

# FORZA CASTE DOCUMENTATION

## Castellani Mattia - 881427

[Castellani Mattia - 881427](#)

[Introduction](#)

[Data Model](#)

[Endpoints](#)

[Socket.io](#)

[Common response](#)

[Authentication](#)

[Frontend](#)

[Services](#)

[Components](#)

[Module](#)

[Routes](#)

[Extra](#)

[Application Workflow](#)

[Login](#)

[Registration](#)

[First login as a new moderator](#)

[Homepage](#)

[Homepage \(user\)](#)

[Homepage \(Moderator\)](#)

[Notification](#)

[Moderator Dashboard \(moderators only\)](#)

[Profile](#)

[Profile data display](#)

[Friendlist](#)

[Friend Chat](#)

[Add Friend](#)

[Game](#)

[VS CPU](#)

[VS Friend or Strange](#)

[Watch a game](#)

[End-of-game screens](#)

## Introduction

The application allows users to play the game force 4 with friends or other users, and also allows additional features to be used. The creation of this application was done through continuous updates from the members on github, and through the use of a container with the aim of running the application locally and always having the updates applied by other members of the group.

This container provided the hosting of the following services:

- Image of node12.22
- JavaScript package manager, npm
- Typescript compiler
- Angular and the corresponding CLI

The Angular framework is used to create the frontend of our application, and allows you to make it a Single Page Application (SPA), to move between components is used the directive of Angular called router-outlet that allows you to display on screen the components that match the routes defined within the application, in addition we have used other services for the frontend, which are:

- CSS Framework Bootstrap
- The Angular material library, which allows the implementation of mat-icons
- Ng2-chart, for generating charts

Instead, the creation of the backend is based on a REST API architecture, the code is written via TypeScript while the API is implemented via Express.

In order to save all the information within the application we used MongoDB Atlas as the database host, while the MongoDB Compass client is used to display the database documents. The application uses [Socket.io](#) and HTTP to exchange information between client and server; [Socket.io](#) allows real-time exchange while HTTP is used to interact with the endpoints.

For testing the application in the backend we used the Postman application. Finally, in order to be able to use the application not only locally, we used Heroku, which allows the application to be used online through the use of a distributed cloud.

## Data Model

- User

```
export interface User extends mongoose.Document {
  username: string,
  name?: string,
  surname?: string,
  mail?: string,
  avatarImgURL?: string,
  roles: string,
  inbox?: Notification[],
  statistics?: Statistics,
  friendList: [{username: string, isBlocked: boolean}],
  salt?: string, // salt is a random string that will be mixed with the actual password before hashing
  digest?: string, // this is the hashed password (digest of the password)
  deleted?: boolean,
  setPassword: (pwd: string) => void,
  validatePassword: (pwd: string) => boolean,
  hasModeratorRole: () => boolean,
  setModerator: () => void,
  hasNonRegisteredModRole: () => boolean,
  setNonRegisteredMod: () => void,
  hasUserRole: () => boolean,
  setUser: () => void,
  addFriend: (username: string, isBlocked: boolean) => void,
  isFriend: (username: string) => boolean,
  addNotification: (notId: Notification) => void,
  deleteFriend: (username: string) => void,
  setIsBlocked: (username: string, isBlocked: boolean) => void,

  //User management
  deleteUser: () => void
}
```

Passwords are encrypted with the Bcrypt library before being saved.

- **Statistics**

```
export interface Statistics extends mongoose.Document {
  nGamesWon: number,
  nGamesLost: number,
  nGamesPlayed: number,
  nTotalMoves: number,
  ranking: number,
  getGamesDrawn: () => number,
}
```

This object allows you to identify the statistics for each user, specifically highlighting:

nGamesWon indicates the number of games won, nGamesLost indicates the number of games lost, nGamesPlayed indicates the number of games played, nTotalMoves indicates the number of moves a player has made to win a game, ranking indicates the score associated with individual players

- **Notification**

```
// A notification has a type of notification, some text content and a string that identify the sender
export interface Notification extends mongoose.Document {
  _id: mongoose.Types.ObjectId,
  type: string,
  text?: string,
  sender: string,
  receiver?: string,
  deleted: boolean,
  inpending: boolean, //It's used to show if a request has already been displayed
  state?: boolean, //It's used to show if a request is accepted
  ranking?: number,
  isFriendRequest: () => boolean,
  isNotification: () => boolean
}
```

This object is used to define the structure of the requests/notifications that users receive or send, specifically:

type is used to indicate what type of notification it is, e.g. a notification could be of type friendRequest

- **Message**

```
export interface Message {
  content: string,
  timestamp: Date,
  sender: string,
  receiver: string,
  inpending: boolean,
  isAModMessage: boolean
}
```

This object identifies the structure of the individual messages, specifically:

inpending indicates if the message has not yet been displayed, isAModMessage indicates if the message belongs to a chat between a user and a moderator

- **Match**





```
export interface Match {
  inProgress: boolean,
  player1: string,
  player2: string,
  winner: string,
  playground: Array<any>[6][7],
  chat: message.Message[],
  nTurns: number
}
```

This object identifies the structure of a match with attached playing field and in-game chat.





## Endpoints





For more in-depth endpoint documentation, please consult the pdf file available in the project folder named `endpoint_documentation.pdf` . This file was generated using the tool available in PostMan.

### Endpoints Table

 Endpoints	 Attributes	 Method	 Description
<a href="#">/</a>	-	GET	Returns the version and a list of available endpoints
<a href="#">/login</a>	-	GET	Login an existing user, returning a JWT
<a href="#">/whoami</a>	-	GET	Get user information and refresh the JWT
<a href="#">Users Endpoints</a>			
<a href="#">/users/:username</a>	-	GET	Return a user that has username specified
<a href="#">/users</a>	-	GET	Return a list of available user
<a href="#">/users/online</a>	-	GET	Return a list of online player at this moment

 Endpoints	 Attributes	 Method	 Description
<u>/users</u>	-	POST	Sign up a new user
<u>/users/mod</u>	-	POST	Create a new moderator, only moderator can do it
<u>/users</u>	-	PUT	Update user information
<u>/users/:username</u>	-	DELETE	Deletion of standard players from moderators
<u>Ranking endpoints</u>			
<u>/rankingstory</u>	-	GET	Return a list of ranking that logged user has at the time of game requests
<u>/rankingstory/:username</u>	-	GET	Return a list of ranking that username has at the time of game requests
<u>Game endpoints</u>			
<u>/game</u>	-	GET	Returns a list of game in progress
<u>/game</u>	-	POST	Create a random or friendly match. Furthermore a user can enter in a game as observer
<u>/game/cpu</u>	-	POST	Create a match against CPU. Furthermore a user can enter in a game as observer
<u>/game</u>	-	DELETE	Used by a player in order to delete a started game or to delete a game request
<u>/game</u>	-	PUT	Accept a friendly game request

 Endpoints	 Attributes	 Method	 Description
<u>/move</u>	-	POST	Play the turn making a move, it contains the game logic and the event notifier
<u>/move/cpu</u>	-	POST	Play the turn vs AI, it contains the game logic and call minmax algorithm to play the AI turn
<u>/gameMessage</u>	-	POST	Send a message in the game chat
<u>Notification endpoints</u>			
<u>/notification</u>	-	POST	Create a new friend request
<u>/notification</u>	-	GET	Return all the notification of the specified user. This endpoint returns all the notification that are received and that are not read
<u>/notification</u>	?inpending=bool	GET	Return all the notification of the specified user. This endpoint returns all the notification that are not read
<u>/notification</u>	? makeNotificationRead=bool	GET	Return all the notification of the specified user. This endpoint mark all the notification as read
<u>/notification</u>	-	PUT	Change the status of the notification, so the indicated notification will appear as read
<u>Message endpoints</u>			

 Endpoints	 Attributes	 Method	 Description
<a href="#"><u>/message</u></a>	-	GET	Returns all messages and all messages in pending
<a href="#"><u>/message</u></a>	?modMessage=bool	GET	Returns all moderator messages and all moderator messages in pending
<a href="#"><u>/message</u></a>	-	POST	Send a private message to a specific user
<a href="#"><u>/message/mod</u></a>	-	POST	Send a private moderator message to a specific user
<a href="#"><u>/message</u></a>	-	PUT	Update a specific message and marks it as read
<a href="#"><u>Friend endpoints</u></a>			
<a href="#"><u>/friend</u></a>	-	GET	Return the friendlist of the current logged user
<a href="#"><u>/friend/:username</u></a>	-	DELETE	Deletion of a friends in the friendlist of the current logged user
<a href="#"><u>/friend</u></a>	-	PUT	Change the attribute isBlocked of the specified user in the friendlist

The bool attribute is used to indicate all true or false values.

## Socket.io

The following are the most common events for communication using sockets. In our application, however, it was mainly preferred to use http requests to exchange information.

For more in-depth documentation of the events, consult the pdf file available in the project folder under the name [endpoint\\_documentation.pdf](#) . This file was generated using the tool available in PostMan.



 Event	 Body	 Description
<u>gameReady</u>	{ 'gameReady': true, 'opponentPlayer': "" }	Notifies players that the game is about to start
<u>move</u>	{ 'yourTurn': bool }	Notifies the user whose turn it is in the game
<u>move</u>	{ "error": true, "codeError": x, "errorMessage": "Wrong turn" }	Notifies the player whether the move is valid or not
<u>move</u>	{ move: index }	Indicates the move made by the opponent in the game, index indicates the chosen column
<u>gameStatus</u>	{ 'playerTurn': "" }	Notifies the match observer who will be the first player to make the move.
<u>gameStatus</u>	{ player: username, move: index, nextTurn: otherPlayer }	Notifies the match observer who made the move by indicating the chosen column and who will be playing in the next round.
<u>gameStatus</u>	{ player: 'player username or cpu', move: index, nextTurn: 'player username or cpu' }	Notifies the match observer who made the move by indicating the chosen column and who will be playing in the next round (if playing against the cpu).
<u>result</u>	{ "winner": null }	Notifies the players that the match ended in a draw
<u>result</u>	{ winner: bool }	Notifies whether the user has lost (false) or won (true) the game
<u>result</u>	{ "rank": rank }	Indicates the score lost I gained by a user at the end of the game
<u>result</u>	{ winner: 'username of the player who won' }	Notifies the match observers about the winner of the match
<u>result</u>	{ 'winner': "", 'message': "Opposite player have left the game" }	Notifies the user that his opponent has left the game
<u>gameRequest</u>	{ type: "friendlyGame", player: "" }	Notifies the user that a game request has arrived
<u>enterGameWatchMode</u>	{ 'playerTurn': "", playground: "" }	Notifies the user who wants to enter as an observer about the status of the game

Aa Event	≡ Body	≡ Description
<u>gameChat</u>	{'_id' : '', 'content' : '', 'sender' : '', 'receiver' : '', 'timestamp' : {"\$date": ""}}	Informs all users in a game (observers and non-observers) of the arrival of new chat messages
<u>request</u>	{newFriend: ""}	Informs the user that the friend request has been accepted (if newFriend is null the request will be rejected instead)
<u>friendDeleted</u>	{deletedFriend: ""}	Informs the user that a friend has been removed from the friends list
<u>friendBlocked</u>	{blocked: bool}	Informs the user if he has been blocked or unlocked by a friend
<u>newNotification</u>	{ sender: '', type: "friendRequest"}	Informs the user that a friend request has arrived
<u>message</u>	{'_id' : '', 'content' : '', 'sender' : '', 'receiver' : '', 'timestamp' : {"\$date": ""}}	Informs the user that it has received a new message
<u>message</u>	{'_id' : '', 'content' : '', 'sender' : '', 'receiver' : '', 'timestamp' : {"\$date": ""}, inpending: "", isAModMessage:true}	Informs the user that he has received a new message (in the moderators chat)

## Common response

Below is a table indicating the most common responses that can be returned as a result of an HTTP request.

**Common Response Table**

# StatusCode	Aa Error Message
200	<u>OK , The request has succeeded</u>
400	<u>BAD REQUEST, The server cannot or will not process the request due to something is perceived to be a client error</u>
401	<u>UNAUTHORIZED, The request has not been applied because it lacks valid authentication credentials for the target resource</u>
403	<u>FORBIDDEN, The server understood the request but refuses to authorize it</u>

# StatusCode	Aa Error Message
404	<u>NOT FOUND, The origin server did not find a current representation for the target resource or is not willing to disclose that one exists</u>
502	<u>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 teh request</u>

## Authentication

Authentication is based on the use of tokens, specifically JWT tokens are used. This is done through the use of a Middleware called passport.js, which filters the data for authentication, if the data are correct calls a function that builds a token according to a defined structure, once authenticated a token is generated signed with the JWT\_SECRET.

Server side:

In our case, basic authentication is used through the passport.authenticate('basic') function, which in turn calls a lamda function to define how to authenticate. If the credentials entered are the same as those in the database, the /login function is called, which creates the token that is encrypted with the JWT\_SECRET defined in the .env and then signs it. In addition, the token expired is set to one hour.

Client side:

The JWT token is saved in the client using session storage, which allows the token to be saved in the browser.

A token renewal mechanism is used through the use of the whoami endpoint which is invoked each time the client navigates, this endpoint calculates the expired token and decides whether to create a new token depending on the time remaining on the current token.

## Frontend

The frontend of the application consists of the services, components and modules placed inside the tau folder. The components are organised in different folders that identify them, while the services are visible to the whole application, just like the modules. The components (of which there are 14) define the display of the application using the data provided, while the services define the interaction with the server and between the components.

## Services

- **User-http.service**

This service contains all the possible functions for managing users within the application, and also contains all the methods for interacting with the server. These methods allow http requests to be made to the correct endpoints, returning the server's response. This service also contains all the functions for creating and managing bots.

- **Socketio.service**

This service contains all the methods for interacting with the server using Socketio. It also transforms Socketio calls into observables that can be managed by components.

- **Toast.service**

This service manages the creation of toasts within the client, providing methods for interacting with components

## Components

- **All-user component**

This component is only displayed by the moderators, and allows you to obtain all the players registered within the platform. It also allows the moderators to manage these players.

- **Cpu component**

This component allows players to play a game with a bot, and manages the moves and difficulty of the bot they are playing against.

- **Friend-chat component**

This component makes it possible to obtain and display all the messages between a player and another player in his friends list, using the methods contained in the user-http service. It also allows a real time chat between the two players thanks to the methods contained in the socketio service.

- **Friend-stats component**

This component allows you to view the statistics and details of a given player in your friends list.

- **Game component**

This component allows players to play with other players by displaying the playing field and validating all moves by sending them to the server. It also allows you to send a friend request to your opponent if they are not on your friends list.

- **Homepage component**

This component is displayed by each registered user immediately after authentication, it allows you to start a game with an unspecified user, or with a player in your friends list who is currently online; it also allows you to access games in progress as an observer of both foreign players and those in your friends list

- **Mod-chat component**

This component makes it possible to obtain and display all the messages between a player and a moderator, using the methods contained in the user-http service. It also allows a real-time chat between the two players using the methods contained in the socketio service.

- **New-moderator component**

This component only allows moderators to generate a new moderator with temporary credentials.

- **Sidebar component**

This component is used to manage the application menu. It allows you, if you are a registered user, to display the list of friends and to access all of its functions, to display the list of moderators and to be able to send messages to them; it also allows you, if you are a moderator, access to the components to add a new moderator and to display the list of all registered users. It also allows you to send friendship requests to users registered within the application by displaying them in a list.

- **User-login component**

This component makes use of the authentication part of the user-http service to authenticate the user and allow them access to the application.

- **User-logout component**

This component makes use of the authentication part of the user-http service to authenticate the user and allow them access to the application.

- **User-profile componet**

This component allows users to view their profile with their game statistics displayed via a graph of their game history. It also allows you to modify your profile by changing or adding some of its details and modifying the avatar image. It also manages the first access of a new moderator who must enter a new password.

- **User-signin component**

This component manages the registration of new users within the application.

- **Watch component**

This component allows you to access and view games in progress between two players as an observer, it also allows you to view and use real-time chat between players involved in the game and other observers present.

## Module

- **App-routing.module**

This module contains all the routes that allow navigation between the components within our application.

## Routes

- **path:** “  
**redirectTo:** ‘/login’  
**pathMatch:** ‘full’
- **path:** ‘login’  
**component:** UserLoginComponent
- **path:** ‘home’  
**component:** HomepageComponent
- **path:** ‘logout’  
**component:** UserLogoutComponent
- **path:** ‘signin’  
**component:** UserSigninComponent
- **path:** ‘profile’

**component:** UserProfileComponent

- **path:** 'new-mod'

**component:** NewModeratorComponent

- **path:** 'all-user'

**component:** AllUserComponent

- **path:** 'game'

**component:** GameComponent

- **path:** 'watch'

**component:** WatchComponent

- **path:** 'cpu'

**component:** CpuComponent

- **path:** 'user-stats/:friend'

**component:** FriendStatsComponent

- **path:** 'friend-chat/:friend'

**component:** FriendChatComponent

- **path:** 'mod-chat/:user'

**component:** ModChatComponent

- **path:** \*\*

**redirectTo:** '/login'

**patchMatch:** 'full'

## Extra

- **Toast component**

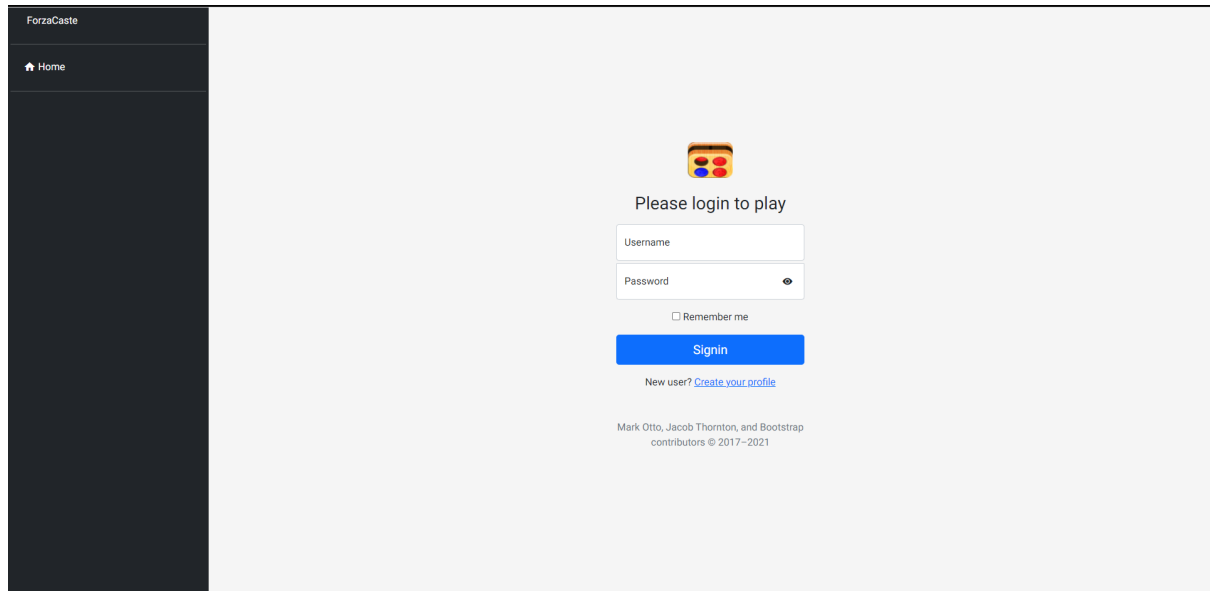
This component allows the creation of toasts within the application.

# Application Workflow

The following views describe how our application works, highlighting its main features.

You can also test a live version of the application by clicking on the following link <http://forzacaste.live/>.

## Login



ForzaCaste

Home

Please login to play

Username

Password

☐ Remember me

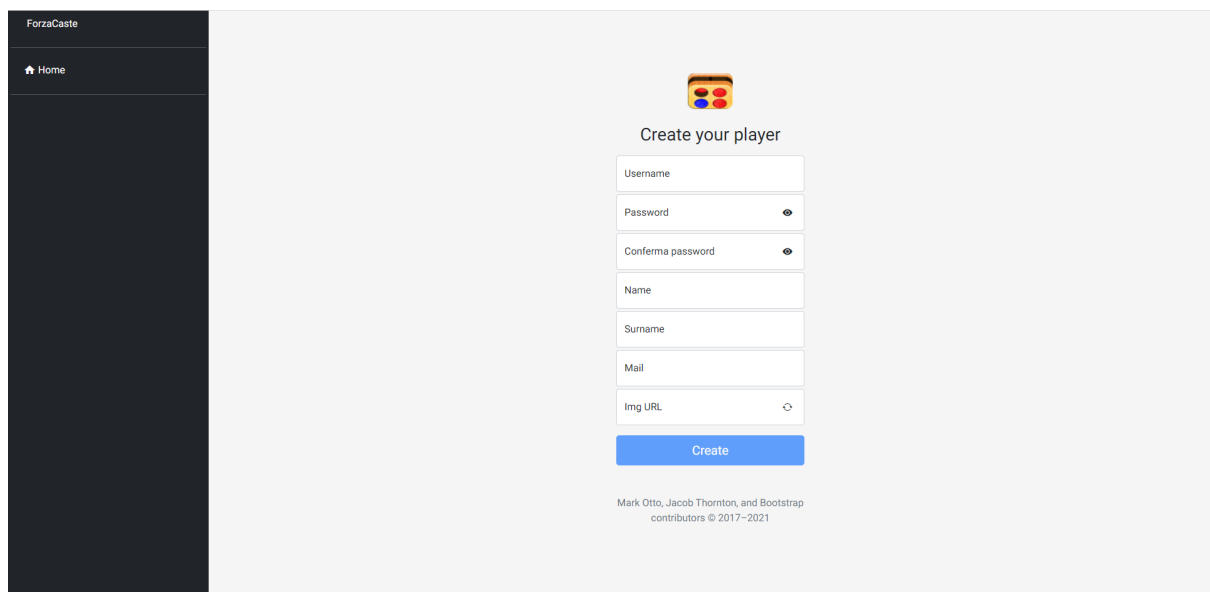
Signin

New user? [Create your profile](#)

Mark Otto, Jacob Thornton, and Bootstrap contributors © 2017-2021

It allows the user to authenticate within the application and to access the registration page for creating a new account.

## Registration



ForzaCaste

Home

Create your player

Username

Password

Conferma password

Name

Surname

Mail

Img URL

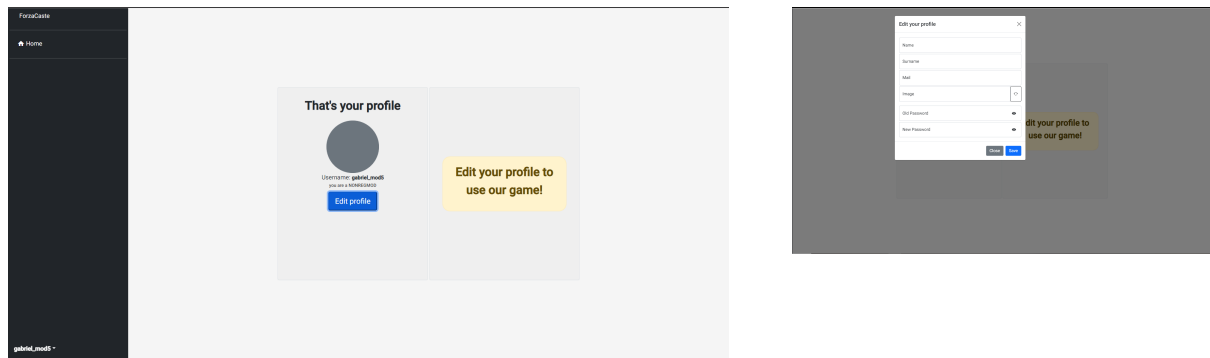
Create

Mark Otto, Jacob Thornton, and Bootstrap contributors © 2017-2021



In addition to allowing you to register a new account, this view allows you to choose the avatar image of your account.

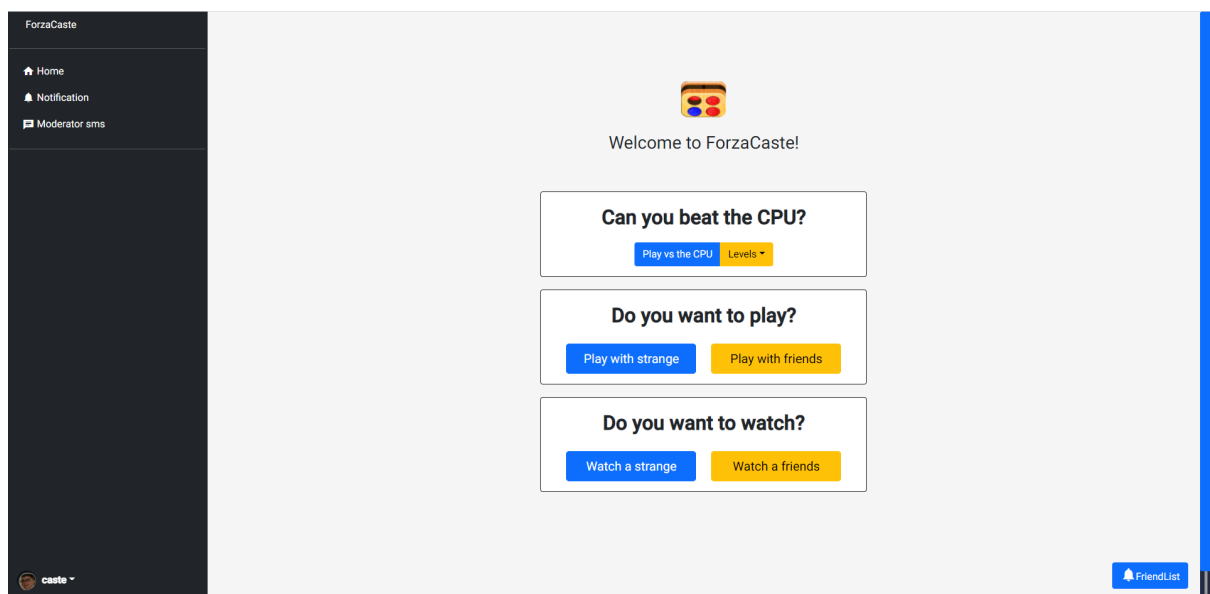
## First login as a new moderator



The first view (left) is displayed when a new moderator logs in for the first time. In order to actually access the application, the moderator will have to enter his new data (second view, right).

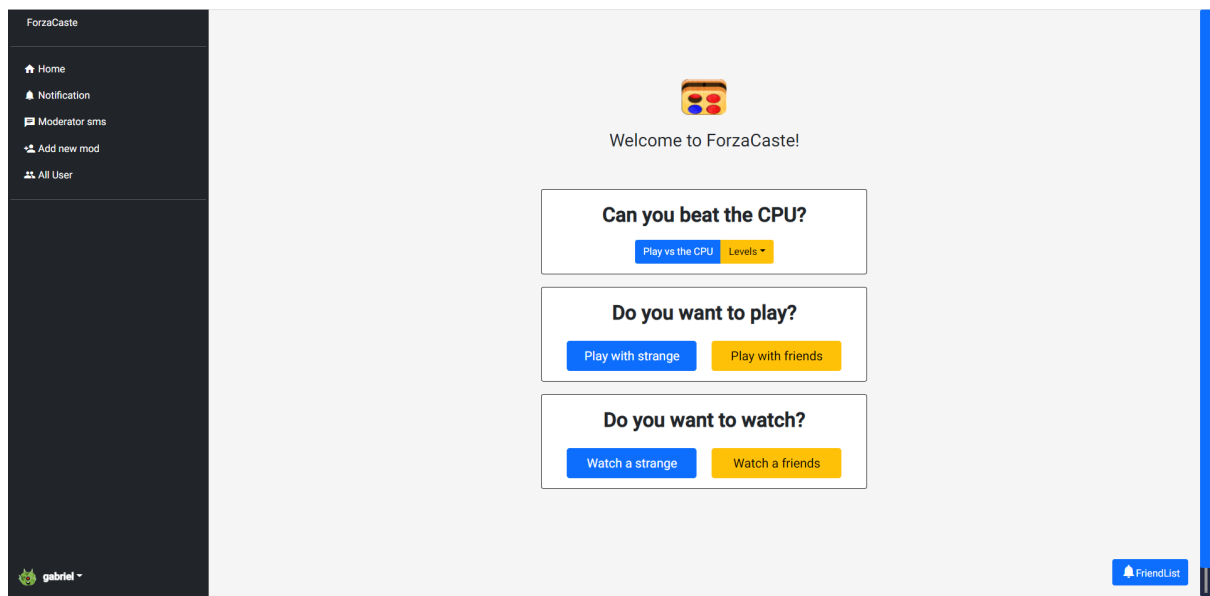
## Homepage

### Homepage (user)



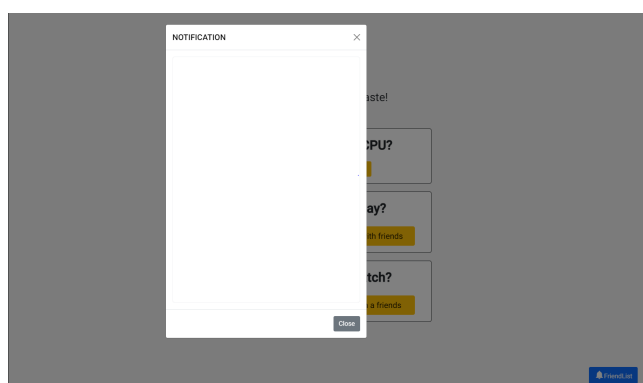
The homepage of the application displays all the possible actions that a user can perform. Primarily, it displays the possibility of playing (or accessing) a game in different modes. It also allows you to send messages to the moderators if problems occur by clicking on the moderator sms button and accessing the required chat.

## Homepage (Moderator)



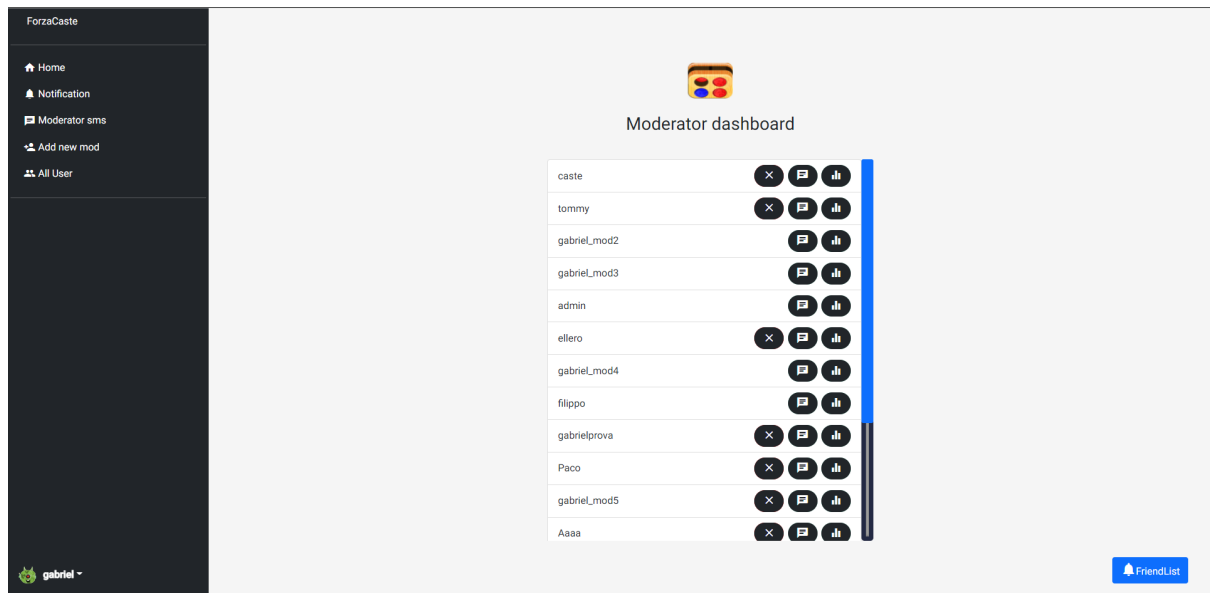
In the case of a moderator, the additional functionality it can undertake is displayed.

## Notification



Clicking on the notification button in the sidebar displays a modal with a list of incoming notifications to the user, so it is also possible to accept any requests.

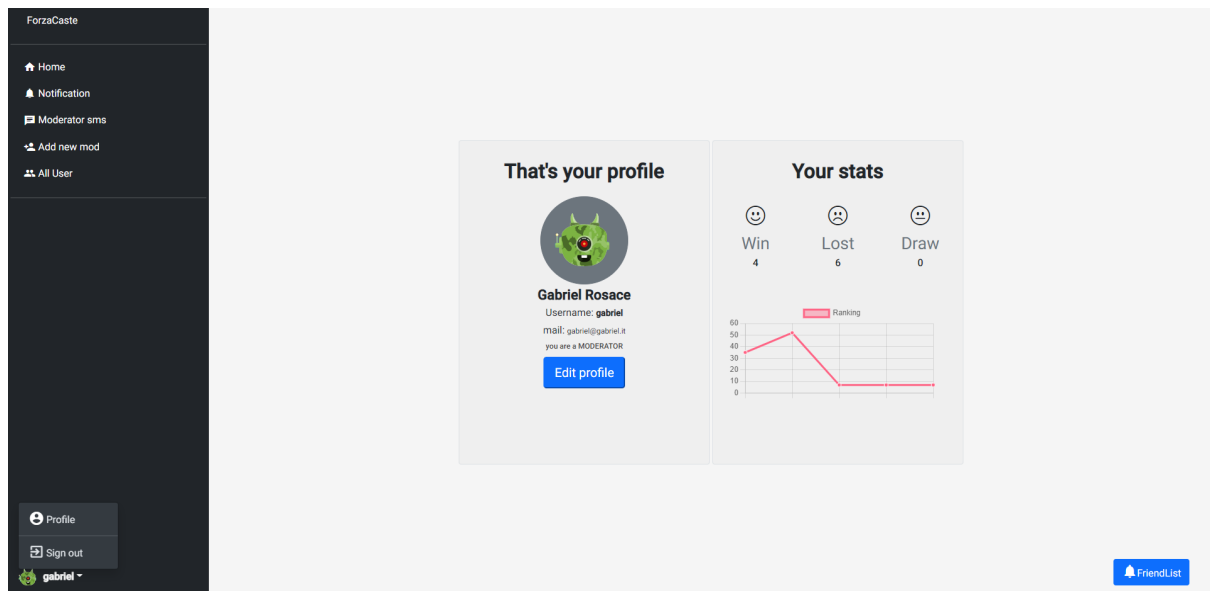
## Moderator Dashboard (moderators only)



This screen allows moderators to manage accounts within the application.

## Profile

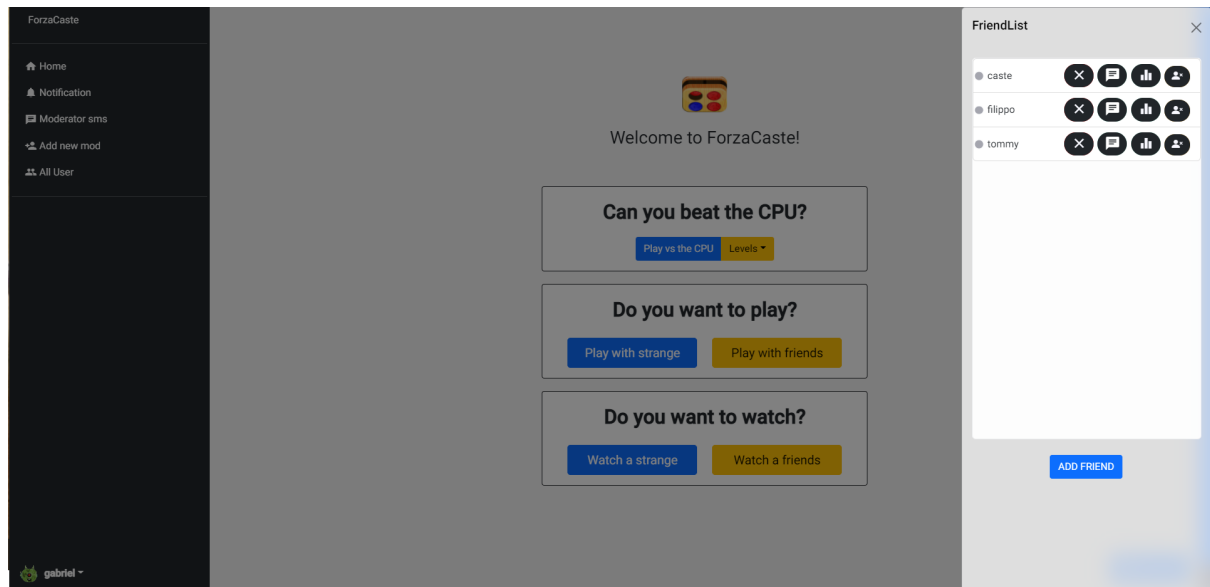
### Profile data display



This view allows you to view not only your account details, but also statistics about the games you have played, as well as a graph of their progress. You can also log out of any view by clicking on the sign out button.

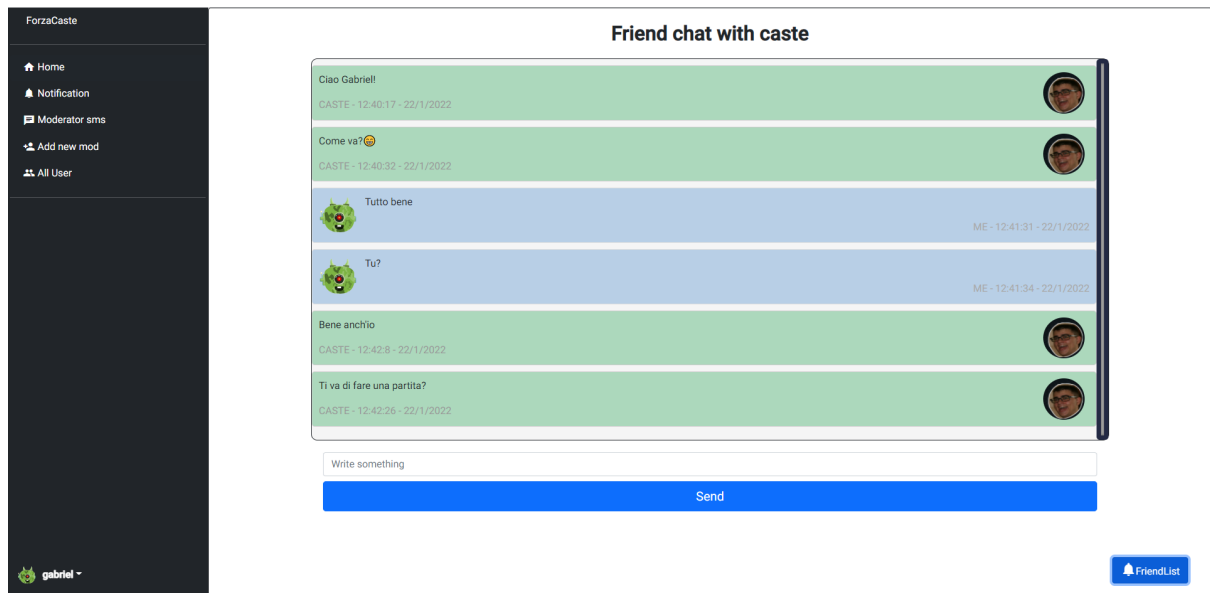
# Friendlist

By clicking on the FriendList button in almost every view (except the game view) you can access your friends list.



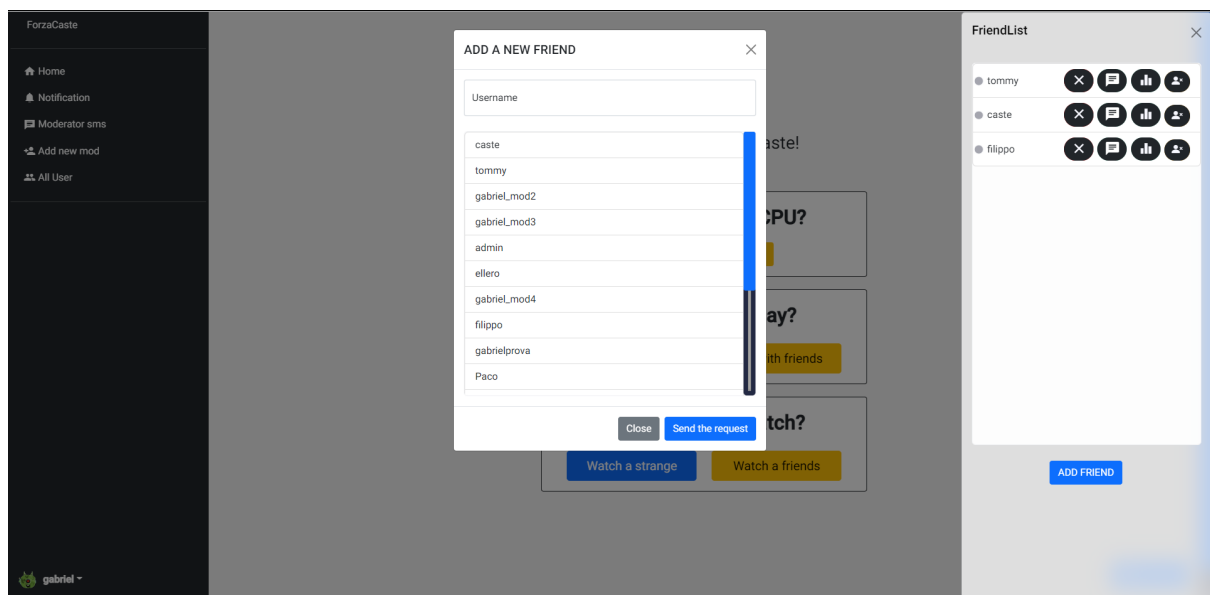
This canvas displays your friends list, indicating your online friends by means of the dot on the left of their names and giving you access to the functions associated with it, such as the possibility of deleting a friend, the possibility of chatting with a friend, the possibility of viewing a friend's statistics and the possibility of blocking or unblocking a friend. It is also possible to send friend requests to existing users within the application.

## Friend Chat



Allows messages to be exchanged between two friends

## Add Friend

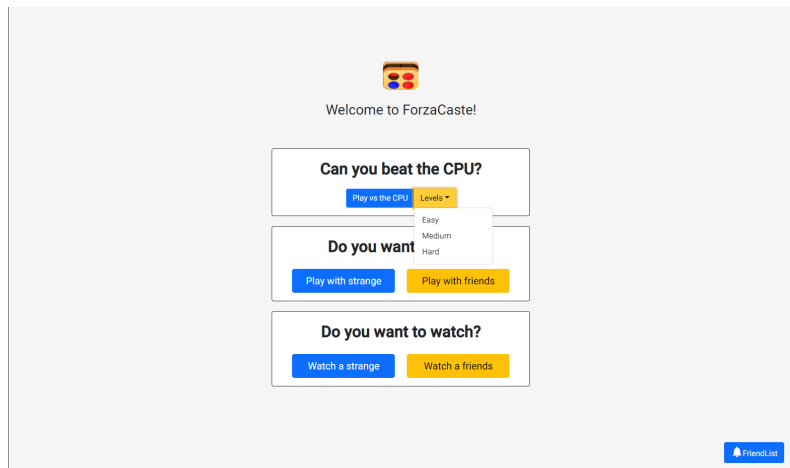


The list of existing users is also displayed.

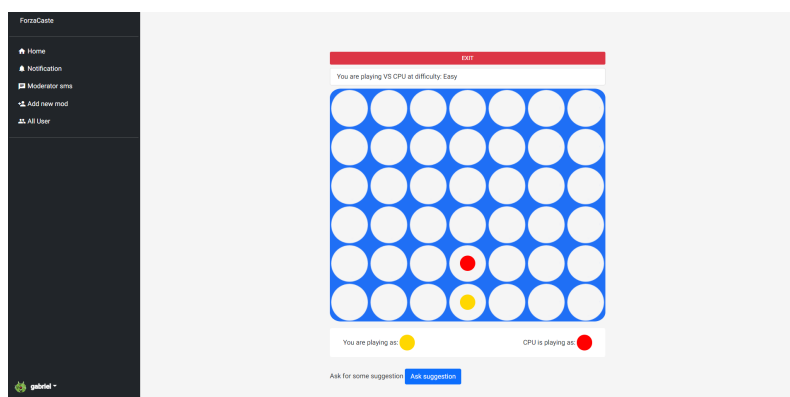
## Game

## VS CPU

You can choose the



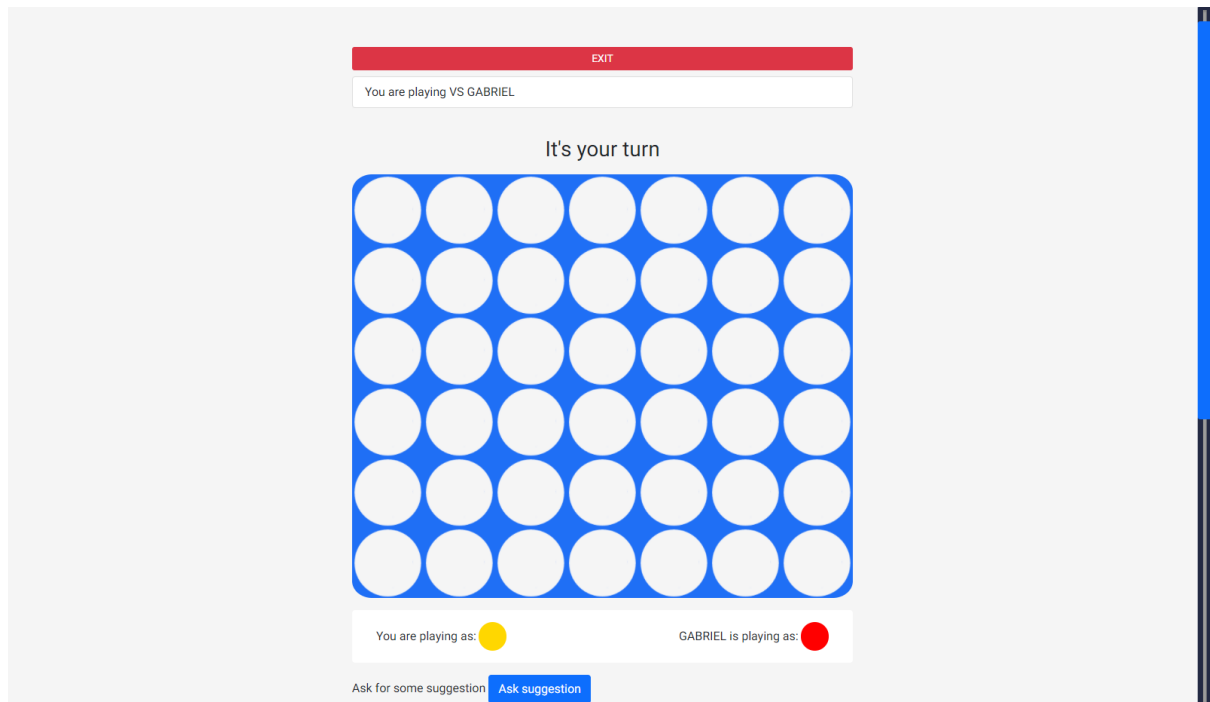
difficulty of the cpu you want to play with, choosing between 3 levels Easy, Medium and Hard.



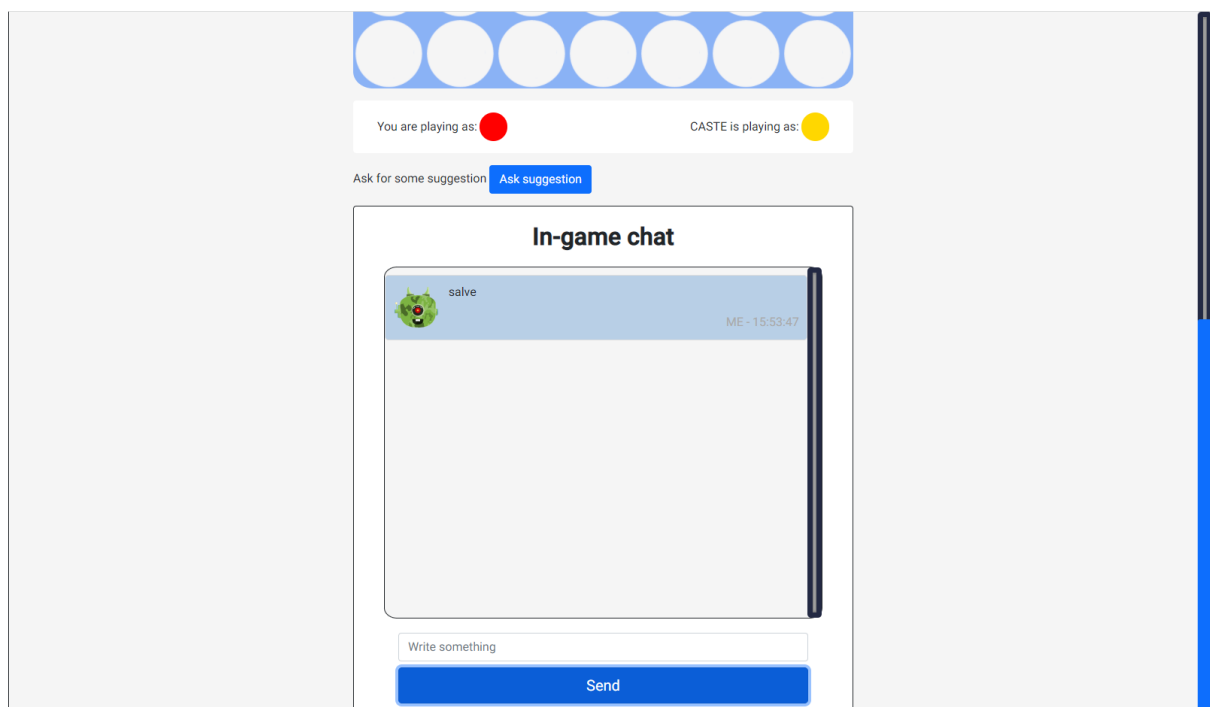
The view opposite shows the visualisation of the playing field, and the possibility of asking for suggestions on the next move.

## VS Friend or Strange

If you play with a friend, you are given the option to choose which friend to play with from those online. If you play with a stranger, you will have to wait until you find another user who is looking for a game (in this case we try to get two people with a similar ranking to play).

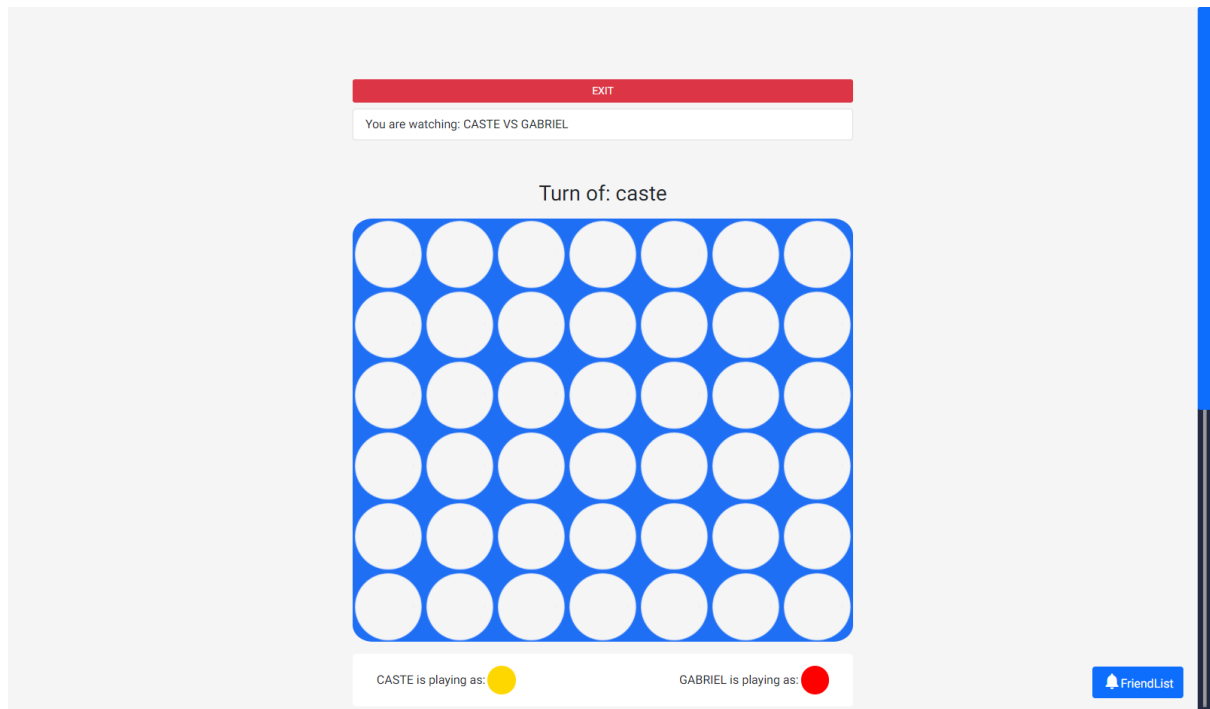


In both cases, a chat room is provided to allow communication between players.

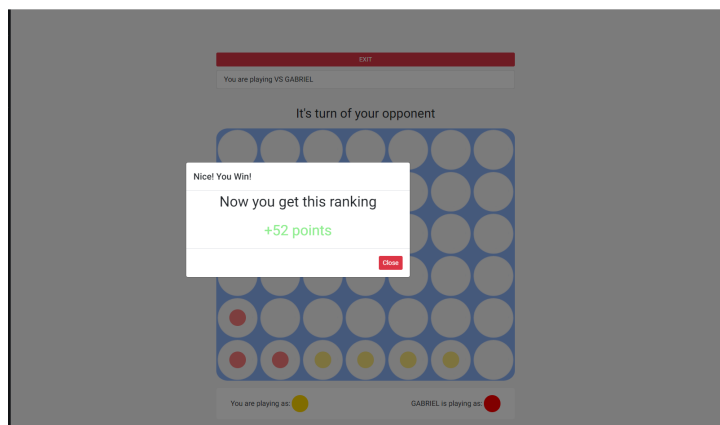


## Watch a game

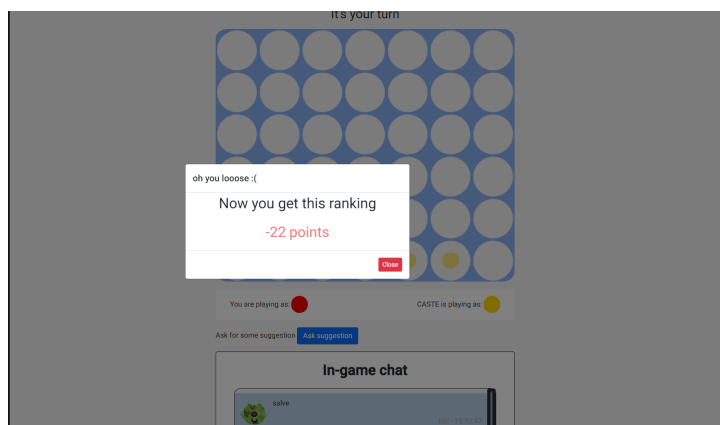
Observers of a game will have the possibility to access ongoing games by choosing among their friends (those who are in the game) or to access a random game. They will also have access to the game chat.



## End-of-game screens

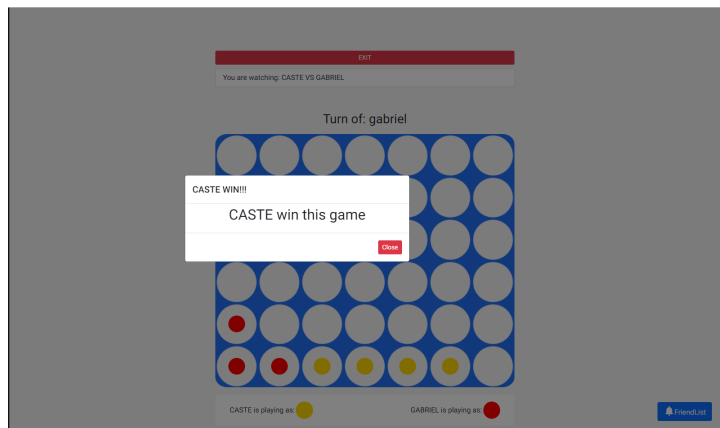


Victory screen



Defeat screen





End of game screen  
displayed by observers