization: Bearer

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6Im5hbm5pliwbmFtZSI6Imdpb3ZhbW5pliwic3VybmFtZSI6Im1hbmVudGUlLCJtYWlsIjoiz2lvbm1hbkBnbWFpbC5jb20iLCJyb2xlIjoivVNFUilsImklIjoijoiNjI5NDIkdGMwMWlyOGUxMjg5YTE1ZDJKliwidGVtcG9yYXJ5IjpmYWxzZSwiaWF0IjoxNjU0MDA3MDcwLCJleHAiOjE2NTQwMjUwNzB9.i2XwFp140Ar7ZFER7w3ArJzLjRroWJ6fPQI8Ejrc3Xw

Request Body:

none

Response Headers:

Content-type: application/json

Response Body:

```
{
  "_id": "6293c57b1bb6639f9328c29f",
  "participants": [
    "6293bd929f3a4c036633adf2",
    "6293bde29f3a4c036633adf8"
  ],
  "messages": [
    "6293c57e1bb6639f9328c2a5",
    "6293c5ea1bb6639f9328c2c2"
  ],
  "type": "friend",
  "__v": 0
}
```

GET '/chats/moderator/:participantid'

Retrieve all the moderator type chats given a certain participant id.

Request Headers:

Authorization: Bearer

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6Im5hbm5pliwbmFtZSI6Imdpb3ZhbW5pliwic3VybmFtZSI6Im1hbmVudGUlLCJtYWlsIjoiz2lvbm1hbkBnbWFpbC5jb20iLCJyb2xlIjoivVNFUilsImklIjoijoiNjI5NDIkdGMwMWlyOGUxMjg5YTE1ZDJKliwidGVtcG9yYXJ5IjpmYWxzZSwiaWF0IjoxNjU0MDA3MDcwLCJleHAiOjE2NTQwMjUwNzB9.i2XwFp140Ar7ZFER7w3ArJzLjRroWJ6fPQI8Ejrc3Xw

Request Body:

none

Response Headers:

Content-type: application/json

Response Body:

```
[
  {
    "_id": "6293c57b1bb6639f9328c29f",
    "participants": [
      "6293bd929f3a4c036633adf2",
      "6293bde29f3a4c036633adf8"
    ],
    "messages": [
      "6293c57e1bb6639f9328c2a5",
      "6293c5ea1bb6639f9328c2c2"
    ],
    "type": "friend",
    "__v": 0
  }
]
```

GET '/chats/friends'

Retrieve all the friend type chats of the logged in user.

Request Headers:

Authorization: Bearer

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6Im5hbm5pliwbmFtZSI6Imdpb3Zhbm5pliwic3VybmFtZSI6Im1hbmVudGUiLCJtYWlsIjoiz2lvbm1hbkbNnWFpbC5jb20iLCJyb2xlIjoivVNFUilsImkljoiNjI5NDIkdGMwMWIyOGUxMjg5YTE1ZDJKliwidGVtcG9yYXJ5IjpmYWxzZSwiaWF0IjoxNjU0MDA3MDcwLCJleHAiOjE2NTQwMjUwNzB9.i2XwFp140Ar7ZFER7w3ArJzLjRroWJ6fPQI8Ejrc3Xw

Request Body:

none

Response Headers:

Content-type: application/json

Response Body:

```
[
  {
    "_id": "6293c57b1bb6639f9328c29f",
    "participants": [
      "6293bd929f3a4c036633adf2",
      "6293bde29f3a4c036633adf8"
    ],
    "messages": [
      "6293c57e1bb6639f9328c2a5",
      "6293c5ea1bb6639f9328c2c2"
    ],
    "type": "friend",
    "__v": 0
  }
]
```

GET '/chats/friends/:friendid'

Retrieve a friend chat from the friend's id.

Request Headers:

Authorization: Bearer

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6Im5hbm5pliwbmFtZSI6Imdpb3Zhbm5pliwic3VybmFtZSI6Im1hbmVudGUlLCJtYWlsIjoiz2lvbm1hbkbNbnWFpbC5jb20iLCJyb2xlljoiVVNFUilslmlkljoiNjl5NDlkNGMwMWlyOGUxMjg5YTE1ZDJkIiwidGVtcG9yYXJ5IjpmYWxzZSwiaWF0IjoxNjU0MDA3MDcwLCJleHAiOjE2NTQwMjUwNzB9.i2XwFp140Ar7ZFER7w3ArJzLjRroWJ6fPQl8Ejrc3Xw
```

Request Body:

none

Response Headers:

Content-type: application/json

Response Body:

```
{
  "_id": "6293c57b1bb6639f9328c29f",
  "participants": [
    "6293bd929f3a4c036633adf2",
    "6293bde29f3a4c036633adf8"
  ],
```

}

POST '/messages'

message was sent.

Request Headers:

Content-type: application/json

Authorization: Bearer

p140Ar7ZFER7w3ArJzLjRroWJ6fPQl8Ejrc3Xw

Request Body:

 $\{$

Response Headers:

Content-type: application/json

Response Body:

 $\{$

```
"_id": "6293c57e1bb6639f9328c2a5",
"owner": "6293bd929f3a4c036633adf2",
"owner_username": "simo",
"content": "vediamo",
"visibility": true,
"createdAt": "2022-05-29T19:11:58.434Z",
"updatedAt": "2022-05-29T19:11:58.434Z",
```

```
    "__v": 0
  }
```

GET '/messages'

Retrieve a list of messages.

URI parameters:

- visibility (required): set to false only if you want to retrieve observers' messages
- ids (required): list of the messages' ids you want to retrieve

Request Headers:

Authorization: Bearer

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6Im5hbm5pliwbmFtZSI6Ikdpb3Zhbm5pliwic3VybmFtZSI6Im1hbmVudGUlLCJtYWlsIjoiz2lvbm1hbkBnbWFpbC5jb20iLCJyb2xlIjoivVNFUilsImkljoiNjI5NDIkdGMwMWlyOGUxMjg5YTE1ZDJKliwidGVtcG9yYXJ5IjpmYWxzZSwiaWF0IjoxNjU0MDA3MDcwLCJleHAiOjE2NTQwMjUwNzB9.i2XwFp140Ar7ZFER7w3ArJzLjRroWJ6fPQI8Ejrc3Xw
```

Request Body:

none

Response Headers:

Content-type: application/json

Response Body:

```
[
  {
    "_id": "6293c57e1bb6639f9328c2a5",
    "owner": "6293bd929f3a4c036633adf2",
    "owner_username": "simo",
    "content": "vediamo",
    "visibility": true,
    "createdAt": "2022-05-29T19:11:58.434Z",
    "updatedAt": "2022-05-29T19:11:58.434Z",
    "__v": 0
  }
]
```

POST '/matchmaking/queue'

Add a user to the matchmaking queue. The corresponding socket.io events will be emitted. Matchmaking is based on the user's stats, so the user will be matched against a player that has the number of matches played and wins within the `match_range` and `win_range` fields of the body.

Request Headers:

Content-type: application/json

Authorization: Bearer

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6Im5hbm5pliwbmFtZSI6Ikdpb3ZhbW5pliwic3VybmFtZSI6Im1hbmVudGUlLCJtYWlsIjoiz2lvLm1hbkbNnbWFpbC5jb20iLCJyb2xlljoivVNFUilslmlkljoijNj15NDIkNGMwMWlyOGUxMjg5YTE1ZDJkIiwidGVtcG9yYXJ5IjpmYWxzZSwiaWF0IjoxNjU0MDA3MDcwLCJleHAiOiE2NTQwMjUwNzB9.12XwFp140Ar7ZFER7w3ArJzLjRroWJ6fPQI8Ejrc3Xw

Request Body:

```
{
    match_range: 25,
    win_range: 25
}
```

Response Headers:

Content-type: application/json

Response Body:

```
{
  error: false,
  errorMessage: "",
  id: "6294ba20514de672ab8d70d6"
}
```

DELETE '/matchmaking/queue'

Remove a user from the matchmaking queue.

Request Headers:

Authorization: Bearer

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6Im5hbm5pliwbmFtZSI6Ikdpb3ZhbW5pliwiw3VybmFtZSI6Im1hbmVudGUiLCJtYWlsIjoizI2IvLm1hbkBnbWFpbC5jb20iLCJyb2xlljoivVNFUilslmlkljoiNj15NDIkNGMwMWlyOGUxMjg5YTE1ZDJkIiwidGVtcG9yYXJ5IjpmYWxzZSwiaWF0IjoxNjU0MDA3MDcwLCJleHAiOiJlE2NTQwMjUwNzB9.i2XwFp140Ar7ZFER7w3ArJzLjRroWJ6fPQI8Ejrc3Xw

Request Body:

none

Response Headers:

Content-type: application/json

Response Body:

```
{
  status: "removed player from queue"
}
```

GET '/notifications'

Retrieve the logged in user's notifications.

Request Headers:

Authorization: Bearer

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6Im5hbm5pliwbmFtZSI6Imdpb3ZhbW5pliwic3VybmFtZSI6Im1hbmVudGUlLCJtYWlsIjoiZ2lvLm1hbkBnbWFpbC5jb20iLCJyb2xlljoiVVNFUiIsImkljoiNjI5NDIkdGMwMWlyOGUxMjg5YTE1ZDJKliwidGVtcG9yYXJ5IjpmYWxzZSwiaWF0IjoxNjU0MDA3MDcwLCJleHAiOjE2NTQwMjUwNzB9.i2XwFp140Ar7ZFER7w3ArJzLjRroWJ6fPQI8Ejrc3Xw

Request Body:

none

Response Headers:

Content-type: application/json

Response Body:

```
{
  "_id": "62949d4c01b28e1289a15d30",
  "user": "62949d4c01b28e1289a15d2d",
  "friend_request": false,
  "match_invite": false,
  "friend_request_accepted": false,
  "friend_messages": [],
  "moderator_messages": [],
  "match_alerts": [],
  "__v": 0
}
```



```
}
```

PATCH '/notifications'

Remove a logged in user's notification. (Used when the user reads a notification.)

Request Headers:

Content-type: application/json

Authorization: Bearer

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6Im5hbm5pliwbmFtZSI6Ikdpb3ZhbW5pliwic3VybmFtZSI6Im1hbmVudGUlLCJtYWlsIjoiz2lvbm1hbkBnbWFpbC5jb20iLCJyb2xlIjoivVNFUilsImkljoiNjI5NDIkdGMwMWlyOGUxMjg5YTE1ZDJKliwidGVtcG9yYXJ5IjpmYWxzZSwiaWF0IjoxNjU0MDA3MDcwLCJleHAiOjE2NTQwMjUwNzB9.i2XwFp140Ar7ZFER7w3ArJzLjRroWJ6fPQI8Ejrc3Xw
```

Request Body:

```
{
  friend_request: false,
  match_invite: true,
  friend_request_accepted: false,
  friend_messages: ["6293c57b1bb6639f9328c29f"],
  moderator_messages: [],
  match_alerts: [],
}
```

Response Headers:

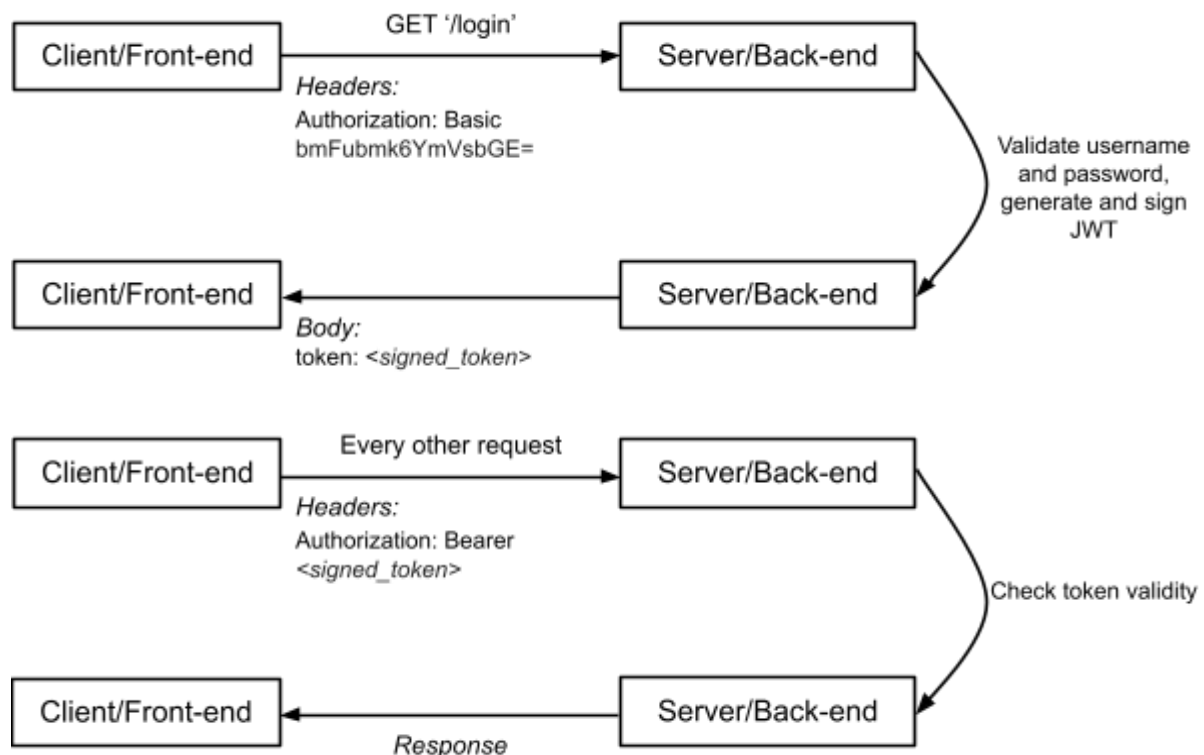
Content-type: application/json

Response Body:

```
{
  "_id": "62949d4c01b28e1289a15d30",
  "user": "62949d4c01b28e1289a15d2d",
  "friend_request": false,
  "match_invite": false,
  "friend_request_accepted": false,
  "friend_messages": [],
  "moderator_messages": [],
  "match_alerts": [],
  "__v": 0
}
```

4. Authentication

Authentication is managed by the back-end through a combination of passport and JWT (JSON Web Token). If a user does not have a valid token he has to log in to the application (authenticate) through the basic access authentication method provided by the HTTP protocol. The back-end uses passport (a JavaScript library) to manage such authorization requests. If the username-password pair is correct then a new JWT token is created, signed and sent back in the response. Once the user has his valid token he has to append it in every other request he makes, only the post request in /user doesn't require a valid token (and the get request in /login). This way no session is created and the back-end doesn't have to keep track of every client. Also, because the token is signed by the back-end it cannot be tampered. Tokens are not encrypted, but they do not contain any sensitive information such as the user's password, so the only possible problem would be if the token got stolen. In this case the token thief would be able to have full access to the user's account, even without knowing the password. Moreover, the basic access authentication isn't encrypted, so the username-password pairs that are sent in the request are visible to anyone watching, though they are encoded in base-64 and the decode process is managed by passport on the back-end. In order to encrypt the basic access authentication request HTTPS must be used. A typical authentication workflow of the application would be the following:



With the token the back-end is able to know which client sent which request. In every other case, invalid token, invalid username and/or password, a 401 (Unauthorized) error is thrown.

5. Front-end

The front-end of the application was developed using Angular. It only uses one module and it consists of fourteen components and seven services. The services are used to communicate with the back-end, and they are the following:

- `user-http.service`: it's the main service and it is used to sign up a new user, login a user, logout a user and manage the JWT token (decode and save the token when necessary).
- `users-http.service`: this service is used to manage requests for the `/users` and `/friends` endpoints, so it manages every request that is related to a user and his friends (aside from login and signup).
- `notification-http.service`: this service is used to load notifications from the back-end when the user logs in and update the back-end when the user reads a notification.
- `message-http.service`: this service is used to retrieve chat messages from the back-end and to add new chat messages to the back-end.
- `match-http.service`: this service is used to manage everything that is related to the matches. It will manage the matchmaking, retrieve a user's matches, retrieve the ongoing matches and manage an actual match itself (add moves, add grids, get a random grid).
- `chat-http.service`: this service is used to retrieve chats from the back-end and create new ones.
- `socketio.service`: this service is used to listen to socket.io events that are emitted by the back-end.

The components, instead, manage the graphical aspect of the application together with the user interaction, and they are the following:

- `signup component`: This component manages the form that allows the user to create a new account. It has its own route inside the application, that is `/signup`.
- `profile component`: This component is used to visualize a user's statistics (number of wins, losses and percentages). It has its own route inside the application, that is `/profile`.
- `play component`: In this component the user can see his ongoing matches and every ongoing match. By clicking on a match he can observe it, if he's not a player, or play in it, if he's a player. He can also start looking for a new match by clicking on the "Find Match" button and as soon as a match is found he will be brought into it. It has its own route inside the application, that is `/play`.

- navbar component: This component is always visible once the user has logged in and it is used to navigate the application and to logout.
- moderator component: This component is only visible to moderators and it contains all of the moderator functions: a form to delete a user, a form to add a new moderator, a search bar to see the stats of any other user and a section to message any other user. It has its own route inside the application, that is /moderator.
- message-editor component: This component is a form used to type and add a new chat message.
- login component: This component is used to login a user. It has the login form and it has its own route inside the application, that is /login.
- friends component: This component is used to manage the social aspect of the application. Here a user can see his friends list, delete friends, chat with friends, invite friends to play, add new friends, accept match invites, accept friend requests and read the messages received from moderators. It has its own route inside the application, that is /friends.
- edit-user component: This component is used when a newly created moderator logs in for the first time. It contains a form that the user has to fill out with his information in order to complete the account creation process. It has its own route inside the application, that is /profile/edit.
- chat component: This component is used to visualize a chat. It's made up of the message-editor component and a list of the messages of that chat. Every chat in the application uses this component.
- game component: This is the main game component that handles every phase of a game: phase one, phase two and observe. According to the user and the phase of the game either the phase one, the phase two or the game observe component is shown, together with the game chat if necessary (using the chat component). It has its own route inside the application, that is /play/match.
- game-phase-one component: This component handles the first phase of the game, where each player has to position the ships on the grid. Each player can either ask for a random grid to be generated or drag and drop the ships on the grid (drag and drop is handled using the interact js library).
- game-phase-two component: This component handles the second phase of the game, where each player takes turns to shoot a position on the grid. Once the game is over a message is shown and the user can only close the game and return to /play.
- game-observe component: This component handles the observing of a game. If a user is not a player he can decide to spectate a game, and so he is taken to this component where he can see every move each player makes and chat with other observers.

6. Typical Workflow

The user opens the application and has to login. If he doesn't have an account he can signup and create a new account, otherwise if he already has a valid JWT token he will be automatically redirected to the /play page.

Sign in

Username

Password

☐ Remember me

[Sign in](#)

New user? [Sign-up](#)

Sign-up

Name

Surname

Email address

We'll never share your email with anyone else.

Username

Password

[Sign up](#)

Already a user? [Login](#)

If the user's account is a newly created moderator account he will be prompted to fill out this form before accessing the application.

Edit User

Name

Surname

Email address

We'll never share your email with anyone else.

Username

Password

Edit

Once logged in the user will be greeted by the following page:

Battleship [Play](#) [Friends](#) [My Stats](#) [Moderator](#) [Logout](#)

My Ongoing Matches

Ongoing Matches

Find Match

riki VS. simo

Forfeit

Battleship [Play](#) [Friends](#) [My Stats](#) [Moderator](#) [Logout](#)

My Ongoing Matches

Ongoing Matches

Find Match

riki VS. simo

Where he will be able to see his ongoing matches, find a new match and observe a match in progress by selecting it from the ongoing matches list. If any of his ongoing matches has had an update (the other player submitted his grid, made his move...) a red dot will be displayed.

If the user clicks on an ongoing match he will be taken to the observe game page where he will be able to see every move each player makes, see the game's chat and chat with other observers:

Battleship

[Play](#)

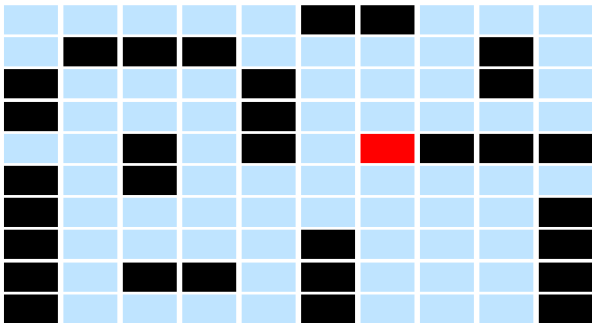
[Friends](#)

[My Stats](#)

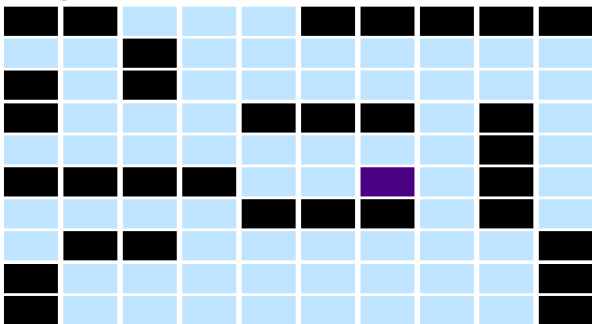
[Moderator](#)

Logout

riki's grid



simo's grid



Light blue: sea, Black: ship, Red: hit, Indigo: miss, Pink: destroyed

simo

02/06/2022 13:10

ciao

riki

02/06/2022 13:10

ciao

mod

02/06/2022 13:10

ciao io sono un osservatore

Send

If the user clicks on the “Friends” link in the navbar he will be taken to the friends section of the application where he will be able to chat with his friends, add and delete friends, invite friends to play and see the messages he received from moderators. By clicking on a friend in the friends list he can see that user’s statistics. If there are any notifications a red dot will be shown, for example if the user receives a new message from a friend a red dot will appear in the chat button, or if the user receives a new match invite a red dot will appear in the match invites tab.

Battleship

[Play](#)[Friends](#)[My Stats](#)[Moderator](#)

Logout

Friends List

Friend Requests

Match Invites

Mods Messages

Username

Add Friend

riki

Chat

Invite to Play

Delete Friend

nanni

Chat

Invite to Play

Delete Friend

Battleship

[Play](#)[Friends](#)[My Stats](#)[Moderator](#)

Logout

Friends List

Friend Requests

Match Invites

Mods Messages

Username

Add Friend

riki

Accept Invite

Reject Invite

Battleship

[Play](#)[Friends](#)[My Stats](#)[Moderator](#)

Logout

Friends List

Friend Requests

Match Invites

Mods Messages

Username

Add Friend

mod

Accept Friend

Reject Friend

Battleship

[Play](#)[Friends](#)[My Stats](#)[Moderator](#)

Logout

Friends List

Friend Requests

Match Invites

Mods Messages

Username

Add Friend

Moderator: mod

simo

29/05/2022 19:11

vediamo

riki

29/05/2022 19:13

bon

riki

02/06/2022 13:12

notifica

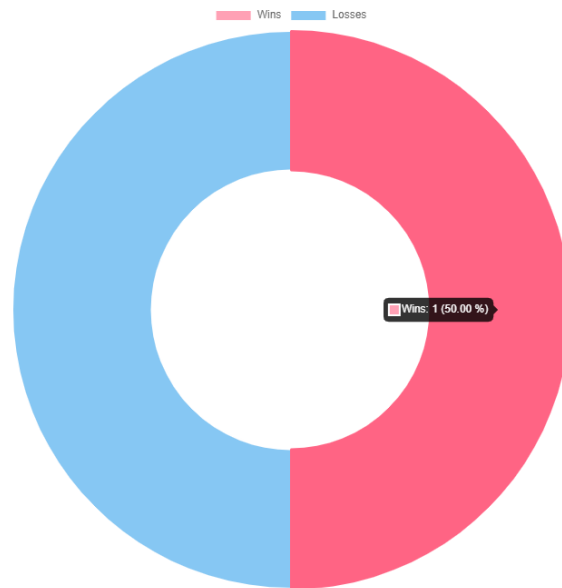
simo

02/06/2022 13:15

ciao riki

Chat with a friend

riki's Stats



A user can also see his own statistics by clicking on the “My Stats” button on the navbar and the same page as for the friends’ statistics will be shown, just with the statistics of the user.

If the user is a moderator he will also be able to see a “Moderator” button in the navbar and by clicking on it he will be able to access all of the moderator’s functions. In the moderator page the user can delete a regular user, add a new moderator, see the statistics of any user and send a message to any user.

Add ModeratorDelete UserMessage User

Username

Enter Username

Password

Password

Add New Moderator

Add ModeratorDelete UserMessage User

Username

Enter Username

Delete User

Battleship
Play
Friends
My Stats
Moderator
Logout

Add Moderator
Delete User
Message User
Username
See User Stats

Username
Message User

Chats

With: mod

With: riki

Either when a match is found (after clicking on the “Find Match” button), when a match invite is accepted or when a user clicks on one of his ongoing matches he will be taken to the page where he can actually play the game. Each game is divided into two phases: in the first phase the two players won’t be able to chat with each other and will have to place all of their boats on the grid. In the second phase, that starts as soon as both players click on the ready button, the two players will take turns to select a position on the grid that they want to shoot. When the game finishes, either by forfeit or because one of the two players won, a message will be shown (both to the observers and to the players) and the user can only exit the game and return to the /play page.

Battleship
Play
Friends
My Stats
Moderator
Logout

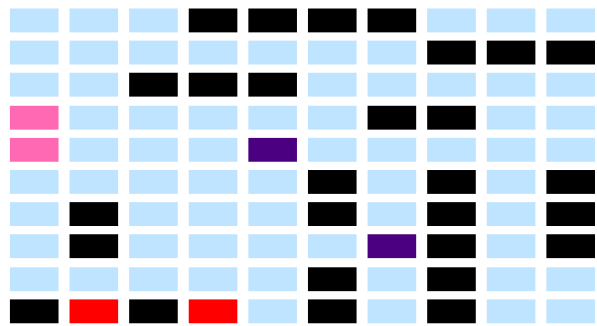
		Destroyer (2)	Destroyer (2)	Cruiser (3)				Carrier (5)	
		Destroyer (2)							
Destroyer (2)									
									Cruiser (3)
					Battleship (4)				
Destroyer (2)									
			Cruiser (3)						
		Battleship (4)							

Drag and drop the boats (the item you drop represents the lower/left end of the boat (depending on whether the boat is vertical or horizontal))
 Right click to change it's direction (red = vertical, blue = horizontal)
 The number between the parenthesis is the length of the boat

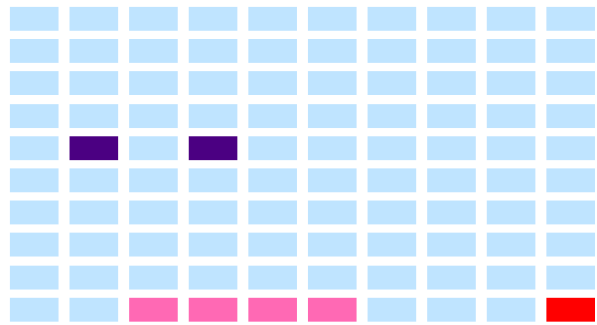
Forfeit
Random
Ready
Reset

Phase one

Your Grid



Opponent's Grid



Light blue: sea, Black: ship, Red: hit, Indigo: miss, Pink: destroyed

Forfeit

Make Move

simo

02/06/2022 13:21

ciao

riki

02/06/2022 13:21

ciao

riki

02/06/2022 13:21

ti batto eh

simo

02/06/2022 13:21

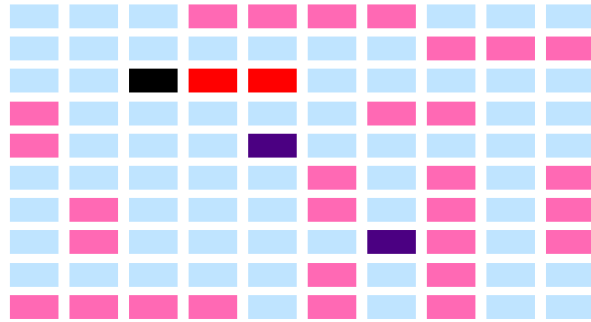
vedremo

Type message...

Send

Phase two

Your Grid



Opponent's Grid



Light blue: sea, Black: ship, Red: hit, Indigo: miss, Pink: destroyed

Forfeit

Make Move

simo

02/06/2022 13:21

ciao

riki

02/06/2022 13:21

ciao

riki

02/06/2022 13:21

ti batto eh

simo

02/06/2022 13:21

vedremo

You have lost

Thank you for playing, we hope you had fun!

Close Game

Type message...

Send

Game finished