



# ForzaCaste documentation

---

*Rosace Gabriel - 880894*

---

## [Introduction](#)

[Architecture](#)

[Development](#)

[How to run the application](#)

[Using docker](#)

[Not using docker](#)

[ForzaCaste on the Web](#)

## [Data model](#)

[Legend](#)

[User](#)

[Statistics](#)

[Notification](#)

[Match](#)

[Message](#)

## [REST API Documentation](#)

[Endpoint list](#)

[Available socket listener](#)

[Common responses](#)

## [Authentication](#)

[Token structure](#)

[Login phase](#)

[Token refresh](#)

[Socket connection](#)

## [Frontend](#)

[Services](#)

[Toast service](#)

[Socket.io service](#)

[User-http service](#)

[Components](#)

[AllUserComponent](#)

[CpuComponent](#)

[FriendChatComponent](#)

[FriendStatsComponent](#)

[GameComponent](#)

[HomepageComponent](#)

[ModChatComponent](#)

[NewModeratorComponent](#)

[SidebarComponent](#)

[ToastComponent](#)

[UserLoginComponent](#)

[UserLogoutComponent](#)

[UserProfileComponent](#)

[UserSigninComponent](#)

[WatchComponent](#)

[Routing](#)

[Additional Features](#)

[Artificial intelligence to play](#)

[How it works](#)

[Use minimax to suggest the move](#)

[Matchmaking algorithm](#)

[Ranking assignment](#)

[Match players](#)

[Block friend](#)

[Typical application workflow](#)

[User](#)

[Login](#)

[Sign up](#)

[Profile](#)

[Send message to moderator](#)

[Moderator](#)

[Add new moderator](#)

[New moderator first login](#)

[View all user subscribed](#)

[Friend](#)

[View friendlist](#)

[Add new friend](#)

[Accept or refuse friend request](#)

[Friendship made](#)

[Chat with friend](#)

[New message arrived](#)

[Friend statistics](#)

[Homepage](#)

[Moderator homepage](#)

[User homepage](#)

[Game](#)

[Invite friend](#)

[Accept game request](#)

[Play your turn](#)

[Wait opponent](#)  
[Game chat](#)  
[Winning message](#)  
[Defeat Message](#)  
[Play against CPU](#)  
[Choose difficulty](#)  
[Play your turn](#)  
[Asking for suggestion](#)  
[Available](#)  
[Not available](#)  
[Observer](#)  
[Watch a game](#)  
[Observer chat](#)  
[Winner message](#)  
[References](#)

---

## Introduction

ForzaCaste is a web application that allows users to play [Connect Four](#) game.

The application allows to make friend with other player and it is possible to chat with them.

There are 3 game modes available:

- Play against a random player chosen by an algorithm to make the game balanced
- Play against a friend
- Play against a CPU, that could be set with 3 different level of difficulty

## Architecture

The application is developed using [MEAN](#) stack.

The frontend is implemented as a Single Page Application ([SPA](#)) using [Angular](#) framework.

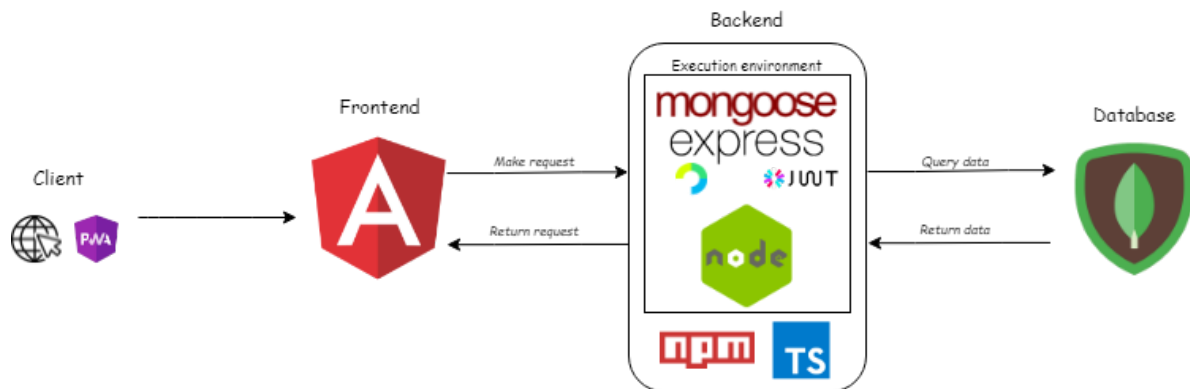
The backend is written using [Typescript](#) and running on [Node.js](#)(version 12.22). It is based on [REST-style](#). It uses [Express](#) as framework to offers an API that contains the application logic.

Backend and Frontend communicate via HTTP protocol and [Socket.io](#), a library that enables real-time, bidirectional and event-based communication between the client

and the server.

The application uses [npm](#) to install and manage version of package used to develop it.

ForzaCaste uses [MongoDB](#), a non relational database document oriented, as persistent data storage. To perform query and model document the application uses [Mongoose](#) driver to interact with the DB.



The image represents the schematic of the architecture and shows how the components interact with each other

## Development

ForzaCaste is developed using [git](#) to track file changes and coordinate work among developers and [Github](#) to host them.

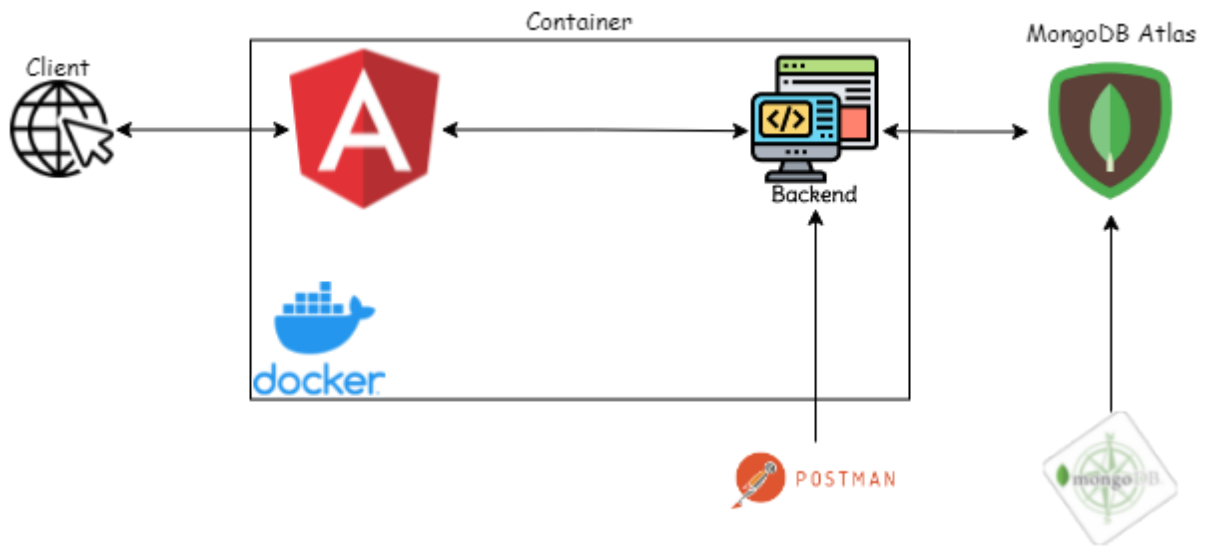
A [docker](#) container was used during the development phase to containerize Angular and the Node server.

The container has been created to expose port 4200, that is used from the `ng serve` daemon that compiles and executes Angular, and the port 8080 used by the web server. A volume was used to maintain data persistence inside the container.

The database is hosted on [MongoDB Atlas](#), the official platform for hosting MongoDB cluster, it allows you to manage the data entered by making queries for creating, searching and deleting document.

Developer team used tool as [Postman](#) to simplifies each step of the API testing and debugging, and [MongoDB Compass](#) an interactive tool to browse and manage our data stored in MongoDB cluster.

During the development of the backend, the team has set the Typescript compiler in watch mode so that it was listening for any changes in the `.ts` files and used the `nodemon` daemon to restart the web server at each change. This allowed to optimize the workflow, avoiding to forget to restart the service at every change.



The image represents the schematic of the architecture used in development and shows how the components interact with each other

## How to run the application

Backend is exposed on port 8080, while frontend is exposed on port 4200 of *localhost*

*you need to create a .env file into backend root directory to initialize the key used to sign the JWT, with this structure*

```
JWT_SECRET=secret
```

## Using docker

### Requirements:

- Docker installed
- Docker-compose installed
- Bash or CMD
- Git (optional)

After cloning or downloading the repository, locate into the application root folder with a terminal and execute `make run` (this can only be done with a Unix-based OS, if you are on Win please run `docker-compose run --service-ports web` ) this, after compiling the docker image, will prompt you into the container in which we can run our services.

**To run backend** → Navigate to `/home/node/app/Backend` and then execute `npm run compile` to compile source code written in Typescript, later execute `npm run start` to start the backend service

**To run frontend** → Navigate to `/home/node/app/taw` and then execute `npm run start` to start the angular frontend

To terminate the container you can run `make stop` after exiting the bash process.

*If some error occurs try downloading npm dependences with `npm install`*

## Not using docker

### Requirements:

- Node.js version 12.22 installed
- Npm installed
- Typescript compiler installed
- Angular-cli installed

After cloning or downloading the repository, locate into the application root folder with a terminal

**To run backend** → Navigate to `Backend` and then execute `npm run compile` to compile source code written in Typescript, later execute `npm run start` to start the backend service

**To run frontend** → Navigate to `taw` and then execute `npm run start` to start the Angular frontend

*If some error occurs try downloading npm dependences with `npm install`*

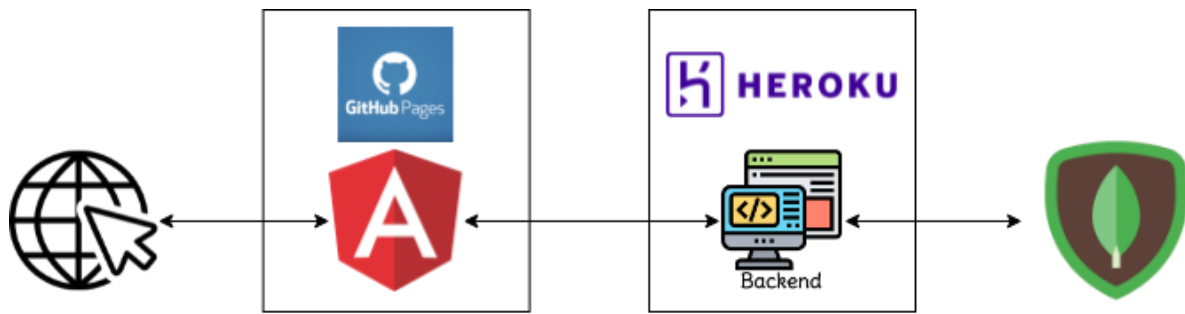
## ForzaCaste on the Web

ForzaCaste is available at <http://forzacaste.live/>

The web frontend has been hosted on [github pages](#), is served under a custom domain obtained on [name.com](#), an online domain register.

The web backend has been hostend on [heroku](#).

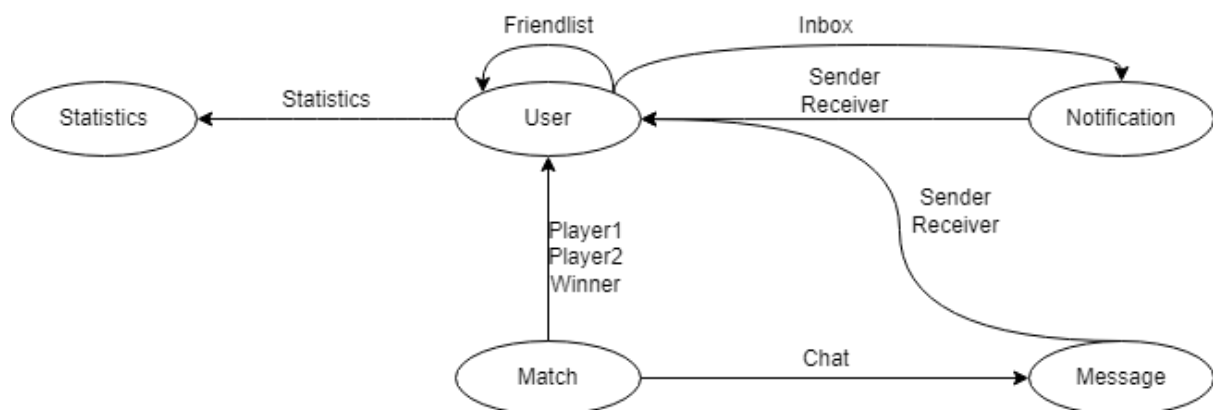
*It is possible to download ForzaCaste PWA.*



## Data model

To implement the application logic, we made use of 5 entities, which are:

- User
- Statistics
- Match
- Notification
- Message



Here is a graphical representation that illustrates the interactions between documents

*For all objects with deletion, it was decided to implement the logical deletion policy to keep track of all information.*

We used [ajv validator](#) to validate objects with more complex structure like match, message, notification and statistics.

## Legend

The following symbols are used in the images below:

- `?` → Encodes that the information is optional
- `[]` → Encodes that the type of the attribute is an array
- `{}` → Encodes that the type of the attribute is an object
- `( a:type, b:type ) => type` → Encodes a method that accept a,b as parameters and return a value of the specified type (Typically the name of the method is yellow)

## User

User document that allows you to store all the information necessary for the operation of the application, we decided to identify the user by the username he chooses during registration, so we applied a constraint on the uniqueness of this attribute.

*Password are not stored as plain text, but are stored in obfuscated form after being encrypted using the **bcrypt** library.*

```

1 export interface User extends mongoose.Document {
2   username: string,
3   name?: string,
4   surname?: string,
5   mail?: string,
6   avatarImgURL?: string,
7   roles: string,
8   inbox?: Notification[],
9   statistics?: Statistics,
10  friendList: [{username: string, isBlocked: boolean}],
11  salt?: string, // salt is a random string that will be mixed with the actual password before hashing
12  digest?: string, // this is the hashed password (digest of the password)
13  deleted?: boolean,
14  setPassword: (pwd: string) => void,
15  validatePassword: (pwd: string) => boolean,
16  hasModeratorRole: () => boolean,
17  setModerator: () => void,
18  hasNonRegisteredModRole: () => boolean,
19  setNonRegisteredMod: () => void,
20  hasUserRole: () => boolean,
21  setUser: () => void,
22  addFriend: (username: string, isBlocked: boolean) => void,
23  isFriend: (username: string) => boolean,
24  addNotification: (notId: Notification) => void,
25  deleteFriend: (username: string) => void,
26  setIsBlocked: (username: string, isBlocked: boolean) => void,
27
28  //User management
29  deleteUser: () => void
30 }

```

Representation of the information that describes a user and the methods used to manage this information

*For ease of management and extensibility it was decided to save the images via URL, as we wanted to avoid an upload/download to the database or web server.*



## Statistics

Statistics document that allows you to store information about the games played by a user, it saves the key information to allow a balanced game during the random mode, such as number of games won, lost and played. We also wanted to give relevance to the moves made as they are essential to understand the skill of a player.

```
1 export interface Statistics extends mongoose.Document {  
2   nGamesWon: number,  
3   nGamesLost: number,  
4   nGamesPlayed: number,  
5   nTotalMoves: number,  
6   ranking: number,  
7   getGamesDrawn: () => number,  
8 }
```

## Notification

Notification document that allows you to store information about notification that one user can send to another.

The notification that can be sent are:

- FriendRequest → If other user accepts, friendship is made between two.
- RandomMatchmaking → It is not targeted at an actual user, but allows you to alert others that there is someone who would like to play a random match. When the notification type is this, then the ranking is set to look for a balanced match. Otherwise ranking is null.
- FriendlyMatchmaking → If other user accepts, game starts between two.

We have provided a mechanism to indicate whether a notification has been read or not, this is done through the inpending attribute.

*After the receiving user has interacted with the notification it is deleted.*

```

1 export interface Notification extends mongoose.Document {
2   _id: mongoose.Types.ObjectId,
3   type: string,
4   text?: string,
5   sender: string,
6   receiver?: string,
7   deleted: boolean,
8   impending: boolean, //It's used to show if a request has already been displayed
9   ranking?: number,
10  isFriendRequest: () => boolean,
11  isNotification: () => boolean
12 }

```

## Match

Match document that allows you to store information about match, for each match there is a chat between the two players.

*After the game is over or one of the two exits the game it is cancelled, acting on the inprogress attribute.*

```

1 export interface Match {
2   inProgress: boolean,
3   player1: string,
4   player2: string,
5   winner: string,
6   playground: Array<any>[6][7],
7   chat: message.Message[],
8   nTurns: number
9 }

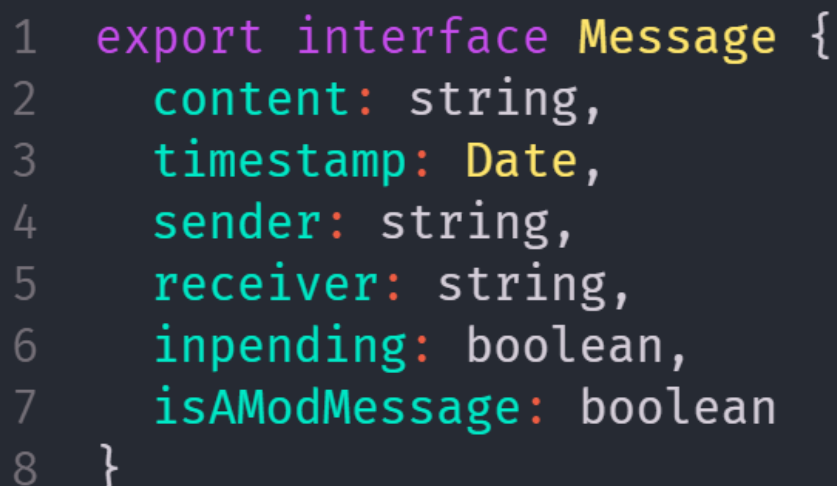
```

## Message

Message document that allows you to store information about message that two player exchange.

**Messages are allowed only between friends or to moderators for information or reports.**

We have provided a mechanism to indicate whether a message has been read or not, this is done through the inpending attribute.



```
1  export interface Message {  
2    content: string,  
3    timestamp: Date,  
4    sender: string,  
5    receiver: string,  
6    inpending: boolean,  
7    isAModMessage: boolean  
8  }
```

## REST API Documentation

*If you want to deepen the documentation of the endpoints you can do it by consulting the file [endpoint\\_documentation.pdf](#) available in the project folder.*

*It was generated by the tool built into Postman, which allows you to enter a description for each endpoint defined in the collection and document it using examples.*

## Endpoint list

The backend provides an API to interface with the application logic, it has been developed following the REST API guidelines.

It provides the following endpoints

Endpoint	Method	Attributes	Description
/	GET		Returns the version and a list of available endpoints
/login	GET		Login an existing user, returning a JWT
/whoami	GET		Get user information and refresh the JWT
/users/:username	GET		Return a user that has username specified
/users	GET		Return a list of available users
/users/online	GET		Return a list of online players at this moment
/users	POST		Sign up a new user
/users/mod	POST		Create a new moderator, only moderator can do it
/users	PUT		Update user information
/users/:username	DELETE		Deletion of standard players from moderators
/rankingstory	GET		Return a list of ranking that logged user has at the time of game requests
/rankingstory/:username	GET		Return a list of ranking that username has at the time of game requests
/game	GET		Returns a list of game in progress
/game	POST		Create a random or friendly match. Furthermore a user can enter in a game as observer

Endpoint	Method	Attributes	Description
/game/cpu	POST		Create a match against CPU. Furthermore a user can enter in a game as observer
/game	DELETE		Used by a player in order to delete a started game or to delete a game request
/game	PUT		Accept a friendly game request
/move	POST		Play the turn making a move, it contains the game logic and the event notifier
/move/cpu	POST		Play the turn vs AI, it contains the game logic and call minmax algorithm to play the AI turn
/move	GET		Ask AI what is the best move, returns the best column in which insert the disk
/notification	POST		Create a new friend request
/notification	GET		Return all the notification of the specified user. This endpoint returns all the notification that are received and that are not read
/notification	GET	?inpending=bool	Return all the notification of the specified user. This endpoint returns all the notification that are not read
/notification	GET	?makeNotificationRead=bool	Return all the notification of the specified user. This endpoint mark all the notification as read

Endpoint	Method	Attributes	Description
/notification	PUT		Change the status of the notification, so the indicated notification will appear as read
/message	GET		Returns all messages and all messages in pending
/message	GET	?modMessage=bool	Returns all moderator messages and all moderator messages in pending
/message	POST		Send a private message to a specific user
/message/mod	POST		Send a private moderator message to a specific user
/message	PUT		Update a specific message and marks it as read
/gameMessage	POST		Send a message in the game chat
/friend	GET		Return the friendlist of the current logged user
/friend/:username	DELETE		Deletion of a friend in the friendlist of the current logged user
/friend	PUT		Change the attribute isBlocked of the specified user in the friendlist

*By bool we mean the data type, not the value.*

## Available socket listener

The application provides the following events to listen for so that the server can inform the client of all interactions that are taking place, this allows for asynchronous event-based programming.

**Communication via HTTP requests was preferred.**

In fact every interaction that starts from the client is based on this protocol, but to make everything more dynamic the client is listening for these events and when it receives them it will never interact using the socket ( `client.emit()` ) but making an HTTP request, so the server is not listening for any event except the connection and disconnection.

Event	Body	Description
gameReady	{ 'gameReady': true, 'opponentPlayer': "" }	If there is another user that want to play a match against a random player
move	{ 'yourTurn': bool }	Informs the user if it's his turn ( <code>true</code> ) or if it's opponent turn ( <code>false</code> )
move	{ "error": bool, "codeError": x, "errorMessage": "..." }	Informs the user who insert a move if it is valid or not.
move	{ move: index }	Opponent player receive this which represents the move played by other player in which <i>index</i> is the chosen column
gameStatus	{playerTurn: ""}	Informs the observers which player will do the first move
gameStatus	{ player: username, move: index, nextTurn: otherPlayer }	Match observer receive this which represents the player who inserted disk into column index and the player who has to play next turn
gameRequest	{type: "friendlyGame", player: "username"}	The friend who receive the friendly game request receive this in which username contains the username of the user that send the request
enterGameWatchMode	{ 'playerTurn': "", 'playground': "" }	Informs the observator of the status of the game, in particular the username of the player that will do the next move and the status of the playground at the moment
result	{'winner': "", 'message': "Opposite player have left the game"}	Informs the other player that the opponent have left the game
result	{ "winner": bool null }	Informs players about the result of the game
result	{ "rank": rank }	Informs players about the ranking obtained

Event	Body	Description
result	{ winner: 'username of the player who won'}	Informs observer about the match winner
gameChat	{ '_id' : '', 'content' : '', 'sender' : '', 'receiver' : '', 'timestamp' : { "\$date" : "" }}	All the users involved in the match receive this that correspond to the match document in the database
newNotification	{ sender: '', type: "friendRequest" }	Informs the user that someone (username) send a new friend request
request	{ newFriend: "" }	The creator of the friend request receive this that inform the other user have accepted the friend request
friendDeleted	{ deletedFriend: "" }	The friend of the user receive this that inform the user who can no longer be friend
friendBlocked	{ blocked: bool }	The friend that is blocked by another receive this that inform if he has been blocked
message	{ '_id' : '', 'content' : '', 'sender' : '', 'receiver' : '', 'timestamp' : { "\$date" : "" }, inpending: "", isAModMessage: true }	The receiver of the message receive this that correspond to the message document in the database
online	{ username: username, isConnected: bool }	Informs listeners that username is now online( <code>isConnected=true</code> ) or is now offline( <code>isConnected=false</code> )
updateUser	user object just updated into database	Informs listeners that a user has updated their information

For more details see the file [endpoint\\_documentation.pdf](#) available in the project folder.

## Common responses

Here is a brief description of the possible responses that can be returned as a result of a request.

For more details see the file [endpoint\\_documentation.pdf](#) available in the project folder.

Status Code	Error Message
200	OK - The request has succeeded
400	BAD REQUEST - The server cannot or will not process the request due to something that is perceived to be a client error

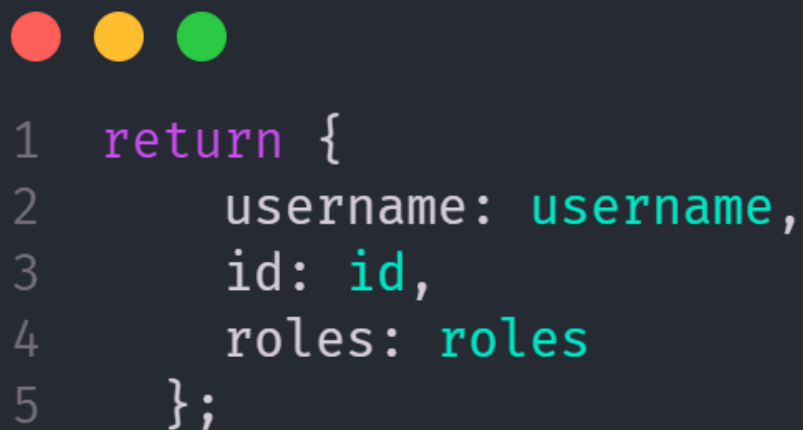


StatusCode	Error Message
401	UNAUTHORIZED - The request has not been applied because it lacks valid authentication credentials for the target resource
403	FORBIDDEN - The server understood the request but refuses to authorize it
404	NOT FOUND - The origin server did not find a current representation for the target resource or is not willing to disclose that one exists
502	BAD GATEWAY - The server, while acting as a gateway or proxy, received an invalid response from an inbound server it accessed while attempting to fulfill the request

# Authentication

## Token structure

It was decided to save in the jwt only the really useful information, such as the username and the role so as to use the token in case you need this information to make requests to the API.



```
1  return {  
2      username: username,  
3      id: id,  
4      roles: roles  
5  };
```

This image represents the structure of the JWT with the fields useful to identify the user

## Login phase

Authentication is based on JWT token to get it you need to login.

On the backend, the login phase is managed by [passport.js](#), an authentication middleware that takes care of defining the actions to be carried out during the Basic Authentication phase.

The actions that must be performed during login are the following; once it is ascertained that the client wants to authenticate itself through Basic Authentication it is necessary to check the credentials and see if they match with those saved in the database(password verification should be done through the [bcrypt](#) library because it is the library that takes care of obfuscating the password by encrypting it).

If this research is affirmative then it is necessary to build the token with the information of the user that wants to log in, sign it (the [jsonwebtoken](#) library takes care of this, it allows through a secret key, stored in the server environment variables, to set a expiration time to JWT, to sign it and verify that it is usable) and finally deliver it to the client.

If the authentication failed then an error is returned to the client.

*We have chosen to set the expiration time to 1 hour.*

*We know that using the HTTP protocol the delivery of the token to the client is a critical step because if there was an attack of type MITM could steal the token and authenticate impersonating the user. **This problem might be mitigated using HTTPS.***



## Token refresh

Since an expiration time was set to the JWT it was necessary to provide a method to update the token's expiration date.

To do this we introduced an endpoint that checks that the token is not close to expiration (our limit is under 5 minutes), if it is close to expire then we create a new

token (containing the information in the 'old' token) and return it to the client, otherwise we return the information of the user who made the request.



## Socket connection

After the login it is necessary to connect to the socket, to do this it will be necessary to send in the parameters of the connection request the jwt.

It will allow to authenticate the client and maintain server side the mapping between user and socket.

Then when the client makes a request to disconnect, the correspondence with the previously associated socket will be deleted.

## Frontend

The frontend is located inside the folder `taw`, it is organized so that each graphical component is inside a folder that is located in the space reserved for sources (`src`), even the services and the router component are located in the `src` folder, but they do not have their own folder (except for the service for the management of toast).

To design the graphical interfaces we used the components that [bootstrap5](#) offers.

Using bootstrap it was possible to make our application responsive, so that it would natively adapt to all existing devices without having to change the styles too much.

In addition we used [ng2-chart](#) for chart creation, [ng-bootstrap](#) for toast management and [Angular Material](#) for badges. For the icons we decided to rely on the default bootstrap and Angular Material icons.

## Services

To implement our frontend logic, we used 3 services:

- `Toast.service` (located into `_services` folder)

- Socketio.service
- User-http.service

## Toast service

Toast service is used to display toast that update the user about the events on which he is listening.

Toasts are used to notify something to the user, for example when they receive a message or a request to play a game.

This service defines two methods, that are used by the toast component, that are:

- `show(...)` → Used to make a toast appear
- `remove(...)` → Used to make a toast disappear

## Socket.io service

This service is used to manage socket.io event.

It allows transforming socket events into observables that components can subscribe to in order to get notified data.

Inside this service there are all the methods for managing the socket and all those directives that serve for the management of the received data.

## User-http service

This service is used to interact with the server performing HTTP request.

It contains all the user information and the interactions that they can make with the server.

It takes care of transforming the server's response into observables that interested components can register to in order to obtain data.

The service takes care of managing the authentication(login and logout phase) and saving the token so that the client continues to be recognized by the server, it also takes care of inserting the token inside the HTTP requests that require bearer authentication.

The token can be saved in different spaces that the client offers, such as:

- `localStorage` → Shared space to all browser tabs and persists on closure
- `sessionStorage` → Space allocated only during the session and it is cleaned up each time the page is reloaded

We decided to support both methods, but we preferred to set sessionstorage as default because the use of localStorage, for the way we decided to manage backend sockets, is not always correct because every time the page is reloaded the socket connection is closed and we lose the mapping made between client and server.

## Components

### AllUserComponent

This component allows moderators to view a dashboard to perform actions regarding users of the application.

The actions that can be done are:

- Delete a standard user(So not a moderator)
- Start a chat with any user (It is different from the friend chat, *i.e. you won't see those messages exchanged through the friend chat*)
- Visualize the profile with the relative statistics of each user

### CpuComponent

This component allows users to play against the artificial intelligence, during the game, if the user has a ranking higher than 100 points, it can ask for a suggestion regarding the next move to make.

At the end of the game the player will be given the result.

### FriendChatComponent

This component allows the user to view the chat with one of his friends.

You can read and send messages.

### FriendStatsComponent

This component allows the user to view the profile with its statistics of a friend.

### GameComponent

This component allows you to play games, it is used both during games between strangers and during games between friends.

During the game it is possible to visualize the status, that is who is going to play the turn and the color of the disk assigned to each player.

In addition, there is a chat between the two players, if the two were friends it will be unrelated to the one outside the game.

During the game, if the user has a ranking higher than 100 points, it can ask for a suggestion regarding the next move to make.

At the end of the match both players will be informed about the result of the match and the points they earned.

## **HomepageComponent**

This is the main component, from here the user can decide to start or observe a game.

The available modes are:

- Play against the cpu, in here the user can decide the difficulty of the game(easy, medium, hard)
- Play against a stranger or with a friend
- Watch a game of a stranger or a friend

## **ModChatComponent**

This component allows viewing and sending the messages exchanged between a moderator and a user

## **NewModeratorComponent**

This component allows you to create a new moderator by specifying the username and password

## **SidebarComponent**

This is the fundamental component used for navigation within the app's menus.

It allows you to view the notifications that have arrived.

It also allows access to the list of friends where the user can manage the relationship with the other person.

In this menu you can do the following operations:

- Block a friend
- Delete a friend
- See the friend's statistics

- Send messages to the friend
- Make new friends

## **ToastComponent**

This component is used to define the structure of the toast that will be shown to the user.

## **UserLoginComponent**

This component is used by the user to log in.

It also allows you to redirect the user to the account creation page if they do not have an account.

## **UserLogoutComponent**

This component is used to logout, it after invalidating the token redirects to the login page.

## **UserProfileComponent**

This component is used to give summary information about the logged-in user, such as the statistics obtained in the game and the history of his ranking.

From here it is possible to edit the user's profile.

This component is also used to force non-registered moderators to enter their information and change their password.

## **UserSigninComponent**

This component is used to create a new user, it allows you to enter a random image based on the chosen username.

## **WatchComponent**

This component is similar to GameComponent, unlike the other it allows observers to watch a game in progress and chat with each other, being able to observe the game chat as well.

## **Routing**

Angular, is based on the philosophy of the single page application (SPA) for this need a module to make the mapping between url and component, so you can always render the component required by the client.

This module is the router, below is how the addresses have been mapped to its components

Path	Component
	UserLoginComponent
**	UserLoginComponent
/login	UserLoginComponent
/home	HomepageComponent
/logout	UserLogoutComponent
/signin	UserSigninComponent
/profile	UserProfileComponent
/new-mod	NewModeratorComponent
/all-user	AllUserComponent
/game	GameComponent
/watch	WatchComponent
/cpu	CpuComponent
/user-stats/:friend	FriendStatsComponent
/friend-chat/:friend	FriendChatComponent
/mod-chat/:user	ModChatComponent

*During each route change, the validity of the token is checked, so that it is refreshed if it expires.*

**So the router allows you to move between components without reloading the page and also allows you to check the expiration of the token.**

## Additional Features

We decided to add the following features to the application:

- Implementation of an AI to make the user play against the PC
- Balanced matchmaking based on an algorithm that analyzes statistics and scores players
- Possibility of blocking a friend to stop receiving requests from it
- Deploy the application on the web and download the PWA (see [ForzaCaste on the Web](#) section)



## Artificial intelligence to play

An algorithm has been implemented in the application to allow the CPU to play against the user.

It was developed based on the [Minimax](#) algorithm, a recursive algorithm which is used in decision-making, it provides optimal moves for the CPU, assuming that the opponent is also playing optimally.

### How it works

Algorithm considers two opponent:

- Max playing → It will try to maximize the value
- Min playing → It will choose whatever value is the minimum

It considers the next  $x$  moves, where  $x$  depends on the difficulty:

- 2 → Easy
- 5 → Medium
- 7 → Hard

The algorithm will play the next  $x$  moves and assign a score to each column and at the end it will get the optimal score, which corresponds to the best possible play.

The score is calculated according to the number of connections between the disks that the AI has inserted, if 4 are connected 100 points will be awarded, when 3 are connected 5 points are awarded and when there are 2 connections 2 points are awarded. You lose 4 points if your opponent has managed to connect 3 disks.

### Use minimax to suggest the move

To suggest the move, it was thought to parameterize the color of the diskette so that the algorithm could only be applied once to get the best move the caller could make.

***To take advantage of this feature it is necessary that the user has a ranking higher than 100 points***

## Matchmaking algorithm

### Ranking assignment

The ranking is calculated as the sum of the results obtained during a game(  
 $\text{Ranking} = \sum \text{game points}$ ), we wanted to give different importance to each

game.

As matches between very strong opponents must have more weight than those between weaker players.

In addition, the number of moves made to win affects the points earned.

At the end of each game, the score is calculated as follows:

A matchmaking rating (MMR, which indicates the player's skills) is calculated for each player based on past statistics (games won, games played and the number of total moves made in all games played by that player).

MMR is calculated as follows:

$$\text{winRate} = \frac{\text{nGamesWon}}{\text{nGamesPlayed}}$$

$$\text{avgMoves} = \frac{\text{nTotalMoves}}{\text{nGamesPlayed}}$$

$$\text{MMR} = \text{winRate} + \frac{\text{winRate}}{\text{avgMoves}}$$

Where:

- **nGamesWon** → Represents the number of games the user has won
- **nGamesPlayed** → Represents the number of games the user has played
- **nTotalMoves** → Represents the number of total moves he made in each game he played

Based on this result and the result of the match (which for simplicity is coded as win and not win), you get game points to add to your global ranking.

***If the playes wins:***

MMR	Game points
$x \geq 85$	[60, 70]
$70 \leq x < 85$	[40, 60]
$50 \leq x < 70$	[30, 40]
$30 \leq x < 50$	[20, 30]
$0 \leq x < 30$	[10, 20]

***If the playes not wins:***

MMR	Game points
$x \geq 50$	[-15, -25]
$x < 50$	[-15, -35]

*The Game points is referred to as a range because to make it a little more random, the score is chosen randomly within that range.*

## **Match players**

Matchmaking is handled as follows:

Player 1 is willing to play, then creates a "matchmaking notification" within which he enters his ranking and waits.

Player 2, who is also willing to play, tries to create a new "matchmaking notification", but this is only possible if he cannot be matched with the other players.

In order to be paired with another, the difference between the two rankings must not exceed 80 points. Once the pairing has been made, the game can start.

There is a mechanism to avoid deadlock, because if a player is too strong or too weak he may never play.

In order to do this, it has been thought that every time a new request arrives, the ranking of those that are in progress and have not been matched with any other is reduced by 80 points.

## **Block friend**

It has been added the possibility for the user to block a friend, this allows you to stop receiving interactions from it, such as requests to play a game or messages in chat.

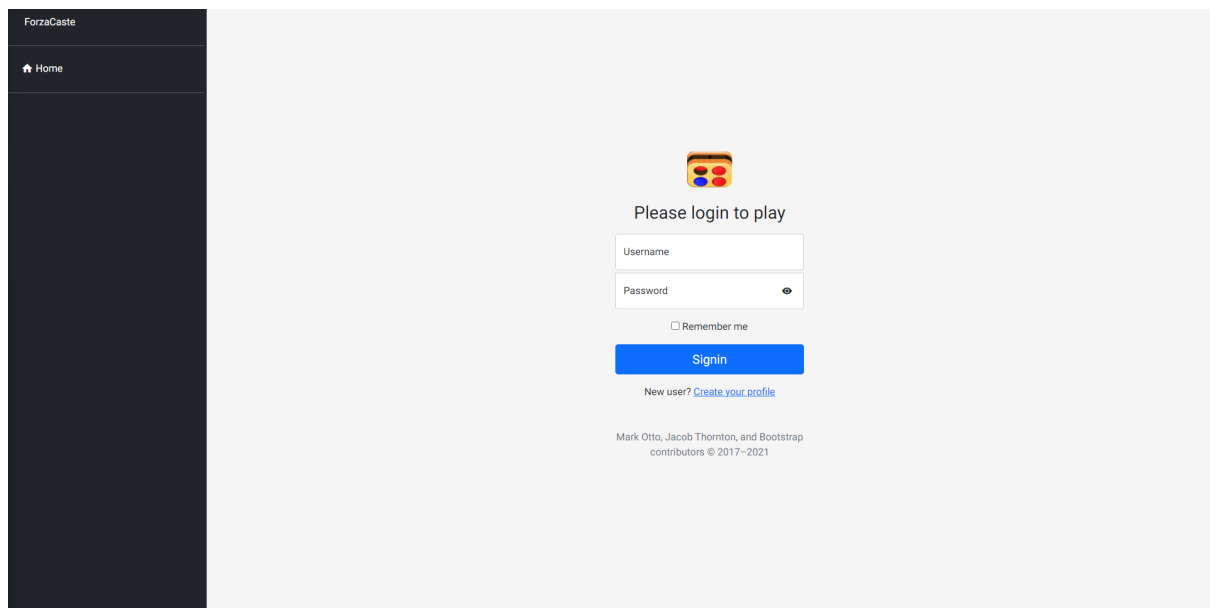
*You can also unlock the friend to return to the initial state*

# **Typical application workflow**

## **User**

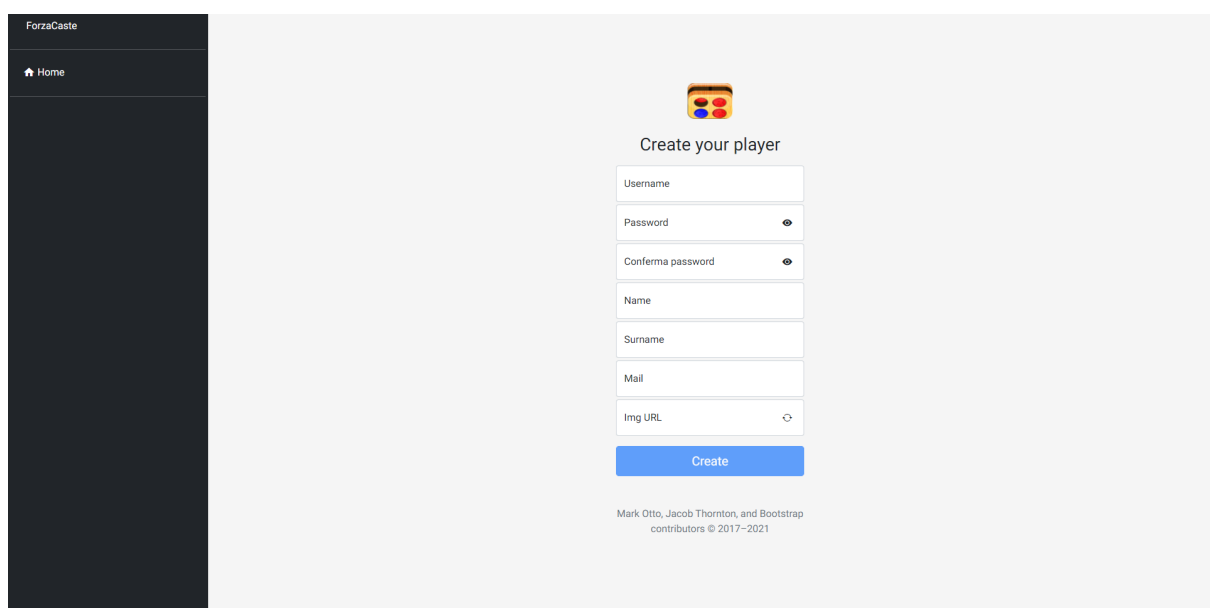
The most important interfaces that the user needs to create, view and edit his profile are shown below.

## **Login**



The user can access the functionality of the application by entering a username and password.

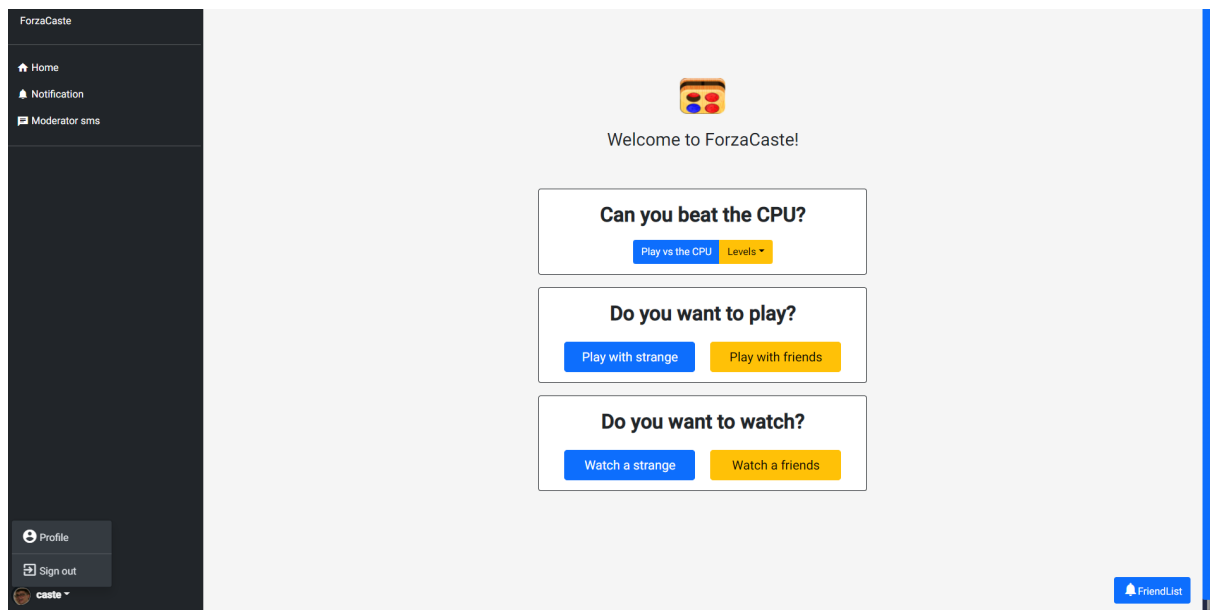
## Sign up



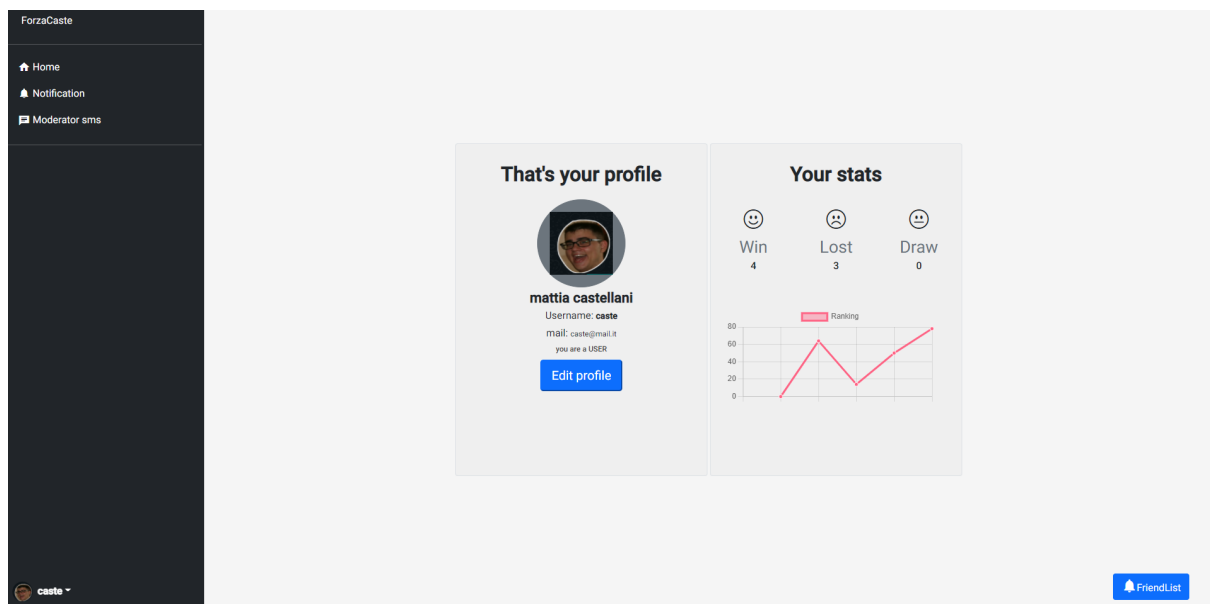
Some user aids are available, such as not blurring the password and checking that the two entered passwords match.

A random image is available based on the chosen username.

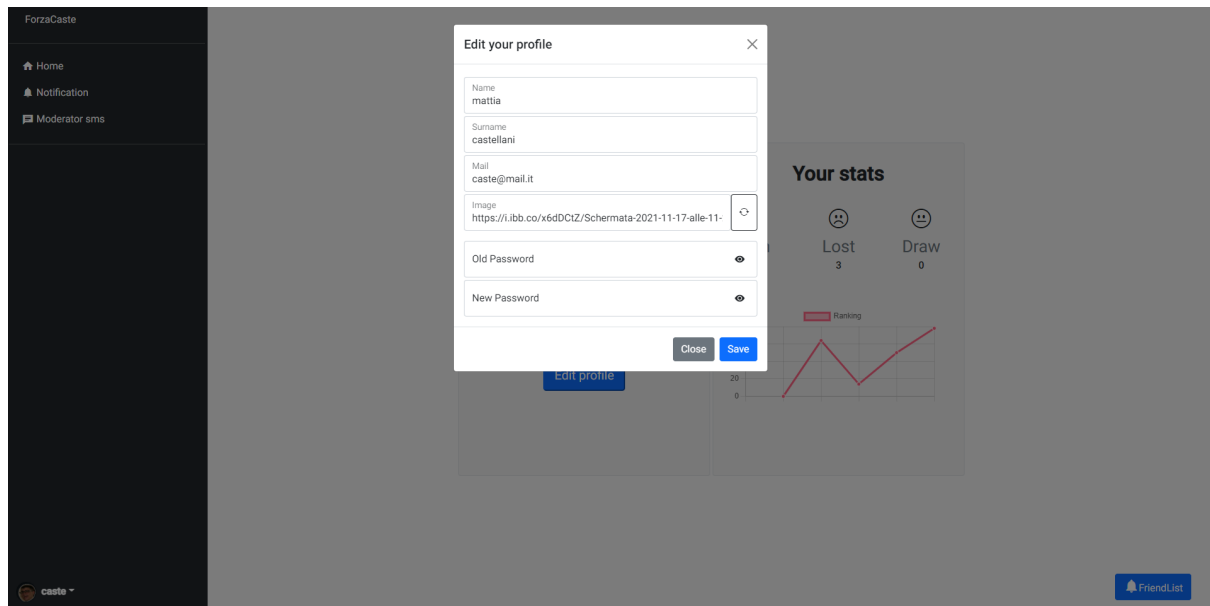
## Profile



From the homepage you can access your profile or log out.

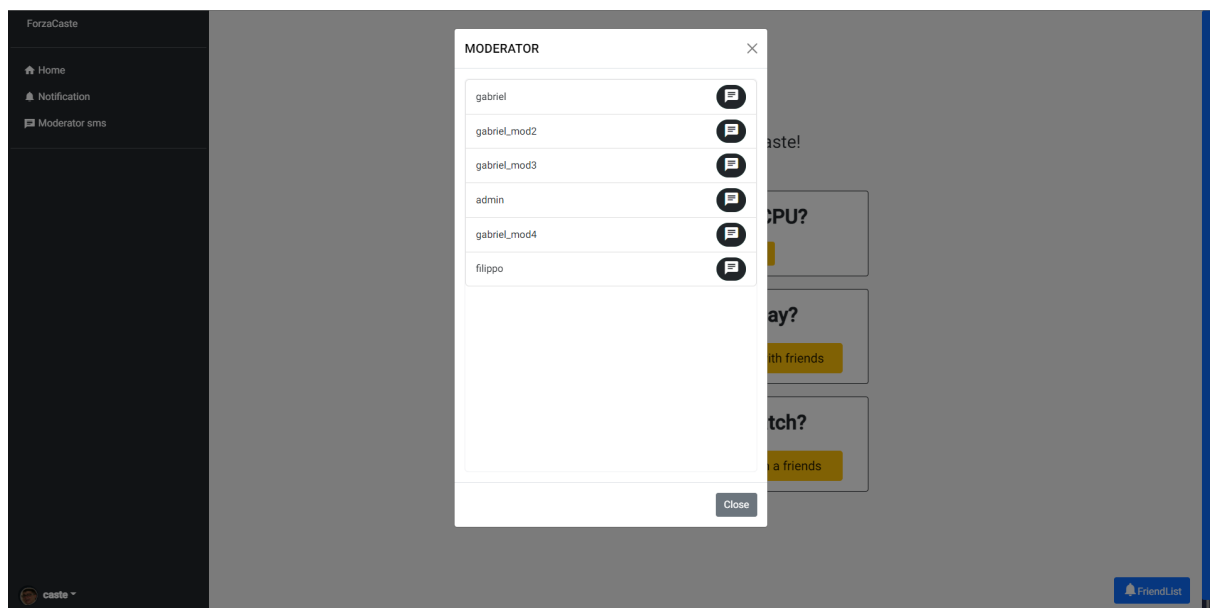


Summary of the user profile, it is possible to see information such as: statistics, with a graph that indicates the trend of the ranking and the personal information entered during the registration phase



From your profile you can access the screen for updating your personal information.

## Send message to moderator



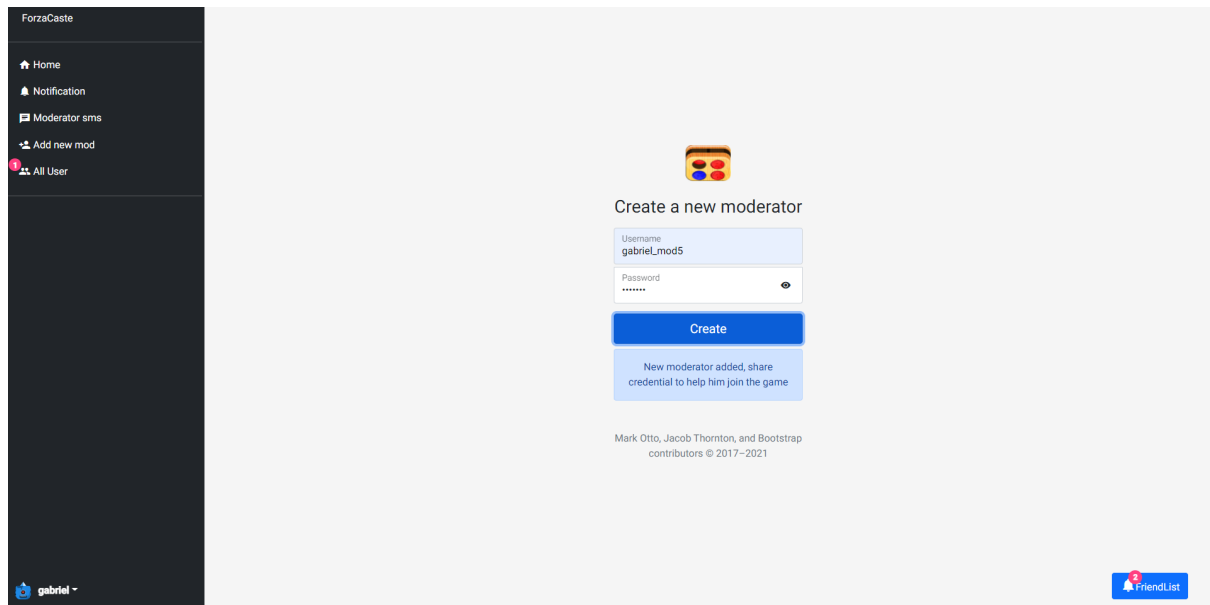
From the sidebar you can access the list of moderators present in the application and send them a message.

*The chat interface that is shown is similar to the others available in the application, so to avoid duplicates we omit this screenshot*

## Moderator

Below are shown the interfaces used by the moderators to manage the application.

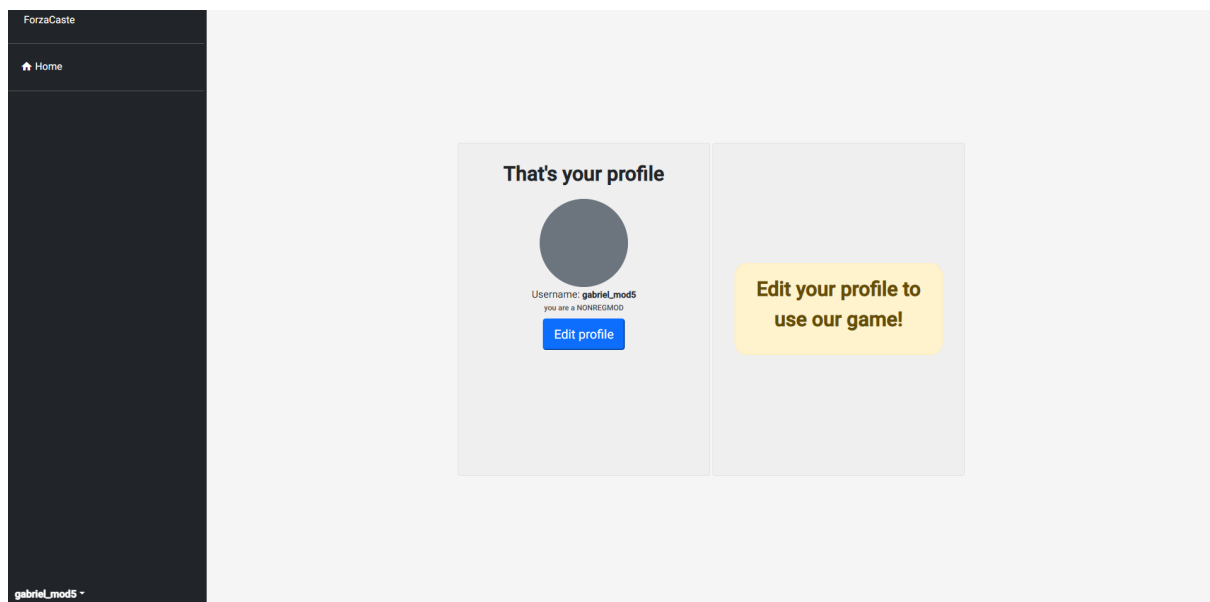
## Add new moderator



The screenshot shows the 'Add new moderator' form in the ForzaCaste application. On the left is a dark sidebar with navigation links: Home, Notification, Moderator sms, Add new mod, and All User (highlighted with a red notification badge). The main content area has a light gray background. At the top center is a game controller icon. Below it, the title 'Create a new moderator' is displayed. The form consists of two input fields: 'Username' with the value 'gabriel\_mod5' and 'Password' with masked characters. A blue 'Create' button is positioned below the fields. A light blue feedback message states: 'New moderator added, share credential to help him join the game'. At the bottom center, small text reads: 'Mark Otto, Jacob Thornton, and Bootstrap contributors © 2017–2021'. In the bottom right corner, there is a 'FriendList' button with a red notification badge. The bottom of the sidebar shows a user profile for 'gabriel'.

From the sidebar you can access the interface for creating a new moderator, in which by specifying username and password you can add him to the users of the application. This action can only be done by moderators.

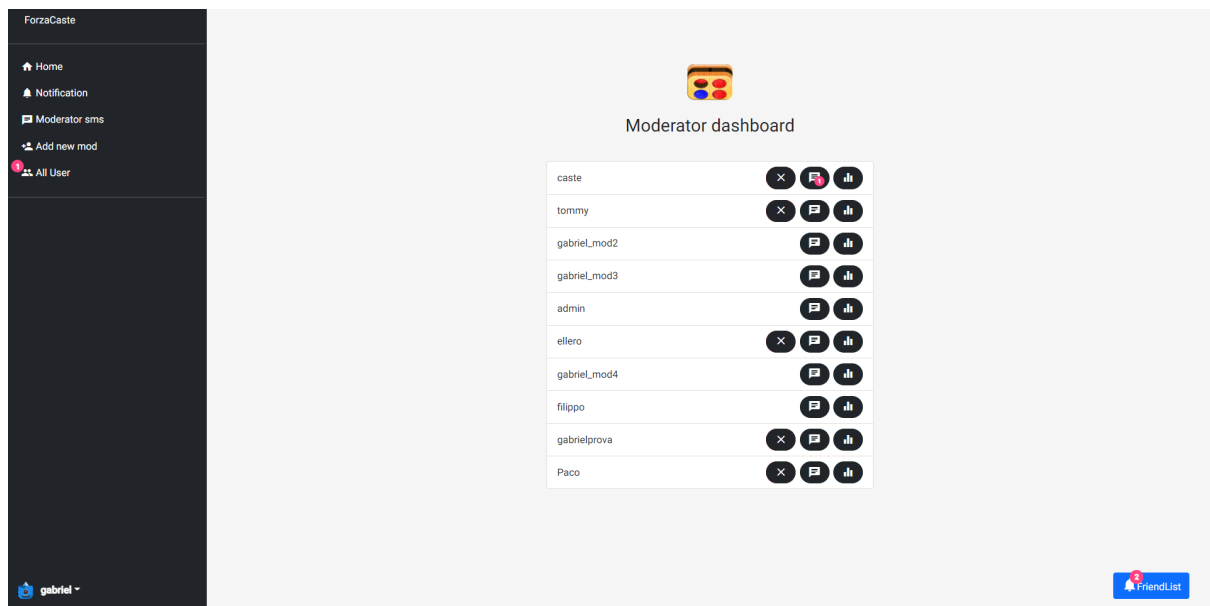
## New moderator first login



The screenshot shows the profile page of a newly added moderator. The sidebar on the left is dark and shows the 'Home' link. The main content area has a light gray background. It features a profile card with the title 'That's your profile', a gray circular placeholder for a profile picture, and the text 'Username: gabriel\_mod5' and 'you are a NONREGMOD'. Below this is a blue 'Edit profile' button. To the right of the profile card is a yellow call-to-action box that says 'Edit your profile to use our game!'. The bottom of the sidebar shows the user profile for 'gabriel\_mod5'.

At the first login the newly added moderator will be redirected to the profile page where he will be required to add his credentials and change his password.

## View all user subscribed



A moderator can access the list of all platform users via the 'alluser' button located in the sidebar. From here it will be possible to messaging with the chosen user, delete him or look at his profile with its statistics.

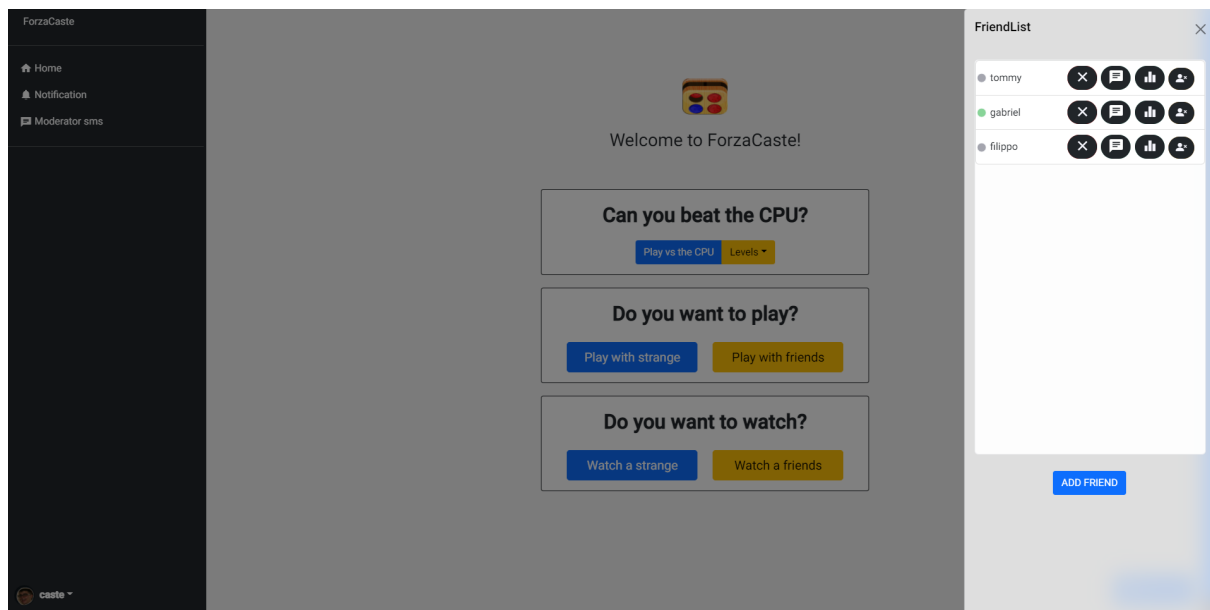
*Also here, to avoid repetition, the chat and profile screens are omitted.*

## Friend

Below are shown the most important interactions that users need to make friends and interact with them.

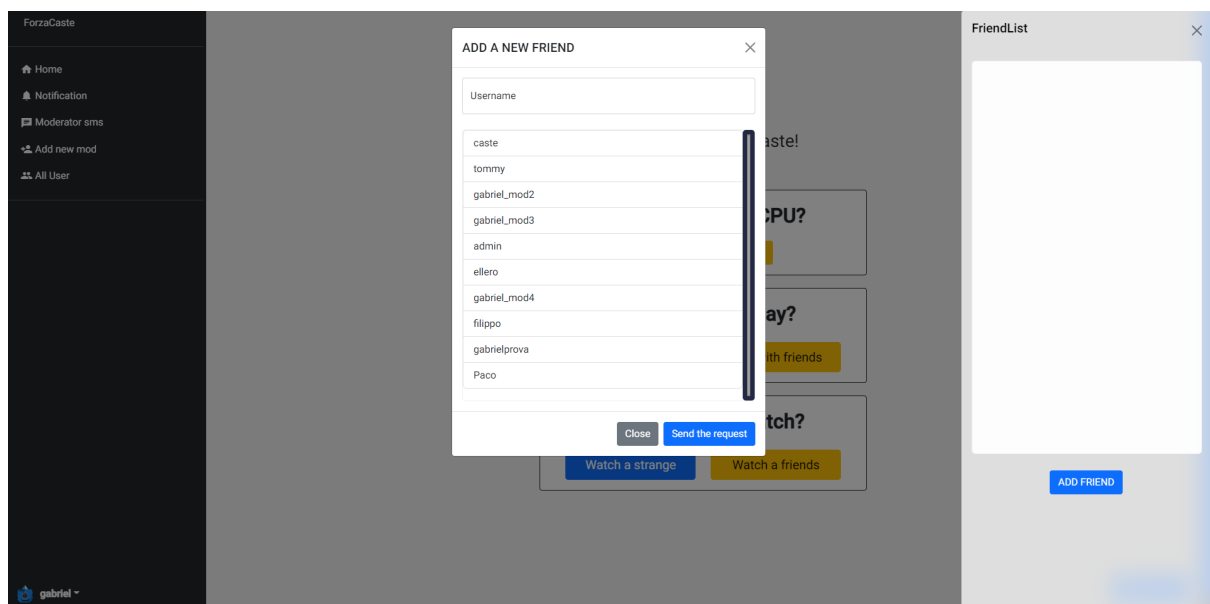
### View friendlist





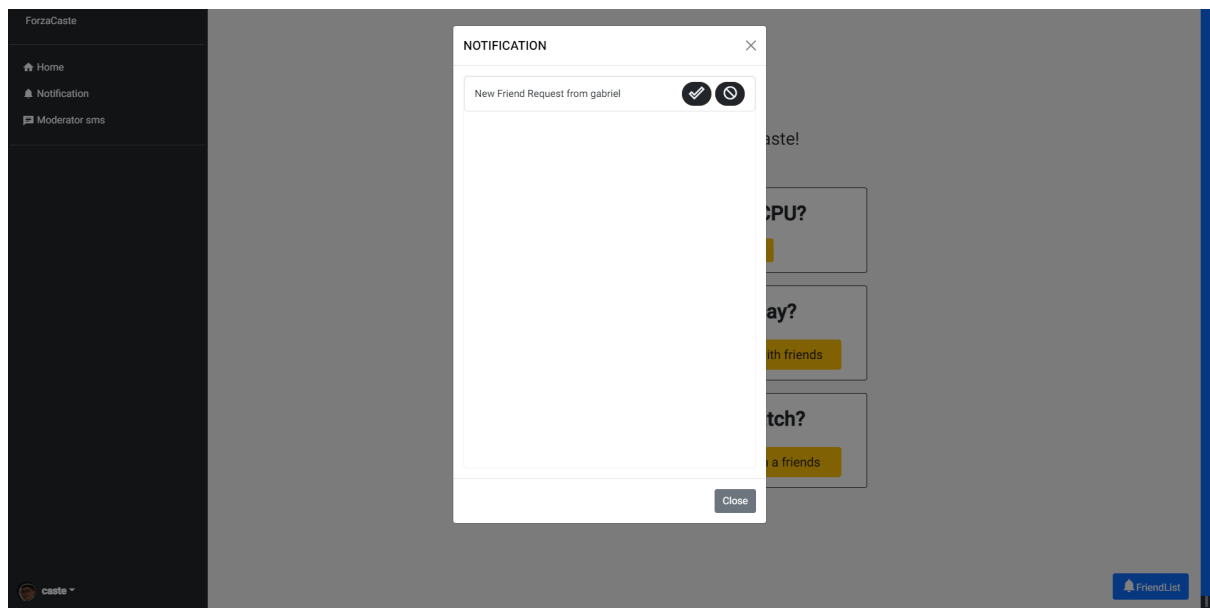
A user can access their friendlist via the 'friendlist' button located on the homepage. From here he can delete a friend, block him, view his profile or send messages to him. It is also possible to add new friends using the 'add friend' button.

## Add new friend



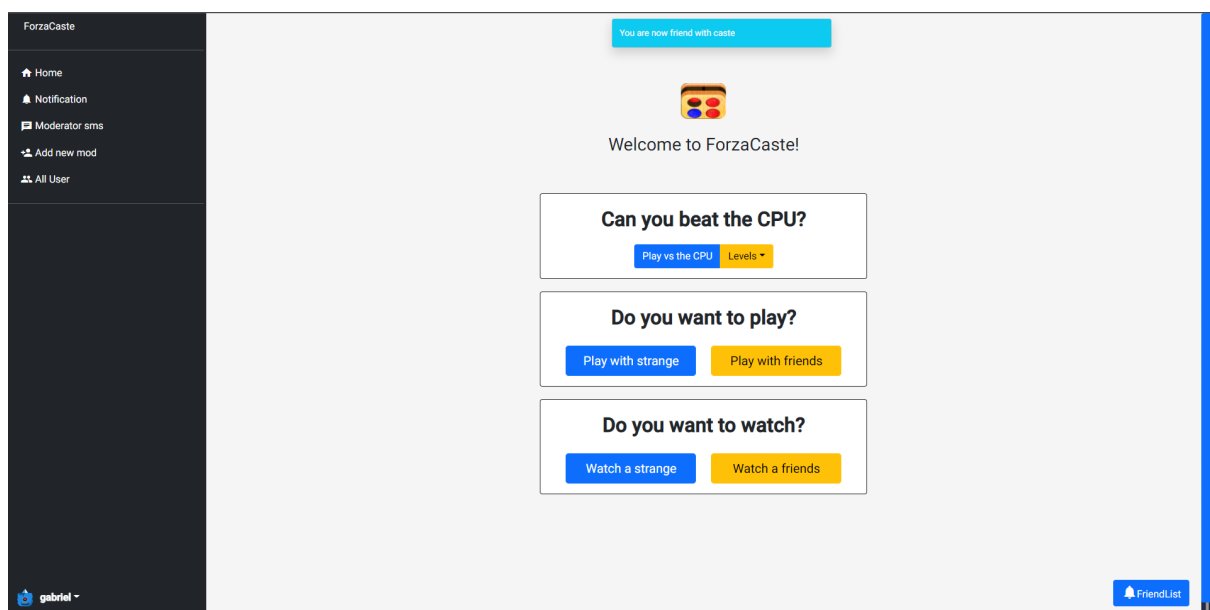
From this screen you can search for a user to become friends with, once the 'send request' button is clicked the other user will be notified and they can accept or refuse.

## Accept or refuse friend request



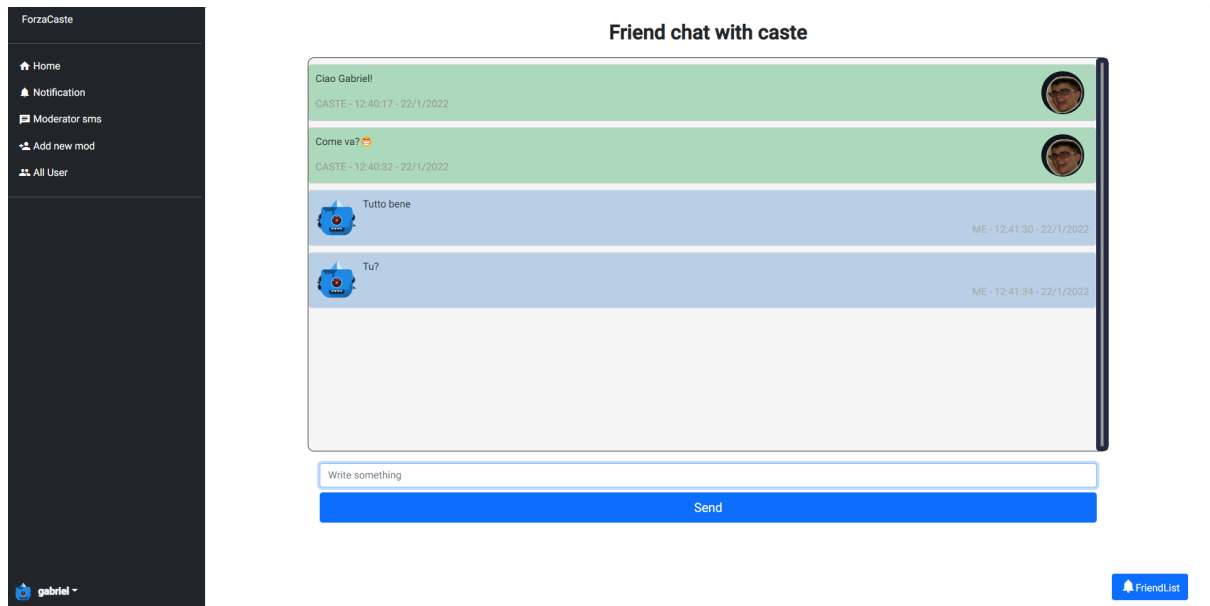
This is the screen presented to a user who has received a friend request.

## Friendship made



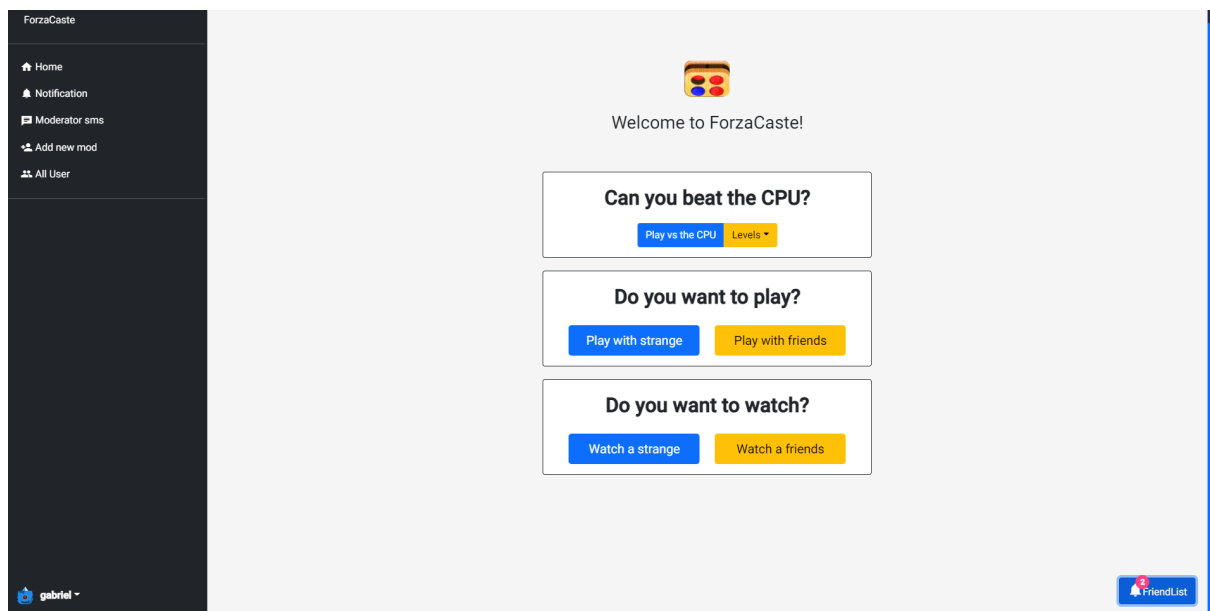
Once the other user has accepted the request a feedback is shown to the user to inform him that the request has been successful

## Chat with friend

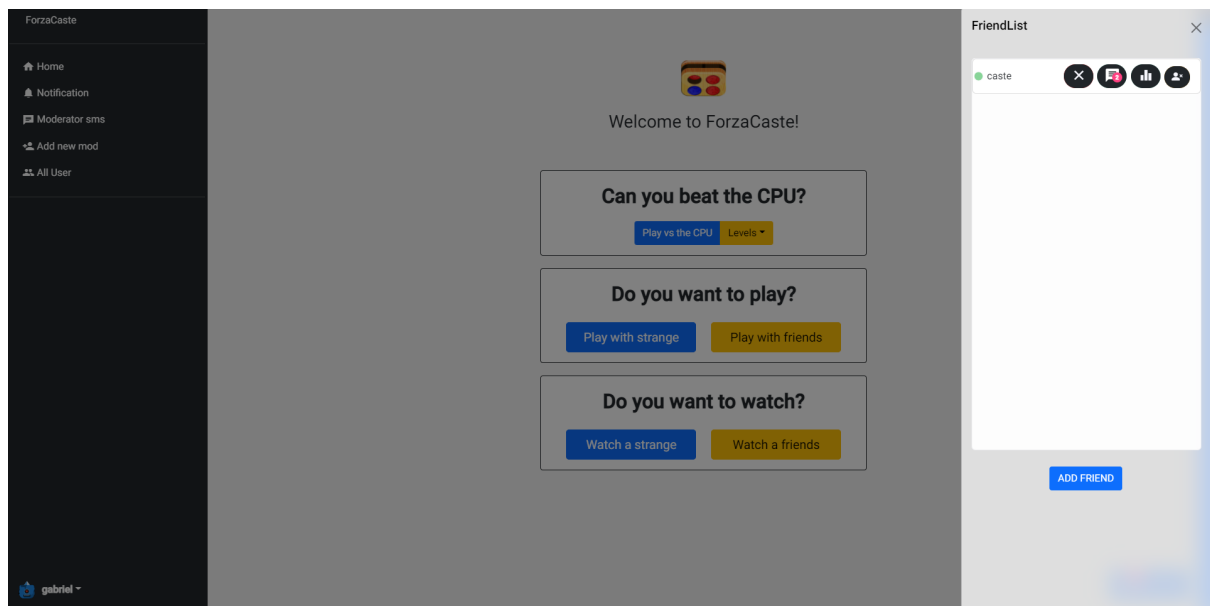


Once you make friends you can start chatting with it.  
Messages in blue represent the current user, those in green the friend.

## New message arrived

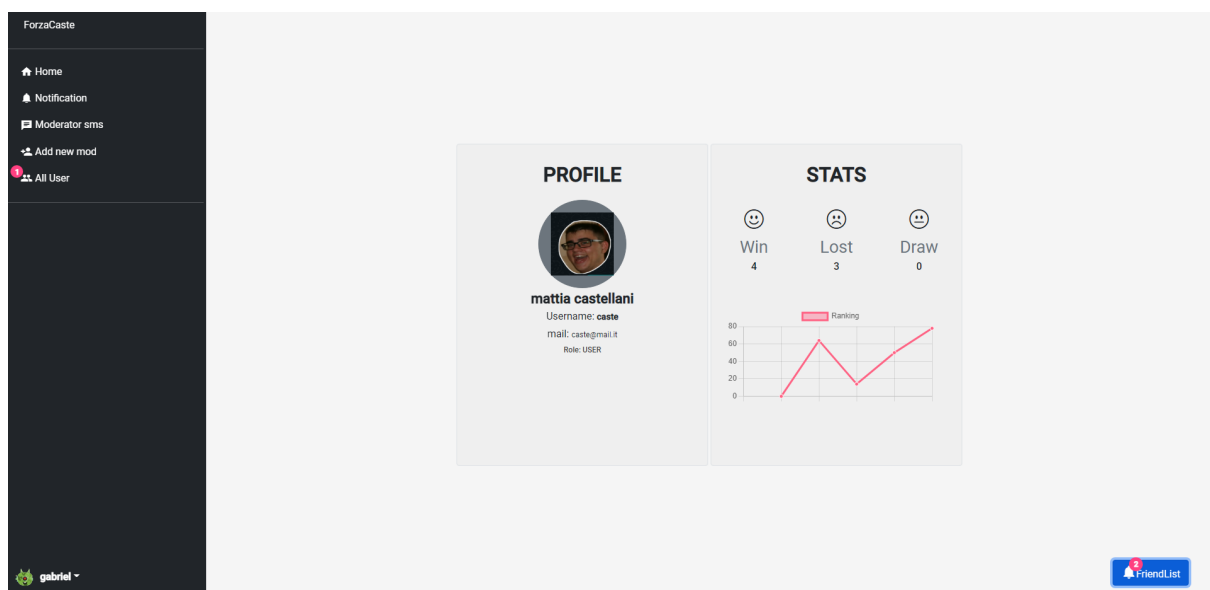


On the homepage you will be notified of new messages from friends by a badge in the friendlist button



Once you open the friendlist, a badge will indicate the number of notifications you have yet to interact with from that user

## Friend statistics



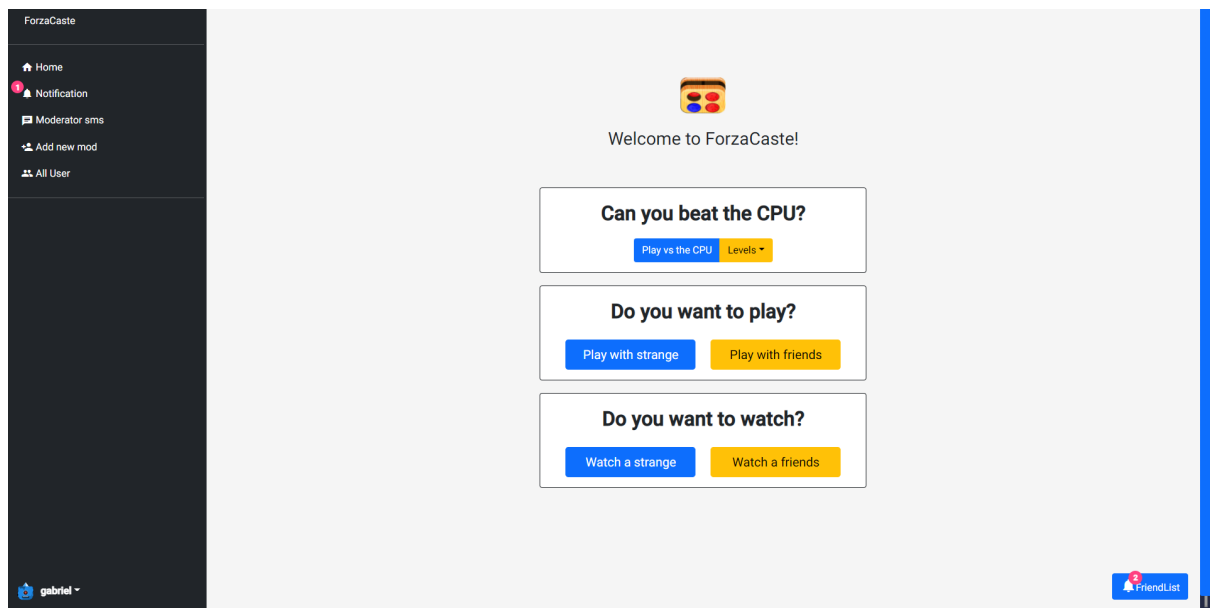
This screen gives you an overview of your friend's personal information and statistics

## Homepage

This is the main screen of the application, from here you can start new matches or watch already started matches.

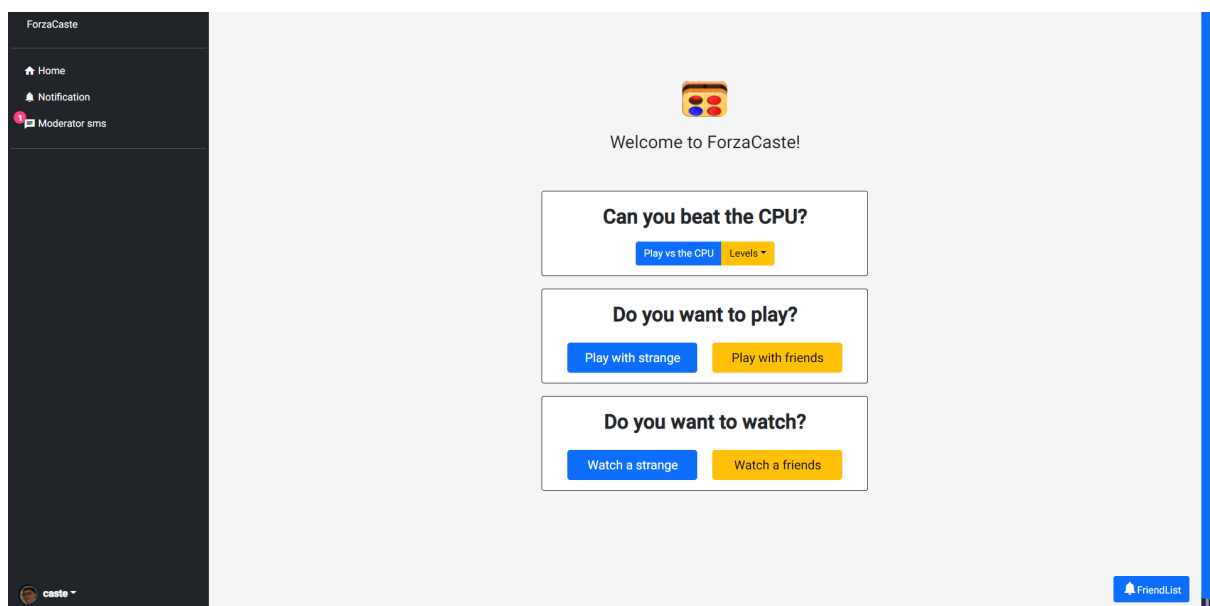
Depending on the role the user has, they will see different information.

## Moderator homepage



The moderator will see the information necessary to manage users and create new moderators

## User homepage

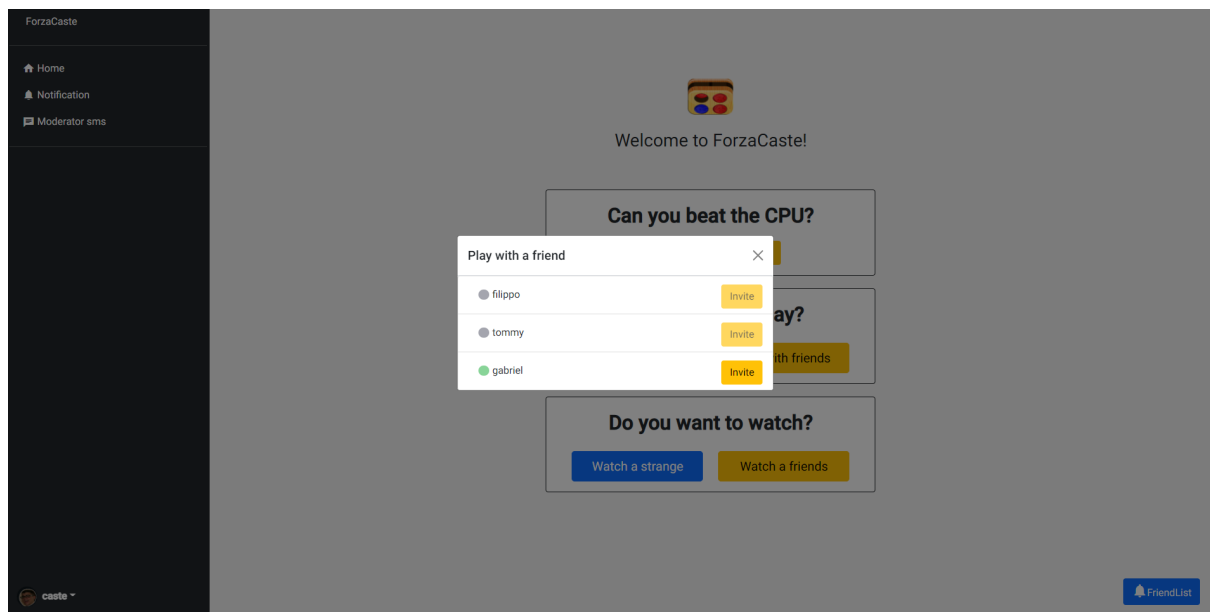


The user will see the basic information so that he can play and communicate with his friends or inform the moderators of any problems.

## Game

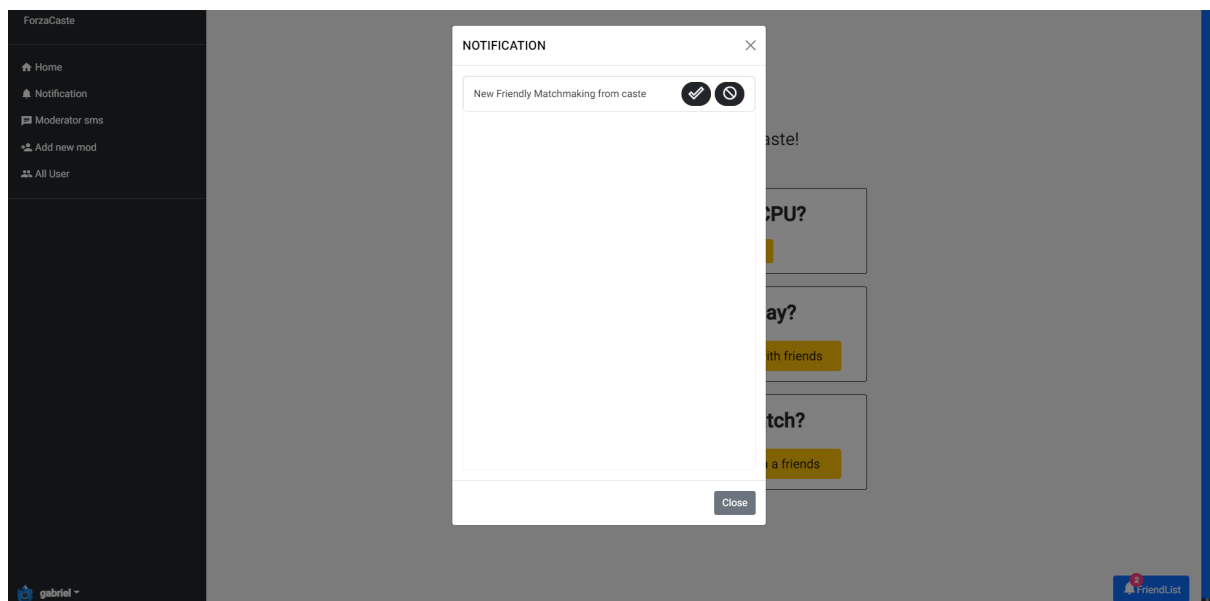
Below are shown the most important interactions that can occur during a game.

### Invite friend



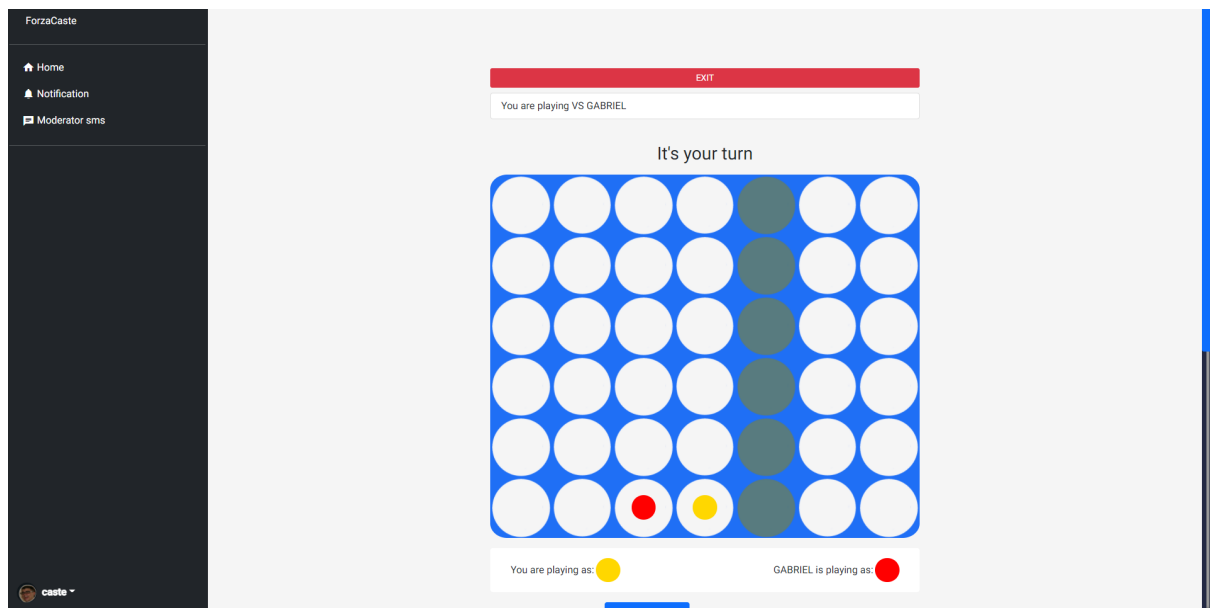
Here you can see the friends you can ask to play, of course those available are only those online (They are distinguished by the green dot).  
The player will wait for the reply from the friend.

## Accept game request



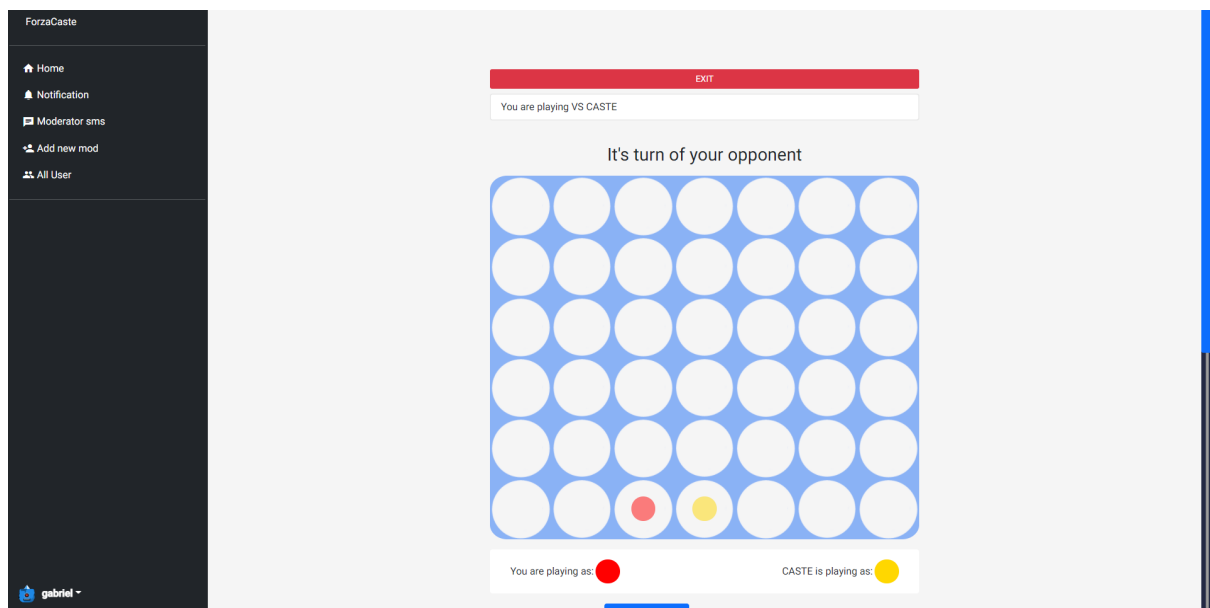
The user who receives the request will be shown a notification that it can be accepted or rejected.  
Once accepted, the game will start

## Play your turn



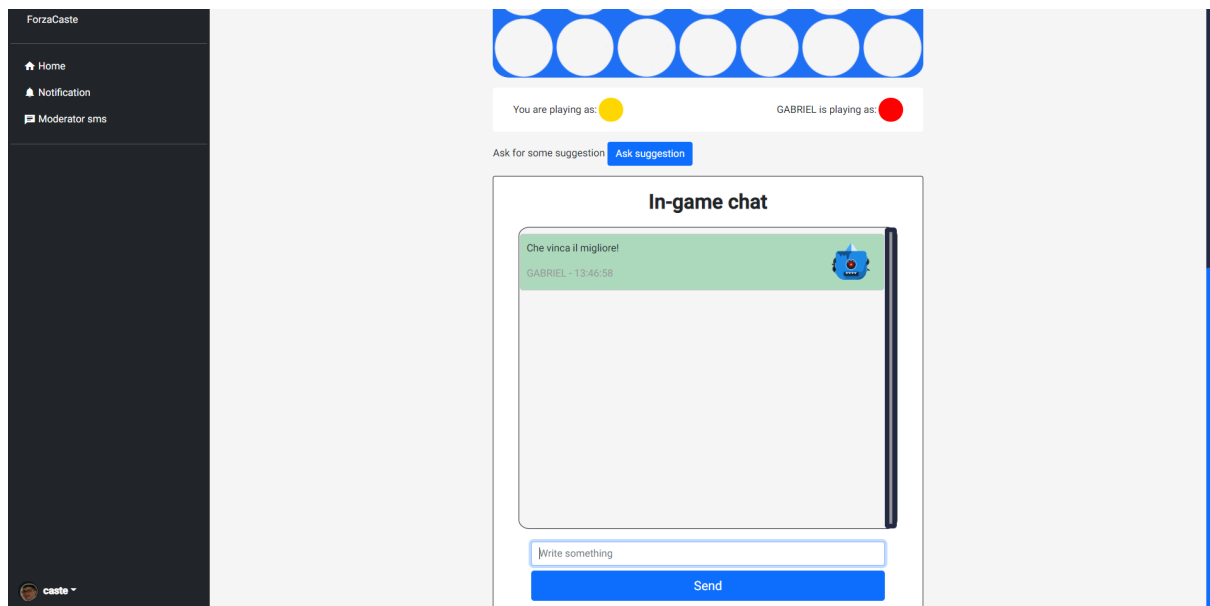
Information about the game is shown here.  
For usability, the column selected with the mouse is highlighted

## Wait opponent



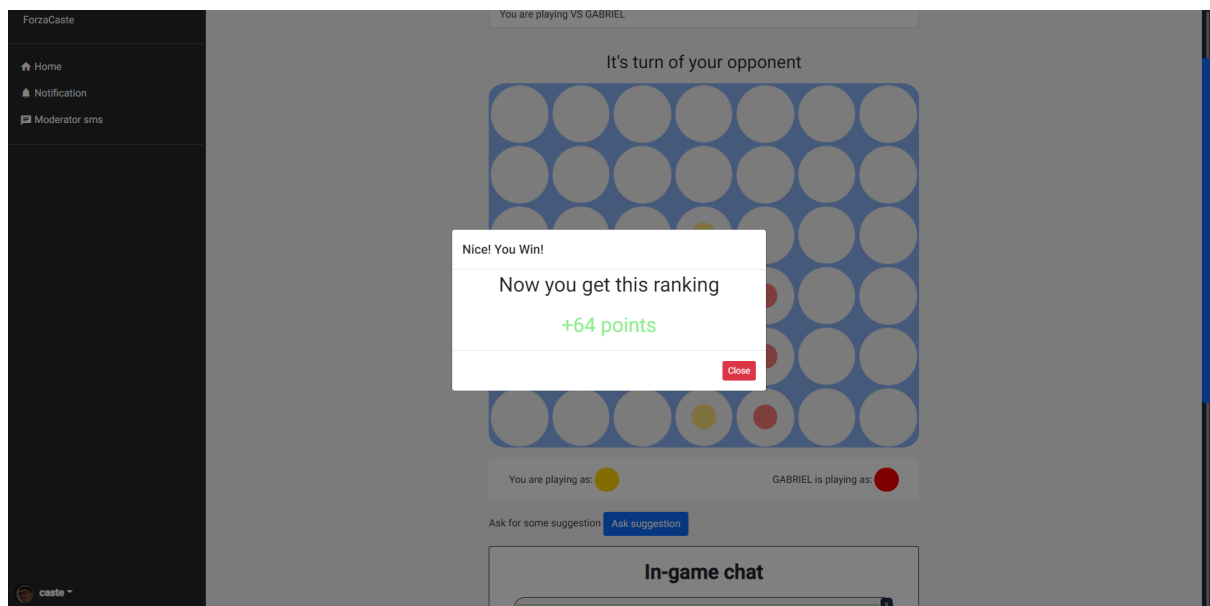
When the playing field is disabled it is the opponent's turn.  
During this phase only chat is available.

## Game chat



In-game chat, the two players can talk to each other during the game.

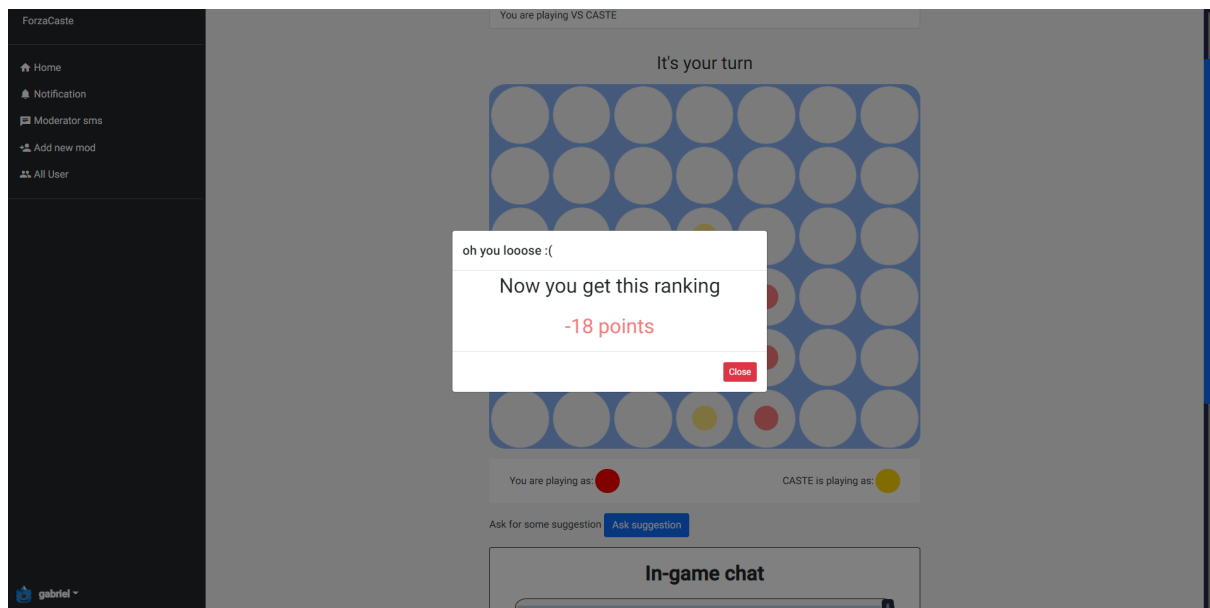
## Winning message



Once the game is over, the result is shown and the player is informed of the points earned.

## Defeat Message



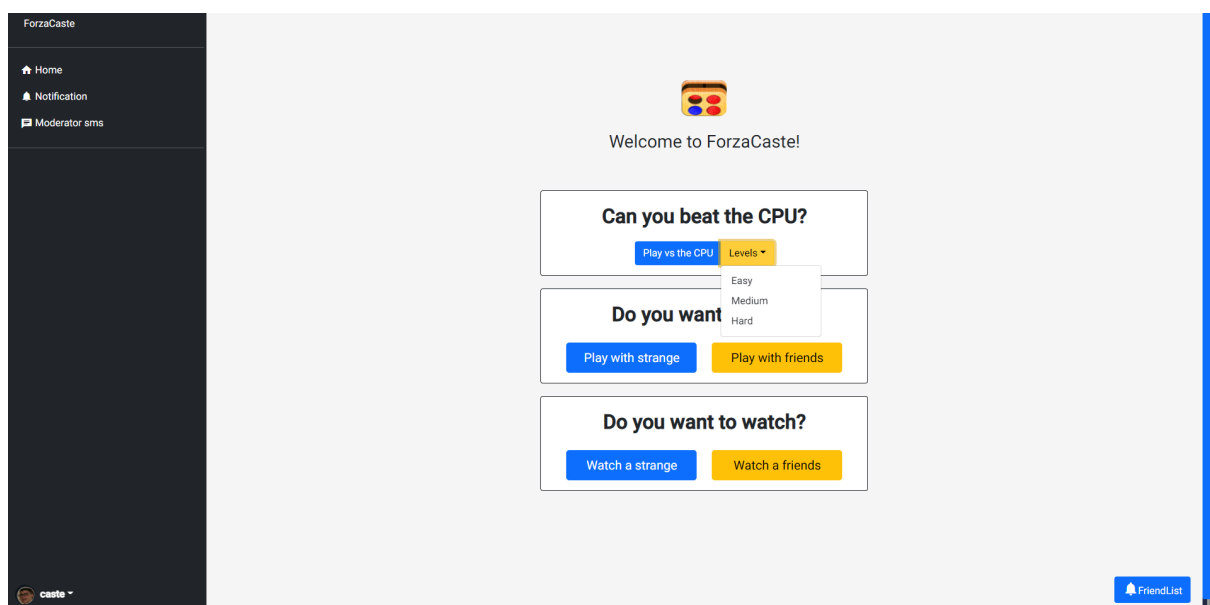


Once the game is over, the result is shown and the player is informed of the lost points.

## Play against CPU

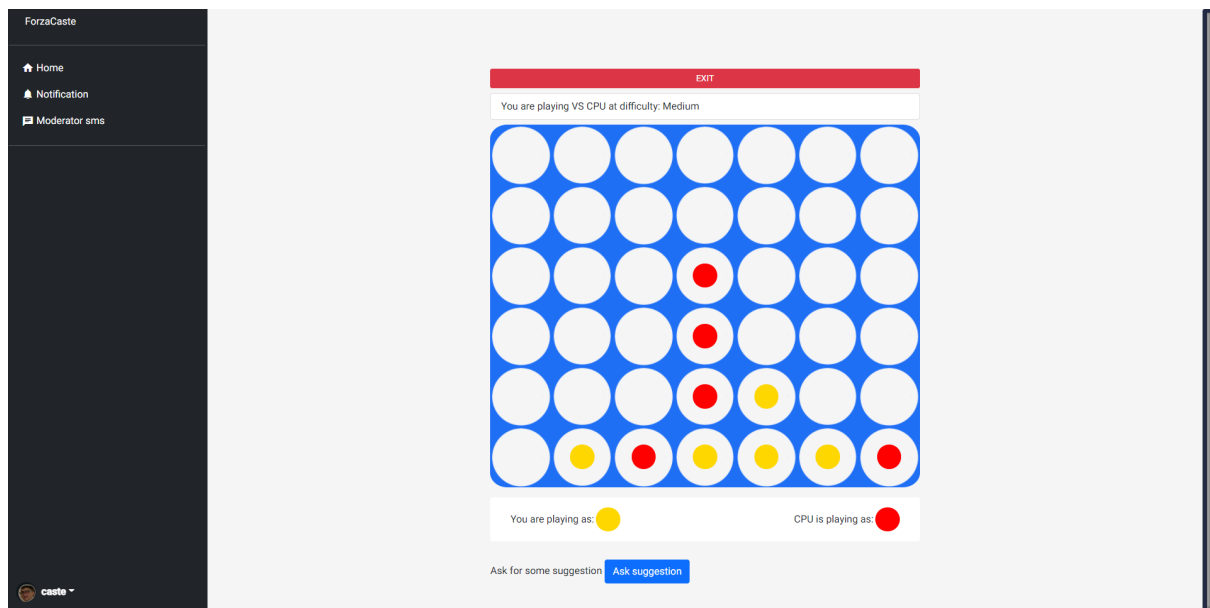
Below are shown the interactions that are needed to start a game against the CPU and those to ask for the move suggestion.

### Choose difficulty



From the homepage you can start a game against the CPU.  
In this screen you can choose the difficulty with which the cpu must play.

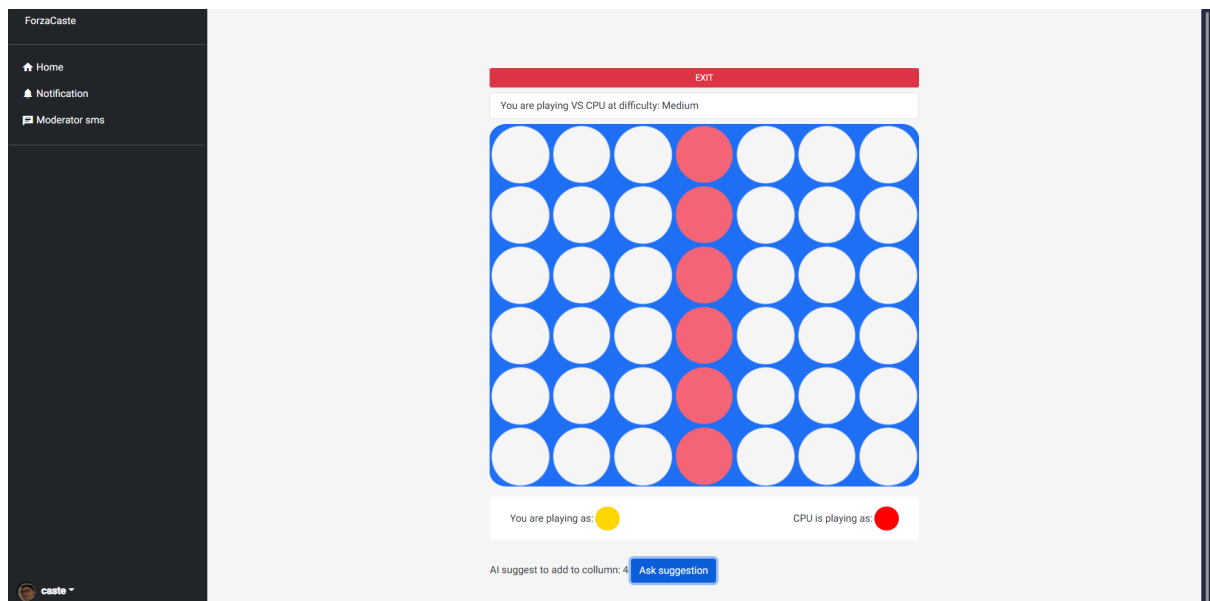
### Play your turn



The game is displayed in the same way as one against physical persons, except that the chat is disabled in this case.

## Asking for suggestion

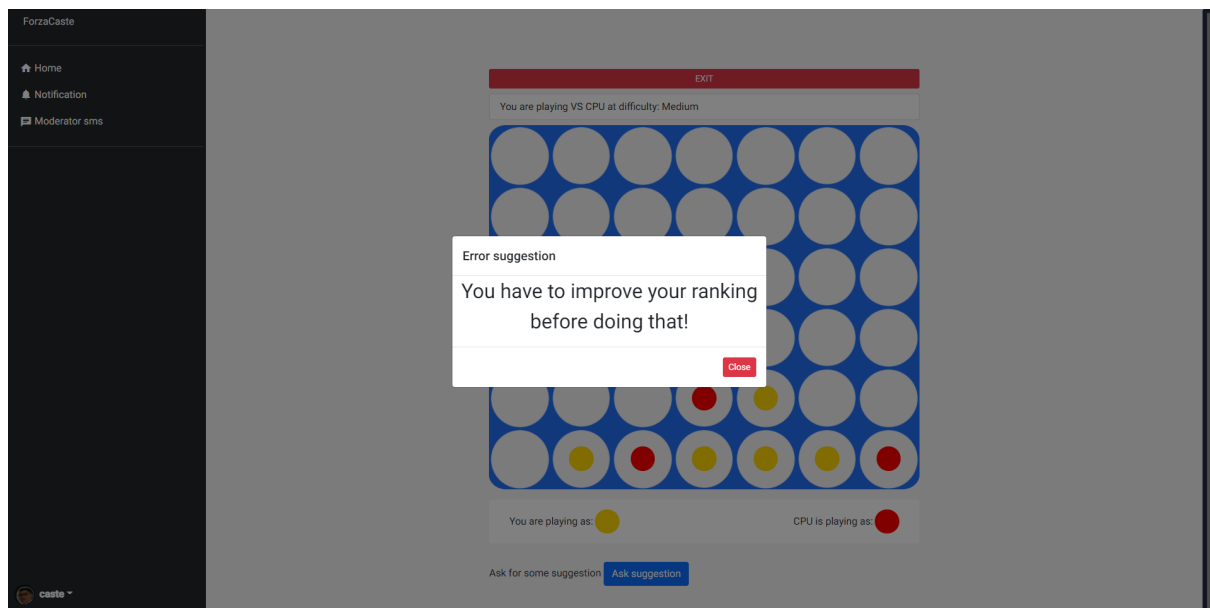
### Available



When you are in the game, if you have a ranking greater than 100, you can ask the AI for the best move to make at that moment.

The column chosen by the AI will be highlighted in red.

### Not available

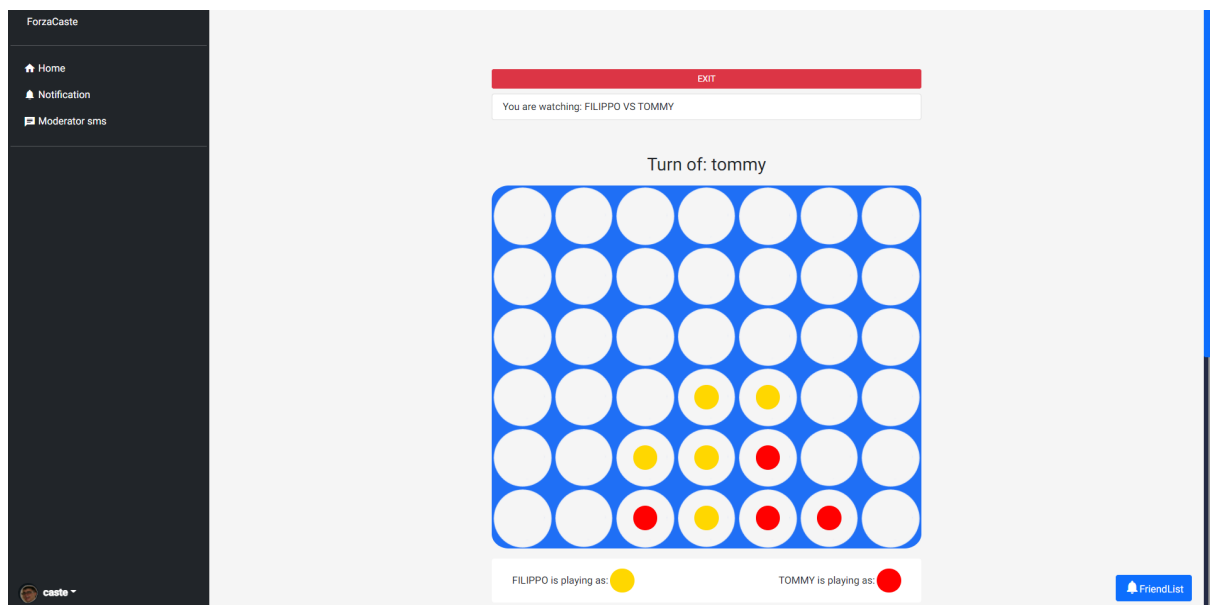


If you do not have the necessary ranking then the following message will be displayed

## Observer

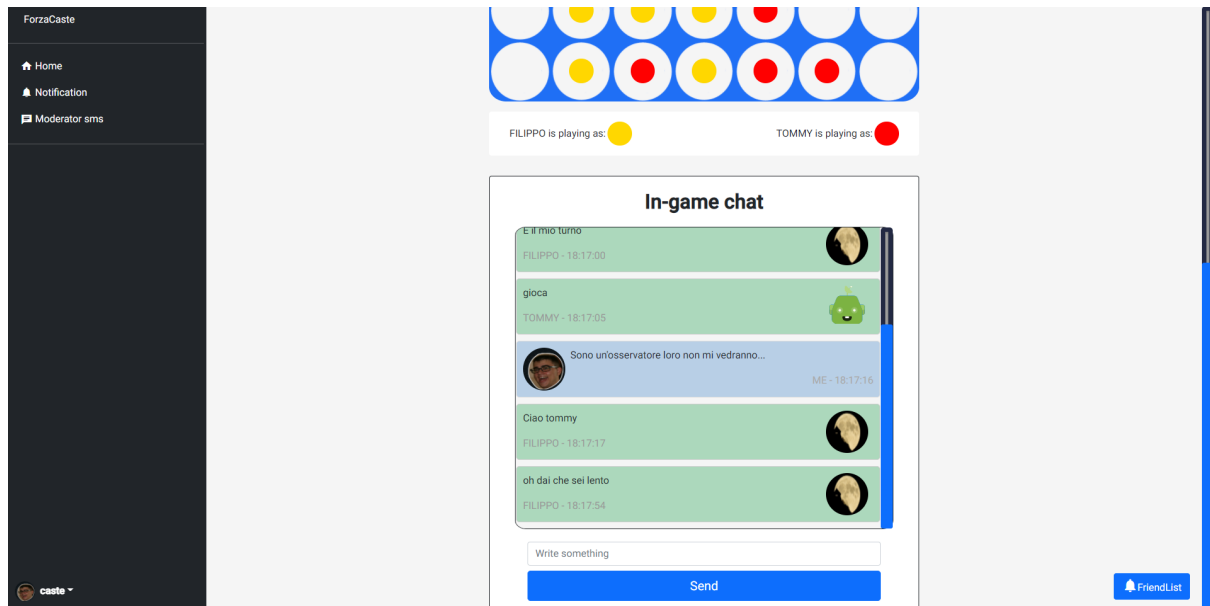
Below are shown the interactions observers can make while watching a match

### Watch a game



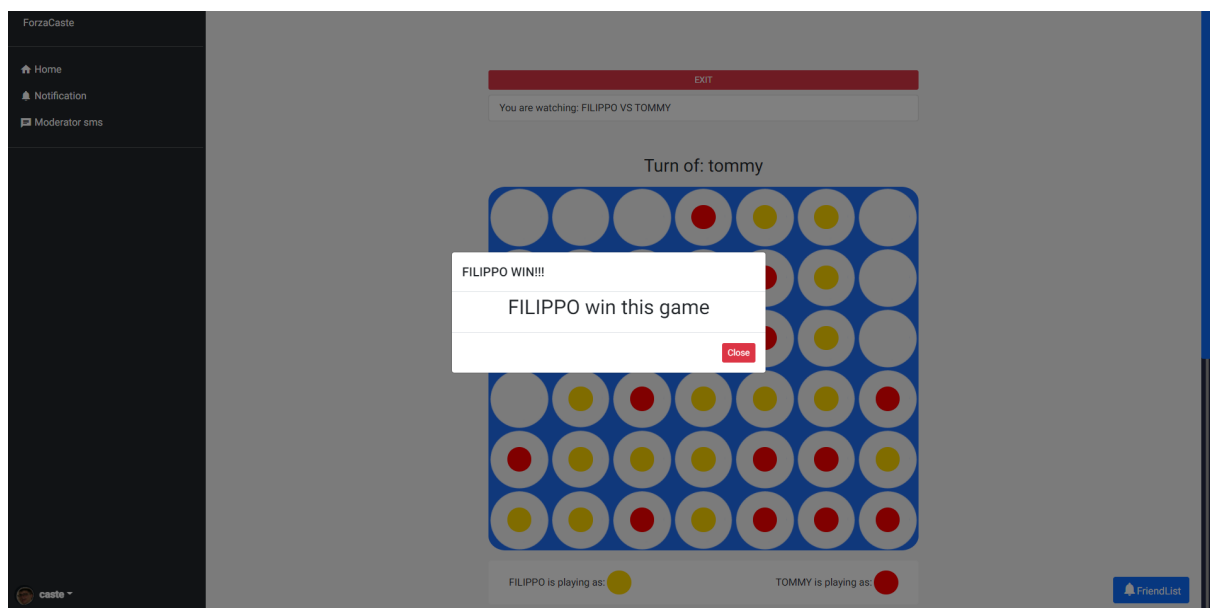
Information about the game is shown here.

### Observer chat



During the match you are watching it is possible to interact with the other observers through the chat, inside it are shown also the messages of the two players

## Winner message



Once the match is over the observers are informed of the result.

## References

1. <https://httpstatuses.com/> → An easy reference of HTTP Status Code
2. <https://app.diagrams.net/> → Software used to draw graphs
3. <https://www.notion.so/> → Software used to write the documentation

4. Minimax algorithm resources

a. <https://medium.com/analytics-vidhya/artificial-intelligence-at-play-connect-four-minimax-algorithm-explained-3b5fc32e4a4f> → Algorithm explanation

b. <https://www.youtube.com/watch?v=y7AKtWGOPAE> → Algorithm applied to Connect4 game

5. <https://avatars.dicebear.com/> → API for generating random images

6. <https://themes.getbootstrap.com/> → Bootstrap templates

7. <https://carbon.now.sh/> → Code screenshot