Tommaso_Pellizzon_880772

Introduction

The web application allow users to play the **Connect Four** game against another user, that can be a friend, or a random user. In addition permit to play against an AI. Besides the game logic, the application allow an user to add, delete, manage and block friend and also allow to chat with friends.

Architecture

The application is created using a **docker container** and interacting continuosly with **GiHub**, in order to keep track of all code update. The docker container contains within it a **Node.js** 12.22 image for the development of the backend part, **npm** for manage JavaScript modules and libraries, **TypeScript** compiler in order to write the code in TypeScript language and then automatically the compiler will translate it in JavaScript language and **Angular** CLI version framework, used for the development of the frontend side.

Other tools used are **Postman** used in the server side development to simulate HTTP requests and Socket.IO messagges and to test the correct functioning of the server logic. Then the application is based on the use of MongoDB database, a non-relational database, in order to store all the information about users, matches, messages and notifications. The database is hosted by MongoDB Atlas and to access the database we have used MongoDBCompass. Finally it has been used Heroku for the deploy of the application.

Let's now see the tecnologies used for the development of the two part: server-side and client-side.

As mentioned above the server-side, based on REST-style, is developed using Node.js and TypeScript and then:

- **Colors.js** used to color the console logs, in order to prettify the server console
- **Mongoose**, a driver used to connect with the MongoDB database, that provides to define data models, as objects, which represent the database documents, and provides functions to perform action on database objects
- **Express.js**, middleware used for the management of the HTTP request, e.g. manage the routing between endpoint and APIs and provides functions that read and write the request and response objects sequentially
- **body-parser**, middleware used for the parsing of the HTTP request body. So the incoming HTTP requests that contain body, are parsed first, and then deliverd to the web server ?

- **passport.js**, middleware used to implement the authentication logic of the web app. In particular provides a set of strategies supports authentication using username and password
- **JSON Web Token**, standard used to manage token-based authentication. Particularly allows to manage the JWT
- **Cors**, package used to enable and manage cors
- **Socket.IO**, a JavaScript library used for realtime web applications. It enables and provides all the functions for realtime, bidirectional and asynchronous communication between web clients and server
- **bcrypt**, a library that helps to hash passwords
- **Ajv validator.js**, a library for string validators and sanitizers. It is used for validate and check the correctness of the attributes of objects used in the web app

Regarding the client-side is implemneted as a **SPA** (*Single Page Application*), using:

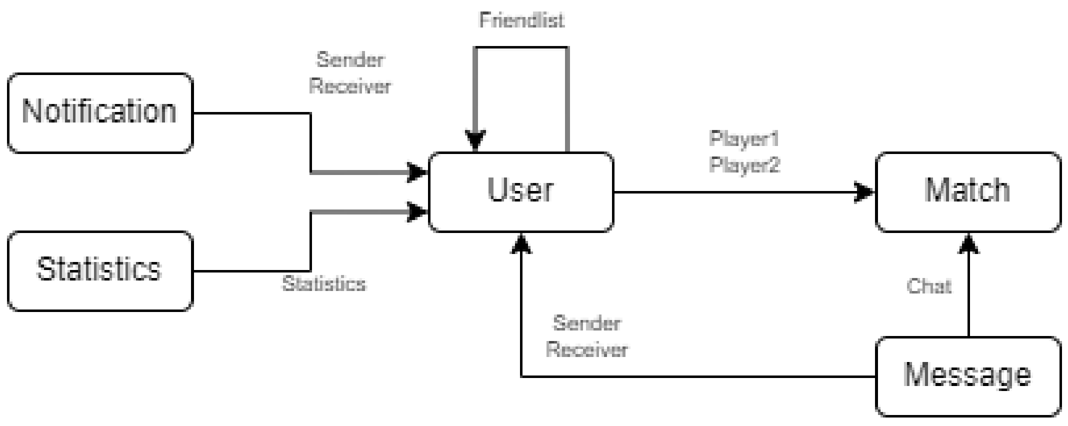
- **Angular**, a front-end framework for developing SPA web clients
- **Bootstrap**, an HTML, CSS and JS library that simplify the development of web pages
- **Angular Material**, a UI component library, that we have used in our web app
- **ng-bootstrap**, a repository that provides Angular widgets, that we have used in out web app
- **ng2-charts**, a library used for chart creations. In our web application is used for creating the charts about the users statistics
- **mat-icon** e **bootstrap icon**, for the icons used in the web app

Data Model

In the diagram below are shown the 5 entities used:

- User
- Notification
- Statistics
- Match

- Message



It follows the data models representing the entities defined before.

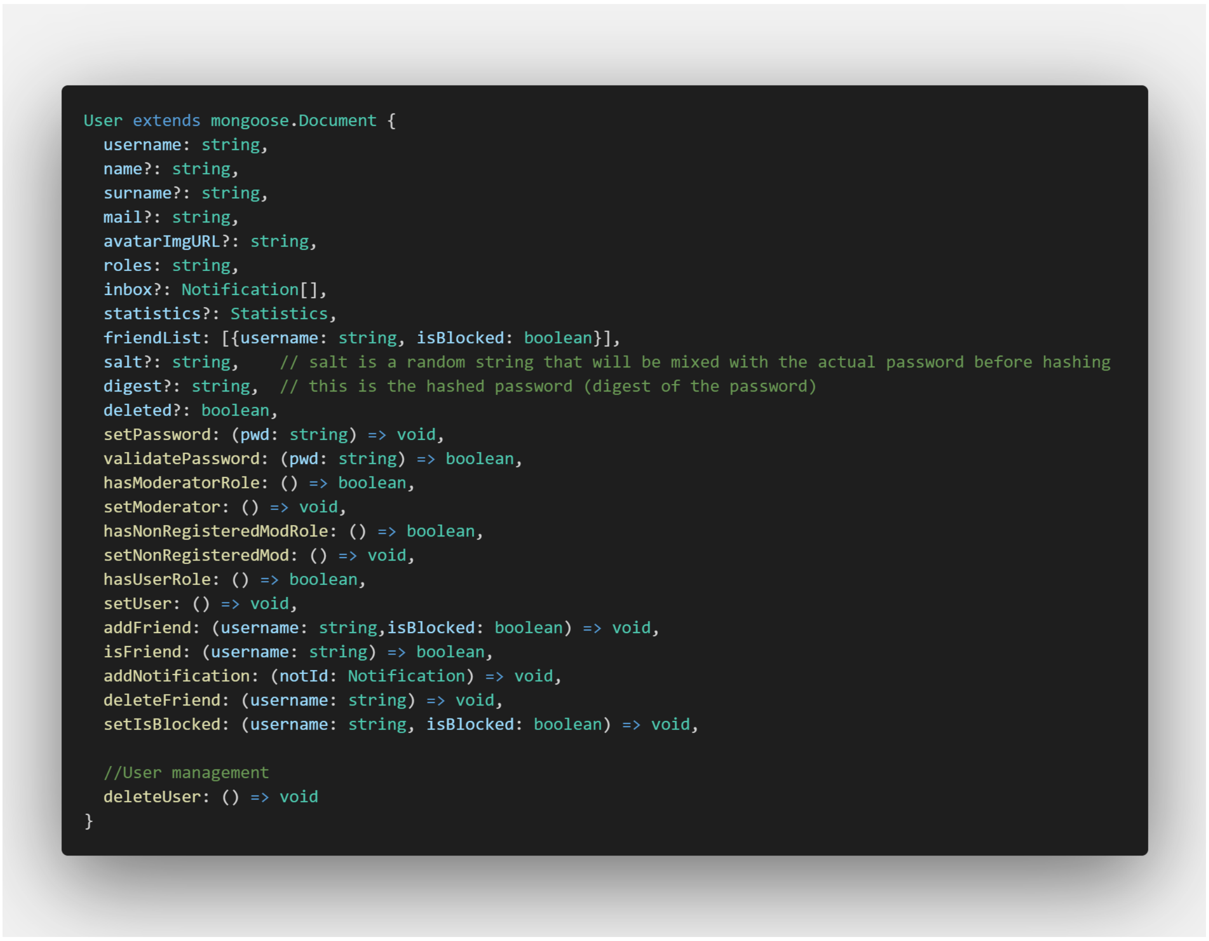
Legend of some characters:

- `?` means that the attribute is optional
- `[]` means that the attribute is an array, so it can contain a collection of the specified elements
- `{}` means that the attribute is an object
- `method (a: type, ...) => type` encode a method belonging to the entity

User

User document that represent the user of the web application. It contains all the usefull information about an user and all methods to read and write the attributes.

The group decided that each user is identified by the username, so we applied a unique index to this attribute.



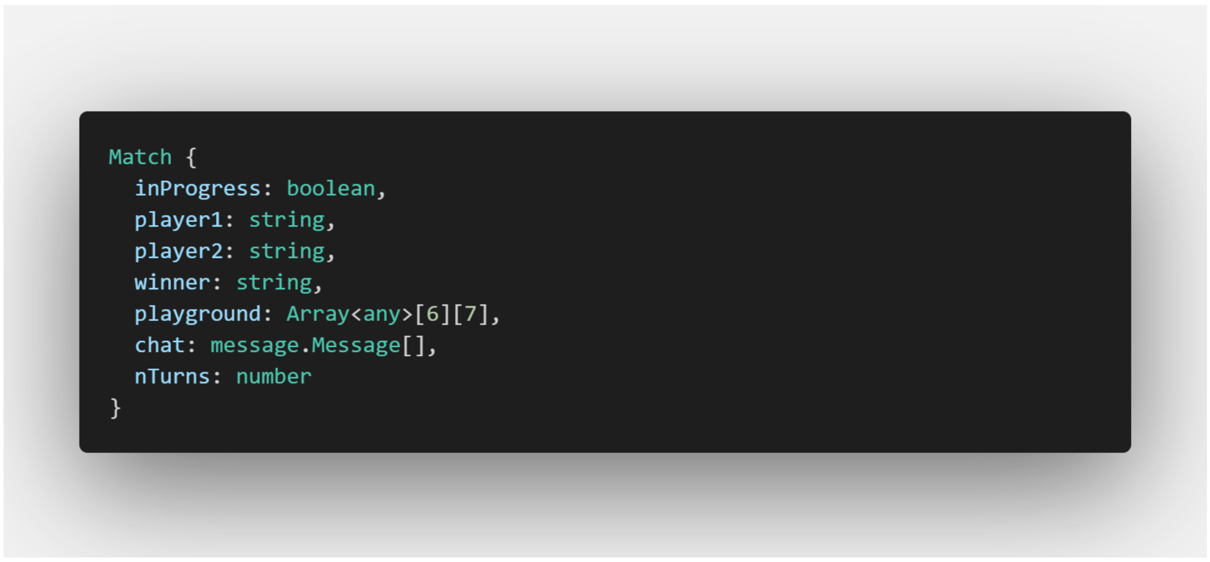
```
User extends mongoose.Document {
  username: string,
  name?: string,
  surname?: string,
  mail?: string,
  avatarImgURL?: string,
  roles: string,
  inbox?: Notification[],
  statistics?: Statistics,
  friendList: [{username: string, isBlocked: boolean}],
  salt?: string,    // salt is a random 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
}
```

Note that is stored the digest or rather the hashed password, and not the clear password.

Match

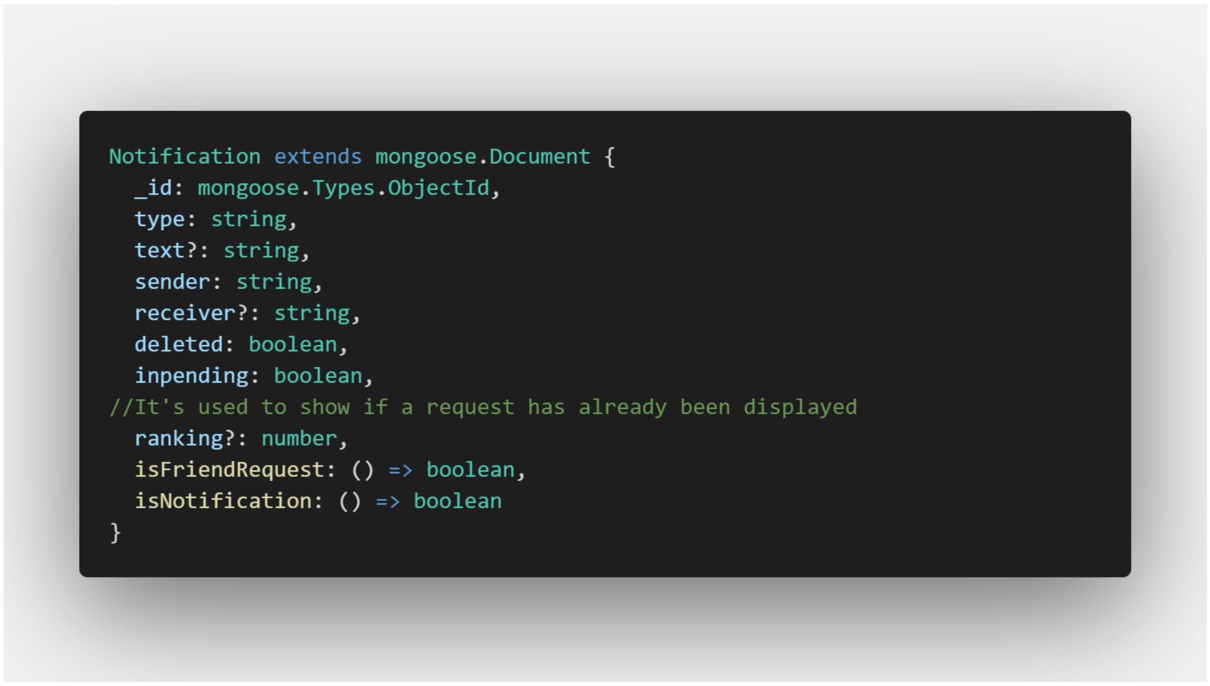
The match document represent a match in progress (`inProgress = true`) or a match played. It contains all the information about a match and the chat of the match, that is an array containing all the messages that have been sent during the match.



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

Notification

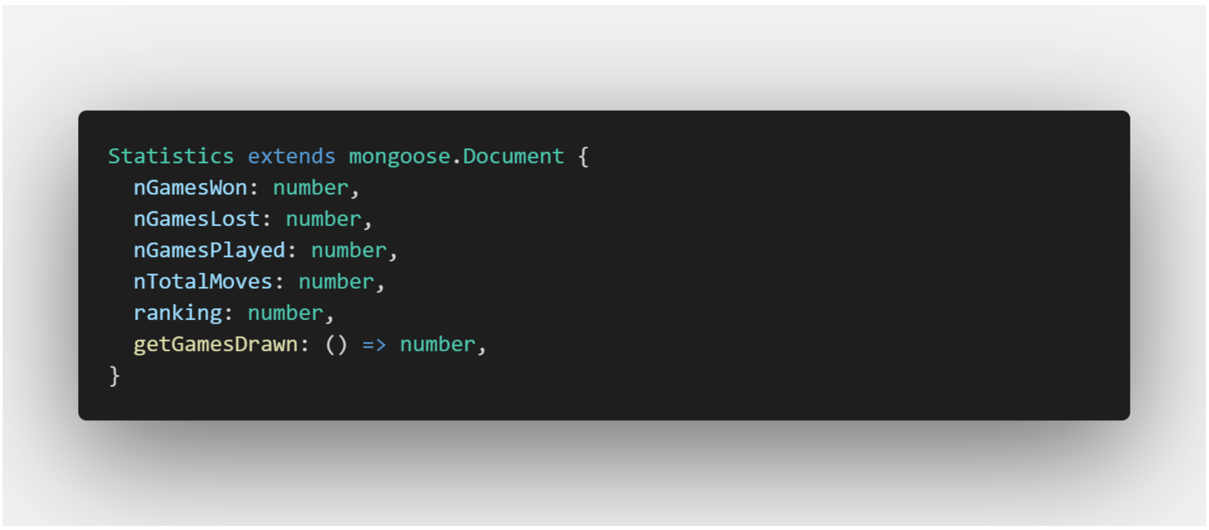
The notification document is used for friend requests (`type = 'friendRequest'`), match request against random user (`type = 'randomMatchmaking'`) or match request against a friend (`type = 'friendlyMatchmaking'`).



```
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
  ranking?: number,
  isFriendRequest: () => boolean,
  isNotification: () => boolean
}
```

Statistic

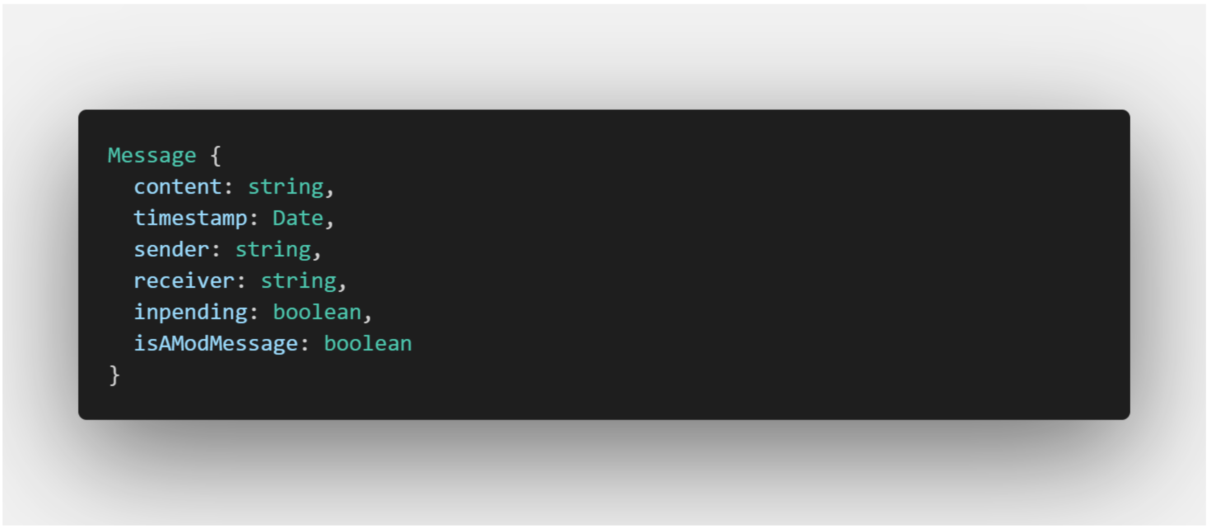
The statistic document is used to keep track of information about the match played by the user.



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

Message

The message document contains all the information necessary to store and manage the messages. In order to notify the user that he has unread messages yet, we have introduced the `inpending` attribute, so the server always knows which messages are still unread.



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

REST API documentation

A more detailed documentation is present in the file `endpoint_documentation.pdf`, available in the project folder.

It was created using Postman, that allows users to document HTTP request. So in that file you can find a description of the endpoint, the possible return values, the Socket.IO interested in a possible communication and the body or the parameter included in the request.

Endpoint list

As mentioned above, the backend has been developed following the REST API guidelines. The endpoint used are proposed here.

Endpoint	Method	Attributes	Description
/	GET		Returns the API version and all the available endpoints
/login	GET		Login a user. Returns a JWT
/whoami	GET		Get the information of the logged user. Refresh the JWT of the user
/users	POST		Sign up a new user
/users/online	GET		Returns the online users, so the users that have logged in
/users/:username	GET		Returns all the information of the user with the specified `:username`
/users	GET		Returns all the existing users
/users/mod	POST		Sign up a new moderator user
/users/:username	DELETE		Delete the user with the specified `:username`
/users	PUT		Update a user’s information
/rankingstory	GET		Returns a list containing the rankings of the logged in users
/rankingstory/:username	GET		Returns a list of the match played by the user with the specified `:username`
/game	POST		Create a random or a friendly match. It is also used for the observator to join a match

/game	PUT		Accept or refuse a friendly match request
/game	DELETE		Delete a game started or a match request sent. It is used by a player if he has to delete the game, or by the creator of the match request if he has to delete it.
/game	GET		Returns the games in progress
/game/cpu	POST		Create a match against the CPU. It is also used for the observator to join a match
/move	GET		Ask to the AI the best move to play. Returns the column on which to make the move
/move/cpu	POST		Play the turn of the player against AI. It also informs all the observator of the move played. Finally performs all checks to see if there is a winner
/move	POST		Play the turn of the player. It also informs all the observator and the other player of the move played. Finally performs all checks to see if there is a winner
/gameMessage	POST		Send a message in the game chat
/notification	POST		Create and send a friend request to another user
/notification	GET		Returns all the notifications received by the user
/notification	GET	?inpending = true	Returns all the

			notifications of the user which have not yet been read
/notification	GET	?makeNotificationRead=true	Returns all the notifications of the user which have not yet been read and marks all the notifications as read
/notification	PUT		Indicates that the unread notifications of the user have been read
/friend/:username	DELETE		Delete the friend with the specified `:username` from the friend lista of the user
/friend	PUT		Block or unblock a friend
/friend	GET		Returns the friendlist of the user
/message	GET		Returns a list of all the messages the user is involved with and another list of the unread messages received
/message	GET	?modMessage=true	Returns a list of all the moderator messages the user is involved with and another list of the unread moderator messages received
/message	POST		Send a message to a friend
/message/mod	POST		Send a moderator message to an user
/message	PUT		Indicates that all the unread messages received have been read

Responses

The possible HTTP request responses that a client can receive are

Status code	Error message
200	SUCCES, means that the required operation has been performed corretctely
400	BAD REQUEST, The server cannot or will not process the request due to something that is perceived to be a client error
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

Available socket listener

As mentioned above for the development of the backend we also used Socket.IO for the communication of the client and server. In particular we used it for the asynchronous communication from the server to the client. The server use a list containing the sockets of all clients, in this way for the server is easier to retrieve the socket of a client to send a message.

Furthermore we used the Socket.IO rooms for the match to make it easier to send a message to a group of user. The rooms are used for the matches: for each match we create 2 rooms, one in which everyone, players and observators, partecipates and where are sent moves and players messages, and then another room used only by the observator where are sent the observator messages, so only observators can read observators messages.

Communications that used Socket.IO are unidirectional and they go from server to clients, except for the connection and the disconnection. This because when a client need to interact with server, send an HTTP request. Whereas, to make all more dynamic, the client is listening for events in which the server send messages. The server send messages using `client.emit('eventListener', message)` where client is the socket of the client.

Follow a list of the event listener used

Event	Body	Description

Event	Body	Description
online	{username: username, isConnected: bool}	Informs the user that he is connected, if `isConnected=true`, or if it is disconnected if `isConnected=false`
updateUser	database user document	Informs all listeners that a user has been updated, sending the user information
gameRequest	{type: “friendlyGame”, player: “username”}	Informs the user that a friend has sent him a game request. The user who sent the request is specified with the `player` field
gameReady	{gameReady: true, opponentPlayer: “username”}	Informs the creator of a game request that has been accepted
move	{yourTurn: bool}	Informs a player if it’s his turn, `true`, or if it’s opponent’s turn, `false`
move	{error: bool, codeError: `number`, errorMessage: “”}	Informs the player that made the move that it is not valid, so he will have to replay it
move	{move: index}	Informs observators the move that has been played, specifing the column
gameStatus	{playerTurn: “username”}	Informs observators whose turn it is at the start of the game, specifing the player’s username
gameStatus	{player: “username”, move: index, nextTurn: “username”}	Every time a move is played, observators receive this message that informs them of the player who made the move, the column where the move has been performed and the player of the next round
enterGameWatchMode	{playerTurn: “username”, playground: “”}	When a user enter in a game as observator, he receive this message that informs him of the status of the match: whose turn it is and the playground, so where the disks were placed

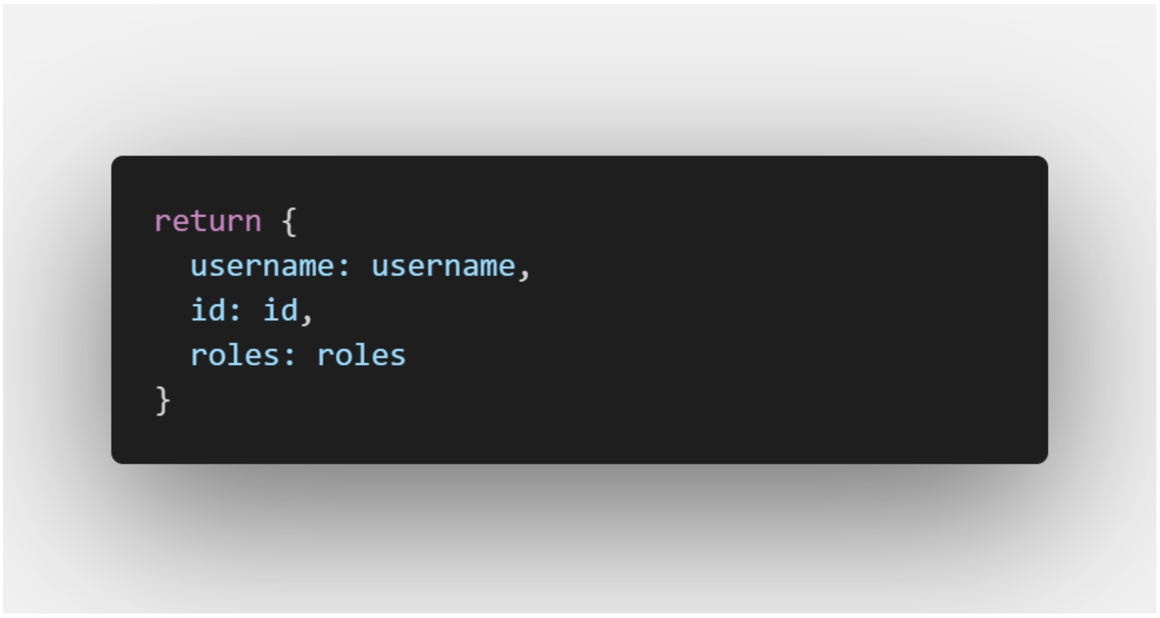
Event	Body	Description
result	{winner: “username”, message: “Opposite player has left the game”}	Informs a player that the opposite player has left the game
result	{winner: bool/null}	Informs players of the result of the game, the player has won if `true`, the player has lost if `false`, otherwise they tied if `null`
result	{rank: rank}	Informs the players about the ranking obtaind from the match
result	{winner: “username”}	Informs observators of the winner of the match
gameChat	database message document	Informs all the users involved in a match that a message has been send into the game chat. The message contains all the information of the message
newNotification	{sender: “username”, type: “friendRequest”}	Informs a user that another user has sent him a friend request. The user that has sent the request is specified with `sendere`
request	{newFriend: “username”/null}	Informs the sender of a friend request that the user with `newFriend` username has accepted the request or if a friend request has been refused
friendDeleted	{deletedFriend: “username”}	Informs a user that a friend has deleted him as his friend
friendBlocked	{blocked: bool}	Informs a user that he has been blocked from another user (`true`), or if he has been unblocked (`false`)
message	database message document	Informs user that he has received a private message from a friend

You can find all listeners in more detail in the file `endpoint_documentation.pdf`, available in the project folder.

Authentication

JSON Web Token

As said previously we used **JWT** to manage the authentication phase. We decided to include in the JWT useful data such as the username, which allows to uniquely identify a user, the database user identifier and the role, so in this way the frontend can build the user view dinamically based on the user role.



```
return {
  username: username,
  id: id,
  roles: roles
}
```

Login

The authentication phase is implemented with the Bearer Token method, with the use of JWT, except for the initial login where is used the Basic Authentication. When a client send an HTTP request, he must include the JWT received and generated previously from the server. In order to get the JWT, a client need to login, so the server will send the token.

The first step of authentication involves the Basic Authentication of the client. So when a client wants to log in, he send the login request to server, specifing his username and his password. On the backend side this phase is managed by **passport.js**, middleware responsible of defining the steps to perform the Basic Authentication. So the server will check the received credentials with those stored in the database; not before having encrypted the password using the bcrypt library. If the credentials check, so the Basic Authentication, returns an affirmative response, then the server build the token, containing the information mentioned above. Then is necessary to sign the token and for this purpose is used the *jsonwebtoken* library which allows it through a secret key, which is stored in the server environment variables. Furthermore to the JWT is applied an expiration time, in order to sign it and verify that it is still usable. The expiration time we decided to use is one hour. In the end the token is send to the client.

In case Basic Authentication fails, no token is generated, and an error message is returned as response to the client.

Using HTTP sending the JWT is a critical phase: it may be steal and use to impersonate the user. We have resolved this problem using HTTPS.

Token refresh

We said that the JWT has an expiration time of one hour, so when this expires the user must log in again. Since we didn't like this solution, we have found a way to refresh the token and extend its expiration time. We implemented it introducing an endpoint that checks the token expiration time: if it is close to the expiration, a new JWT is generated, as a copy of the old one, and then it is returned to the client.

Socket.IO connection

The authentication phase involves another important action: the Socket.IO connection of the client to the server. When a client log in, it connects its Socket.IO socket to the server using the functions provided from the library. In the authentication phase, when the client try to connects his socket to the server, it is used his JWT to get the client information in order to save his socket in the dictionary containing all the sockets. As said previously the dictionary is used to keep the mapping between users and sockets.

Frontend

The code which implement the frontend side is situated inside the 'taw' folder. The folder is organized in such a way that the sub-folder 'src' (folder for sources) contains every graphical components, services and the router component.

For the development of the graphical interfaces we decided to use **Bootstrap v5** in order to make the web application responsive, **ng2-chart** for chart creation and management, **ng-bootstrap** for toast management, **Angular Material** for the badges, while for the icons we decided to use Bootstrap and Angular Material icons.

Services

We created and we used 3 services. They are explained below.

Toast service

Toast service is located in toast.service, inside _services folder.

The toast service allows to display to the user any type of notification he received. For the most part, notifications are received into the Socket.IO event listeners, so when a listener receive a message from the server, the toast service is used to notify to the user that an event has been received.

Toast service provides 2 methods:

- `show`, to make toasts appear
- `remove`, to make toasts disappear

Socket.io service

Socket.io service is located in socketio.service.

The Socket.io service is used to transform Socket.IO events into observables which a component can subscribe to in order to obtain information. Since Socket.IO communications are used only for communication from server to client, the Socket.io service provides all methods for creating and managing sockets that allow the client to receive messages from server.

User-http service

User-http service is located in user-http.service.

The user-http service is used to interact with the server, so to send HTTP request. The requests that need to be sent and that must include the JWT, are handled by the user-http service, in fact it is its task to insert the JWT in the HTTP request. Furthermore awaits for server responses and when he obtain a response, he transform it into observables, then used by components to retrieve data.

The token can be store in:

- localStorage: the token has visibility in all browser tabs
- sessionStorage: the token has visibility only in that session and reloading the page results in the loss of the token

Components

All-userComponent

The component is used only by moderators. It provides a list of action that the moderator can perform, such as:

- Delete a user
- Get a user profile, to show all his information
- Chat with anothe user, so it provides methods for get and send messages

CPUComponent

The component is used to manage user action that involve the AI on the server. For example create a match against the CPU, make a move in a game against the CPU and finally request the suggestion of a move, which is asked to the AI.

FriendChatComponent

The component allows the user to use the private chat with a friend. So it provides to read, send and it contains methods to notify the user a new message received.

FriendStatsComponent

The component is used to show the user the profile, so information and the ranking, of a friend.

GameComponent

The component provides all features to implement the game logic and view. It allows to show the playground, whose turn it is, which pawn belongs to which player, the chat and all messagges that can be displayed, such as winner, draw or quit of a player and the ranking point earned or lost.

In addition provides methods to play a move, to ask a suggestion of the move to play and to receive, read, write and send messages.

HomepageComponent

The component consists of the control center of the web app. So it allows to play a match against a random user, send a game request to a friend, play a match against the CPU, watch a friend’s game and access to the friend list.

In particular for the match against the CPU allows the user to choose the difficulty level.

ModChatComponent

The component is used for the chat between a moderator and another user: it allows to show all messages and to send new messages.

NewModeratorComponent

The component permit to create a new moderator user, specifing his credentials.

SidebarComponent

The component implement the sidebar of the web application.

The sidebar mainly offers to the user two functions:

- A space where the user can access to all the received notifications and methods to accept or refuse them
- The friend list with all actions that can be performed on friends, such as:
    - Delete friend
    - Access the chat with the friend
    - Viewing friend profile
    - Block or unblock friend

ToastComponent

The component defines the structure and behaviour of the toasts that are displayed to the user.

UserLoginComponent

The component implements the start page, where the user can log in or go to the sign in page.

In particular it defines the method for the login that include the Socket.IO connection to the server.

UserLogoutComponent

The component allows the user to logout, invalidating the JWT. Furthermore it provides the disconnection of the Socket.IO socket.

UserProfileComponent

The component allows the user to access to all his information and change them. In the information that can be visualized there is the statistics of the game, such as the game won, lost and draw.

UserSigninComponent

The component is used to create a new user. So it ask to the user to insert his personal information and then it create the account and save all information into the database. Then it redirects the user to the login page.

WatchComponent

The component provides the same functionalities of the GameComponent but for the observer, therefore it does not allow to play a move, but only to watch the game, so it has the task of update the playground. Furthermore it allow the user to use the chat, so to read and send messages into the game chat.

Routes

Since Angular is based on the use of a *SPA*, it is necessary a module to implement the mapping between URLs and components; in this way when a client make a request specifing a URL, is possible to retrieve and render the right component.

The module that does this is the router module and it is located in the app-routing.module.

A list of all addresses mapped in components used follows.

Address	Component
	UserLogin
/login	UserLogin
/home	Homepage
/logout	UserLogout
/signin	UserSignin
/profile	UserProfile
/new-mod	NewModerator
/all-user	AllUser
/game	Game
/watch	Watch
/cpu	Cpu
/user-stats/:friend	FriendStats

Address	Component
/friend-chat/:friend	FriendChat
/mod-chat/:user	ModChat
**	UserLogin

Typical application workflow

The most important interfaces and the interaction between them is shown here. They are divided into macro topics, which are:

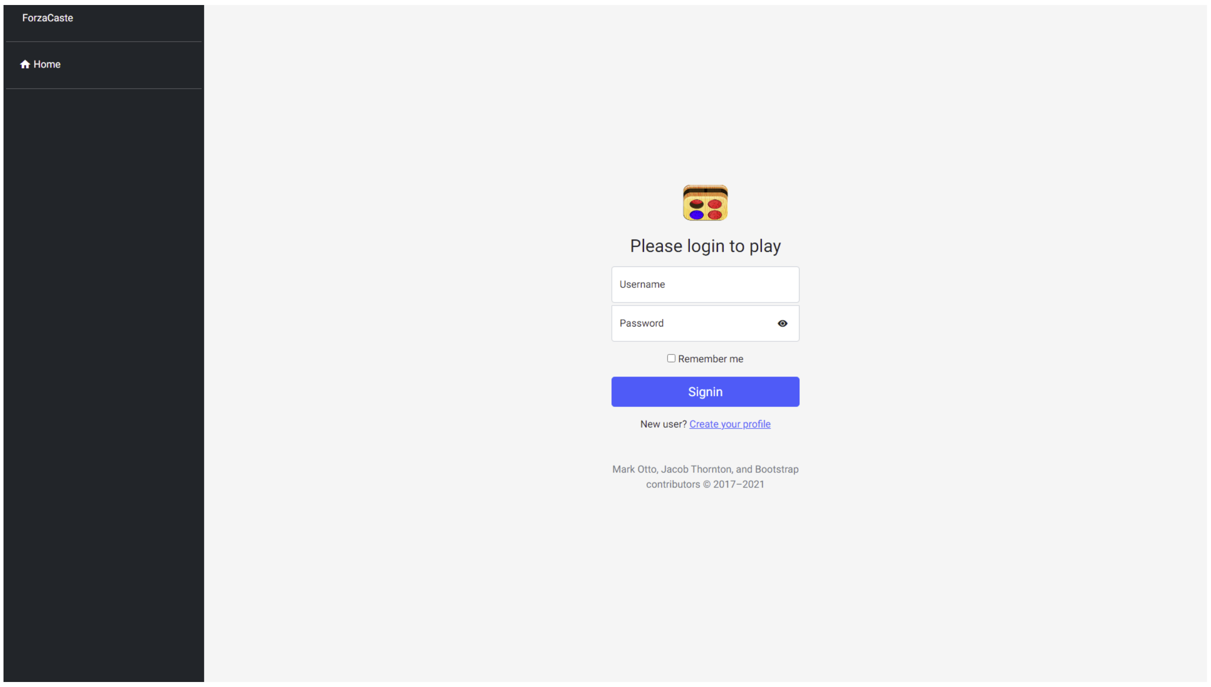
- User
- Game
- Observer
- Friend
- Moderator

User

Includes interfaces for logging in, creating a new profile, homepage and viewing your profile.

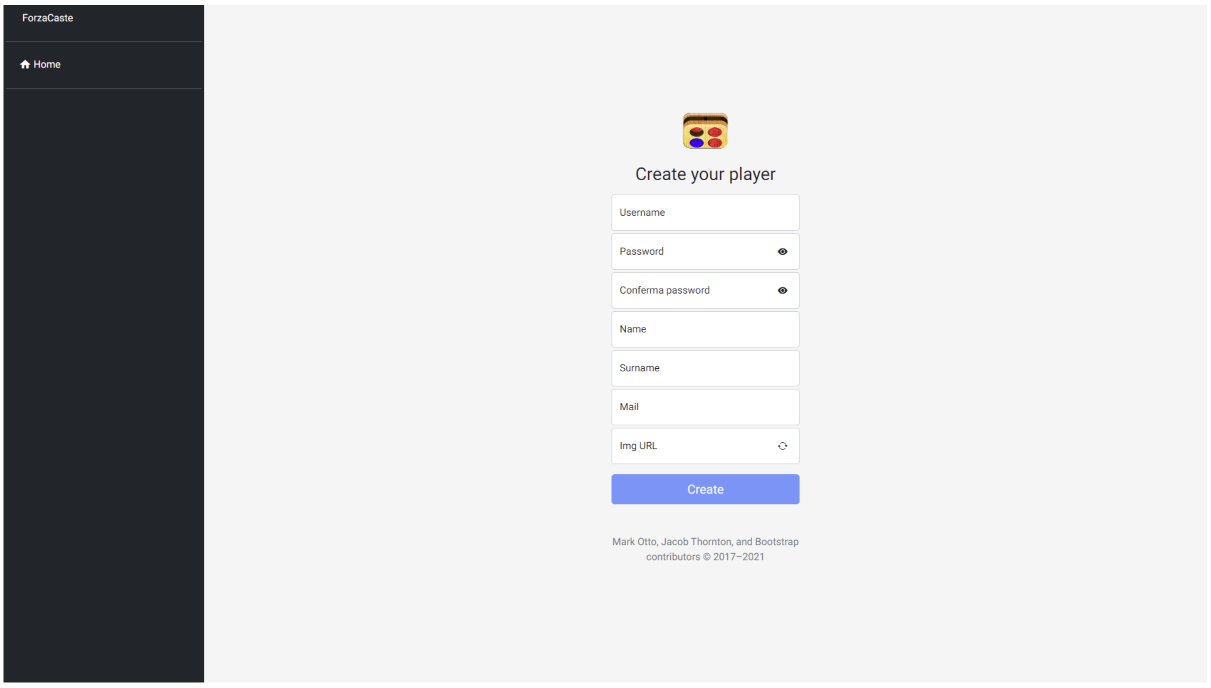
Login

By entering a username and password, the user can access the web application.


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

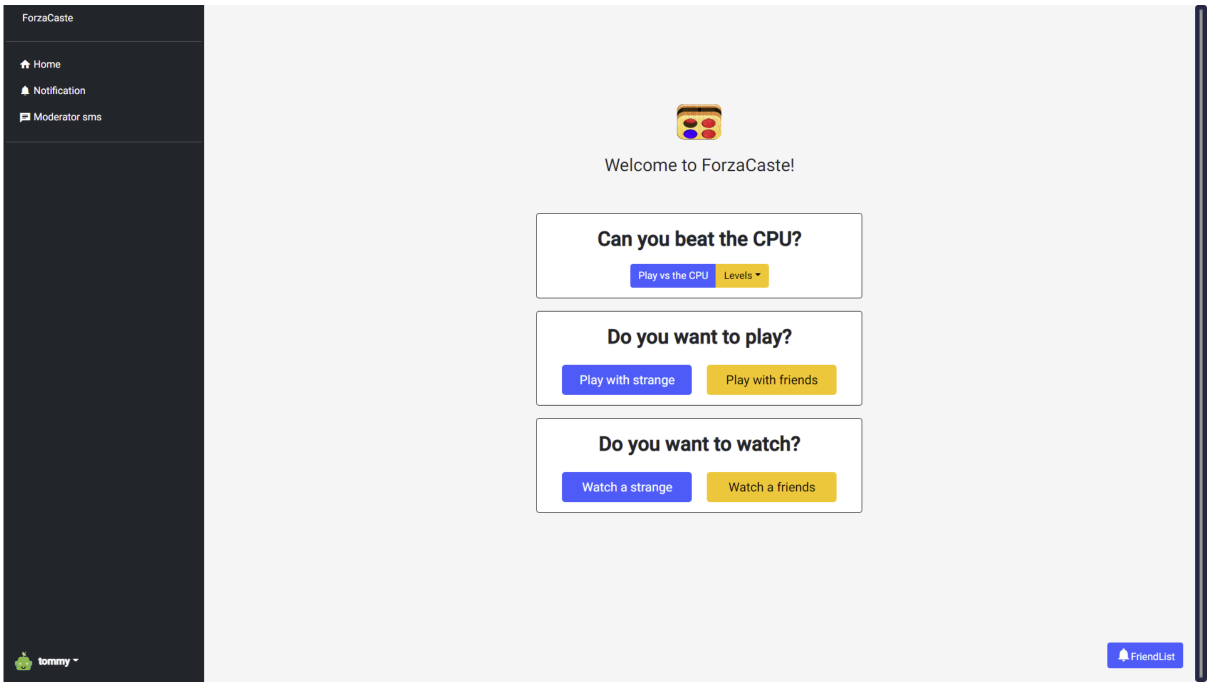
Profile creation

This view allows users to create their own account. They need to choose a username, then they must insert the password twice, to check that it has been inserted correctely and finally the user can enter the URL of an image or, by clicking the appropriate button, he can select a random image.



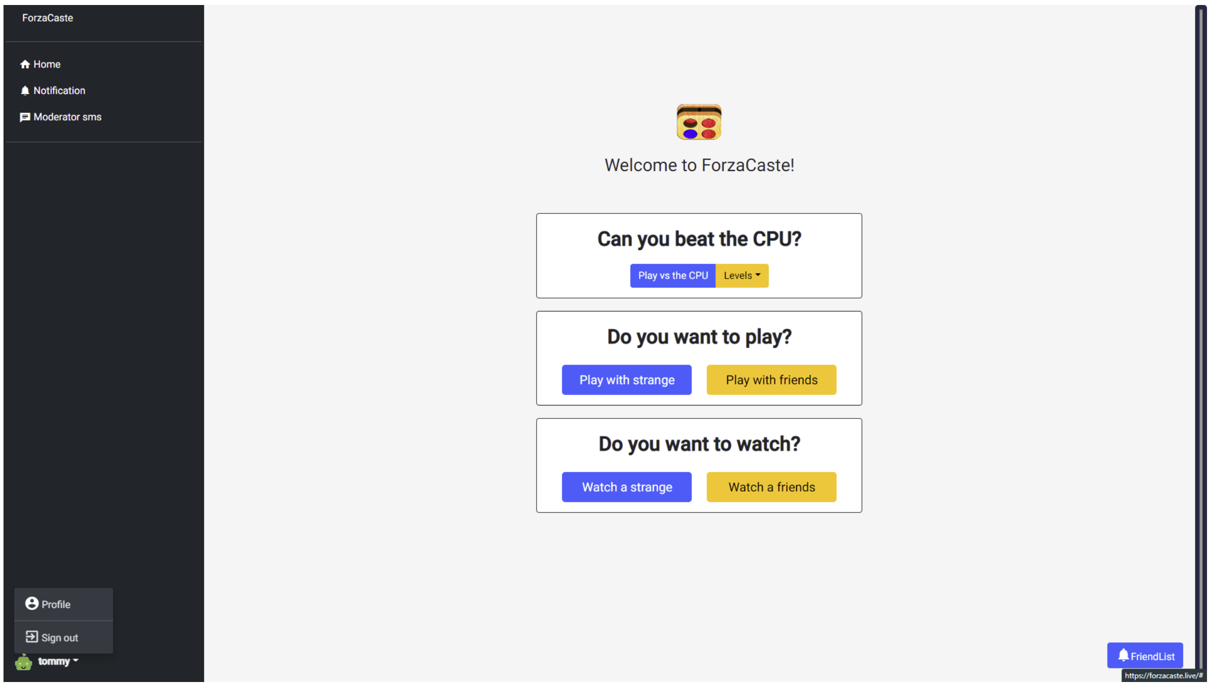
Homepage

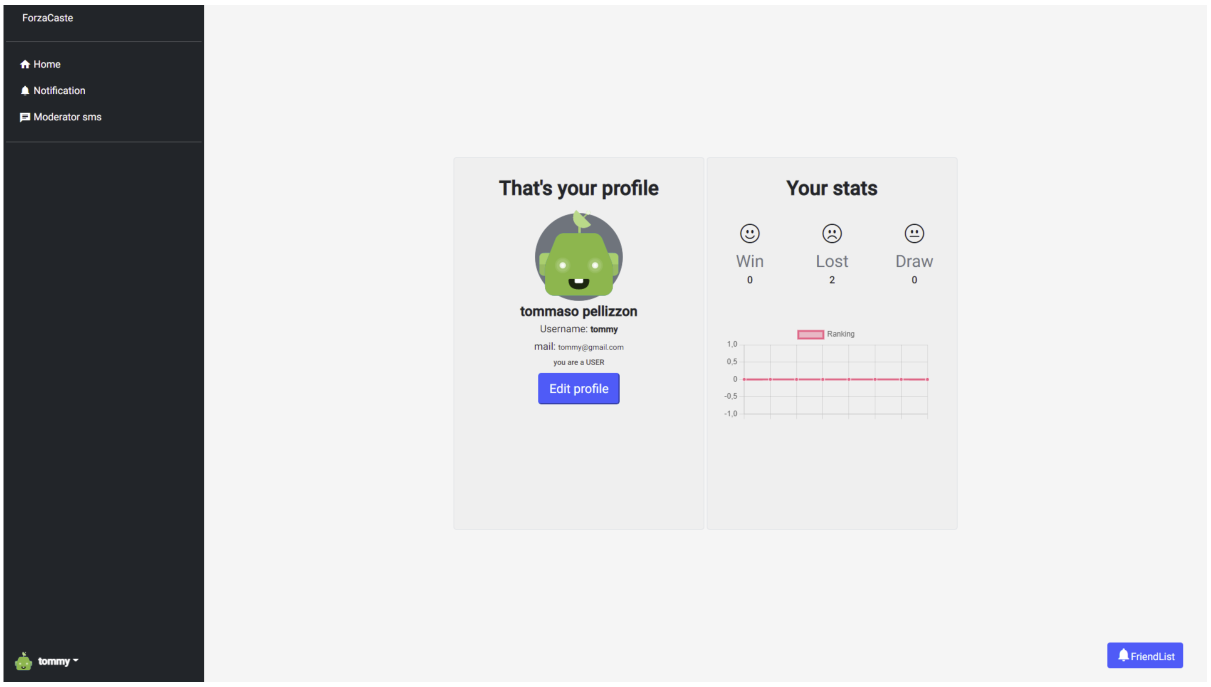
This is the main page, from which you can access all the features offered by the web application.

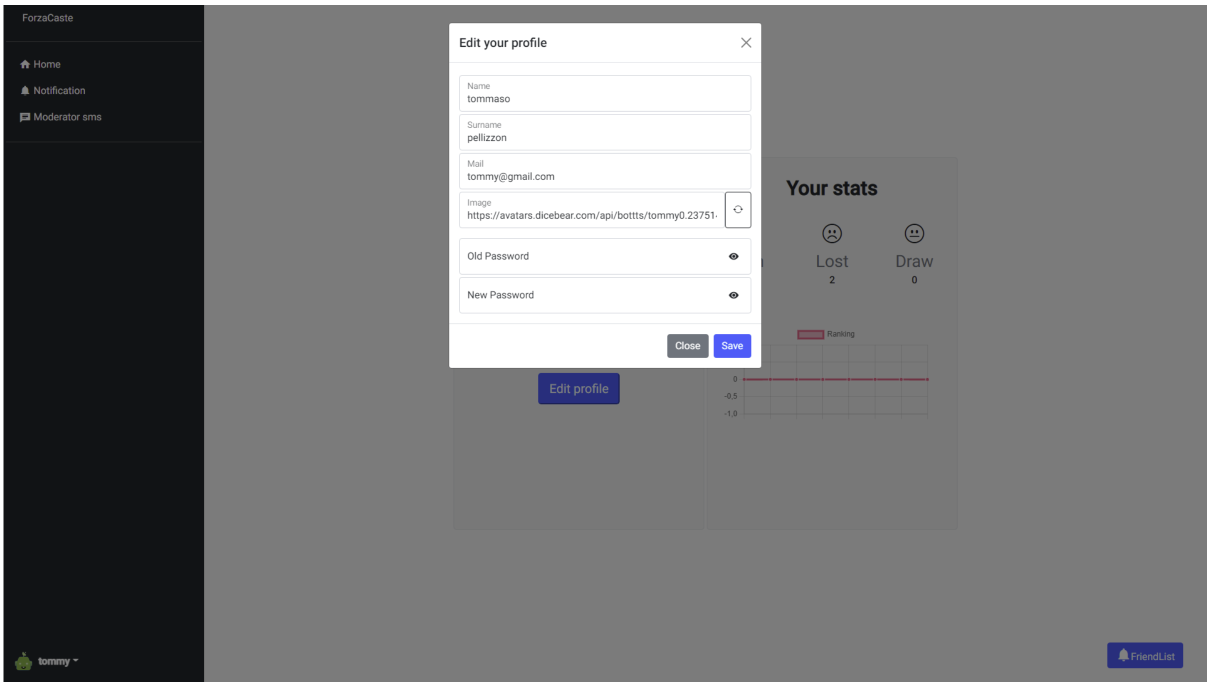


User profile

From the homepage, you can reach your profile page, where you can view your information and game statistics.







Name
tommaso

Surname
pellizzon

Mail
tommy@gmail.com

Image
https://avatars.dicebear.com/api/bottts/tommy0.23751

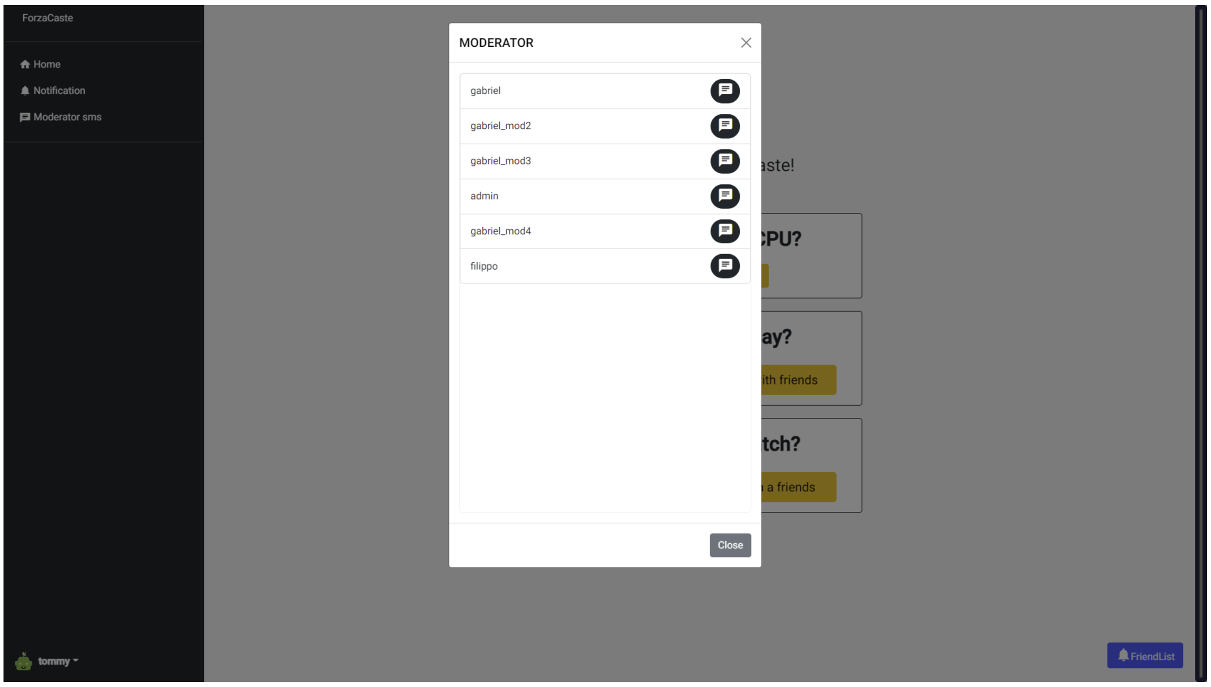
Old Password

New Password

You can also change your account details from your profile page.

Moderator message

From the sidebar the user can access the section to send a message to a moderator. Then through the modal the user can choose who to send the message to.

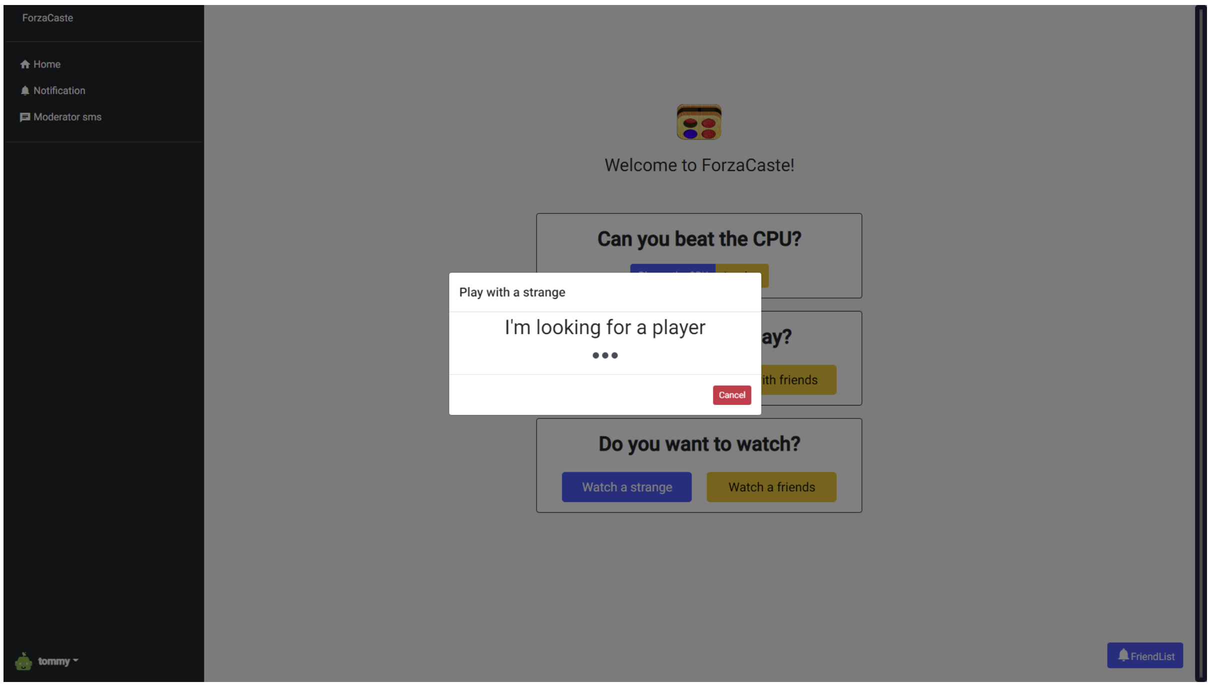


Game

This section includes the interfaces and workflow provided during the game phases, starting from sending a game request until the end of a game, including game chat, suggesting a move and observing a game.

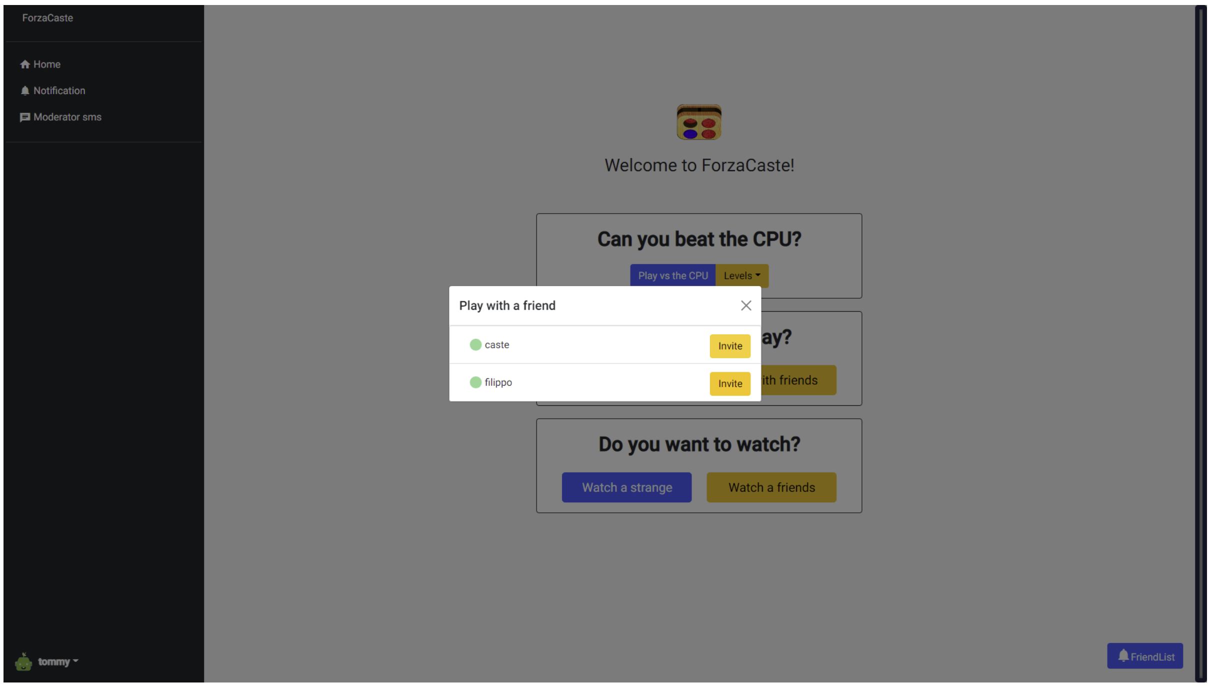
Game against random user

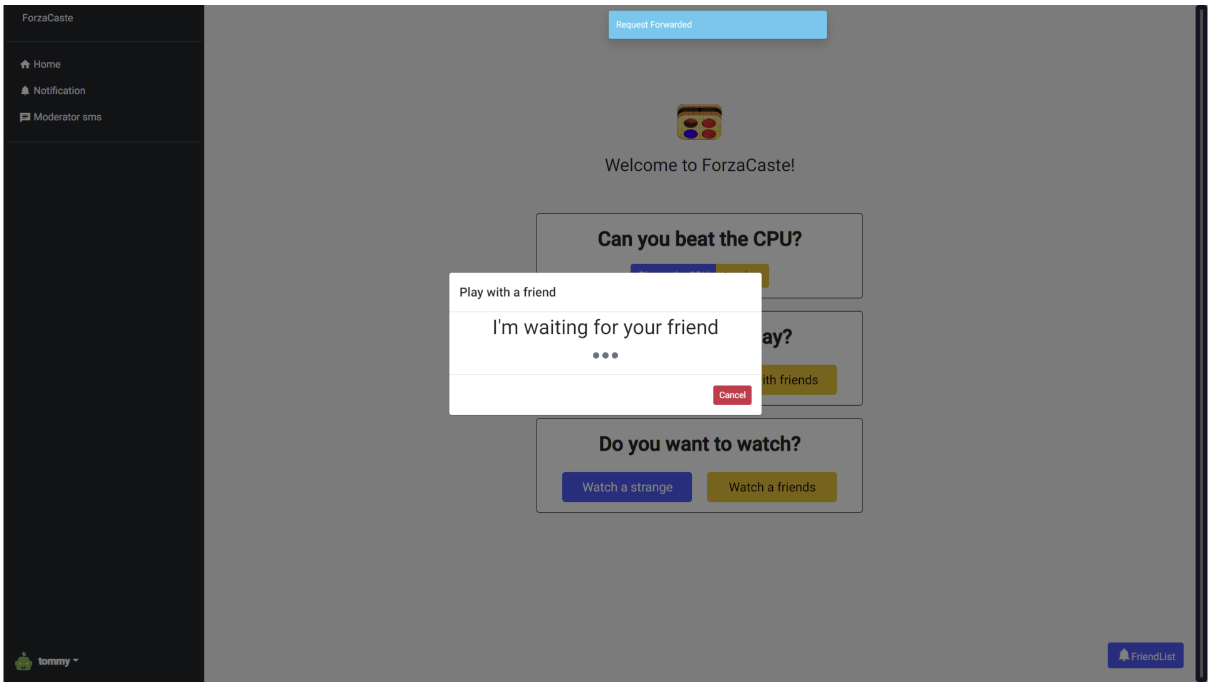
By clicking the ‘Play with strange’ button the user can wait for another player to play a game against a random user.



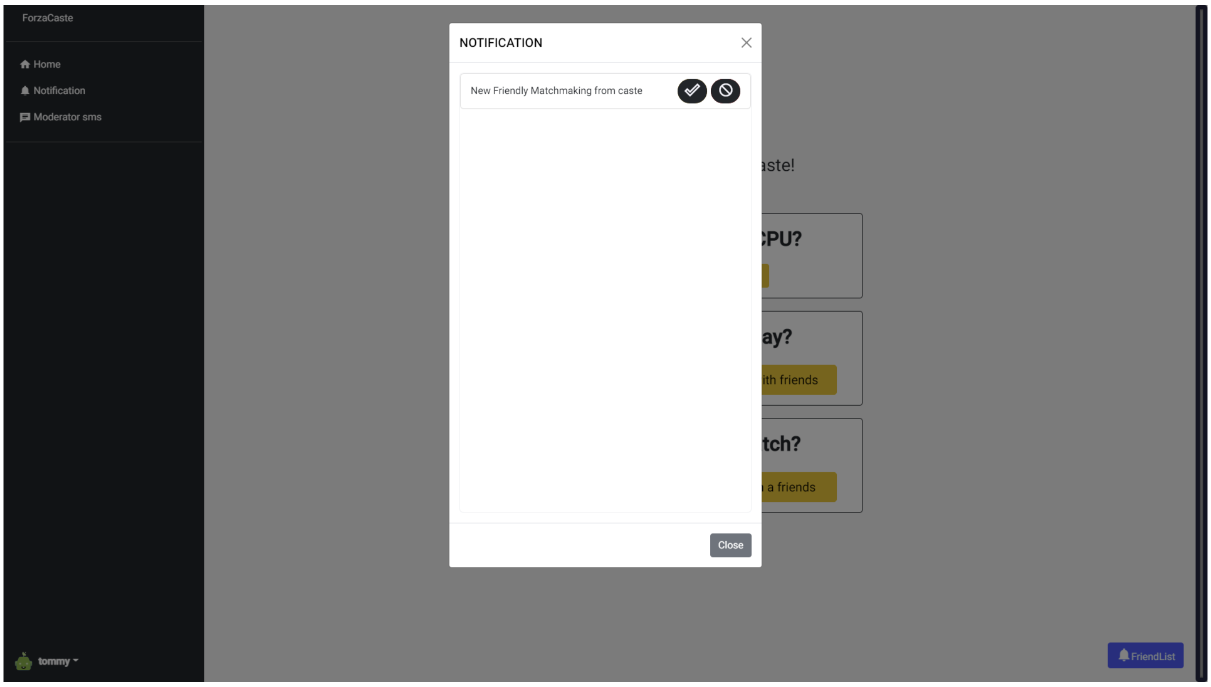
Play with a friend

By clicking the 'Play with friends' button you can select the friend to whom you wish to send the request to play a game.





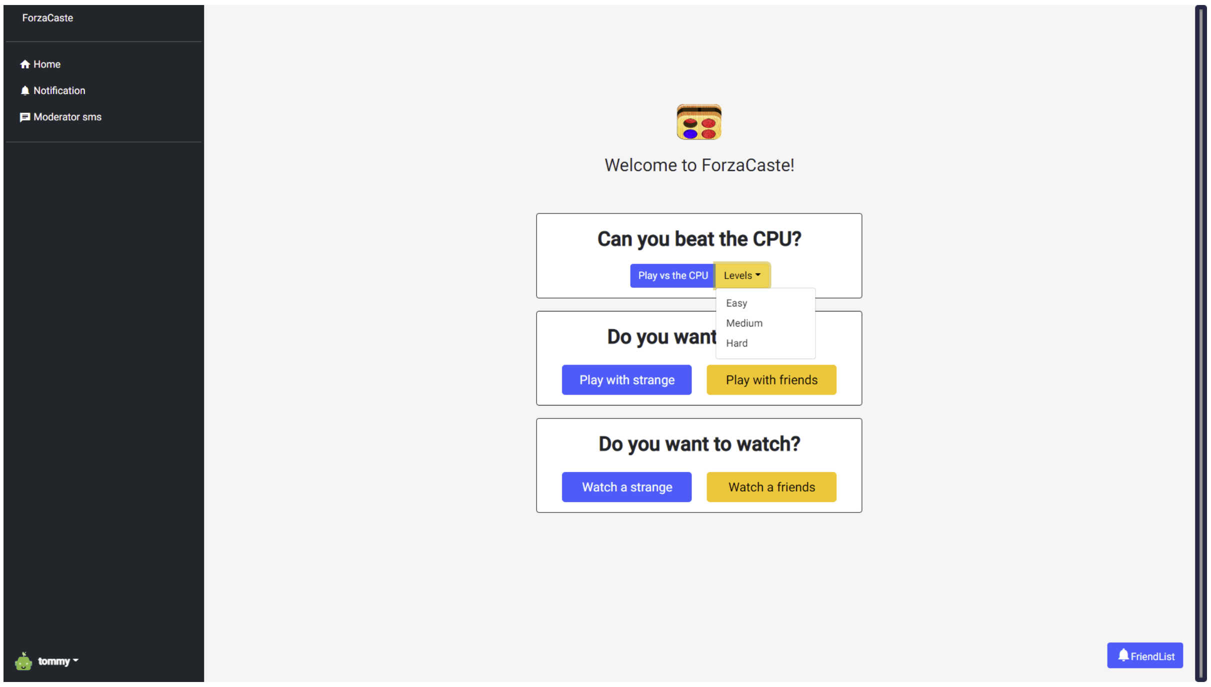
After selecting a friend to play with, the user is put on hold.



The user receives a notification of a game request from a friend in the Notification section in the sidebar. Then by accessing the Notification section all received notifications are shown and here he can accept or reject the request.

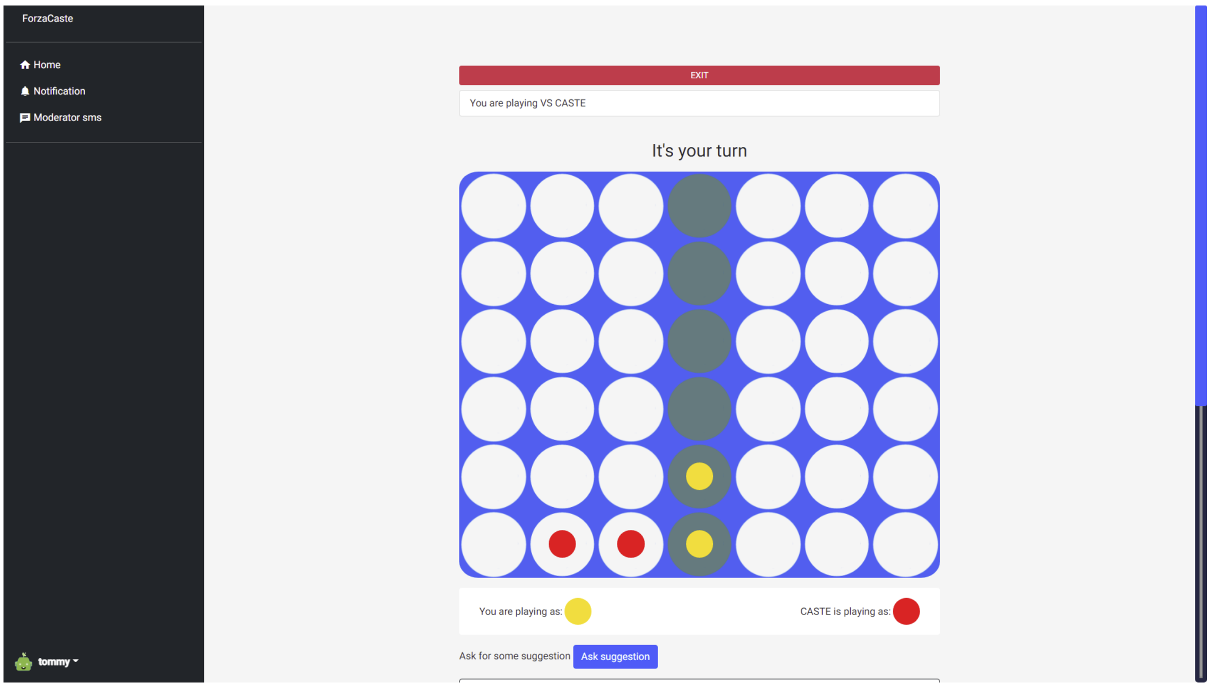
Play against CPU

By selecting the difficulty level, next to the ‘Play vs the CPU’ button you can start a game against the AI.

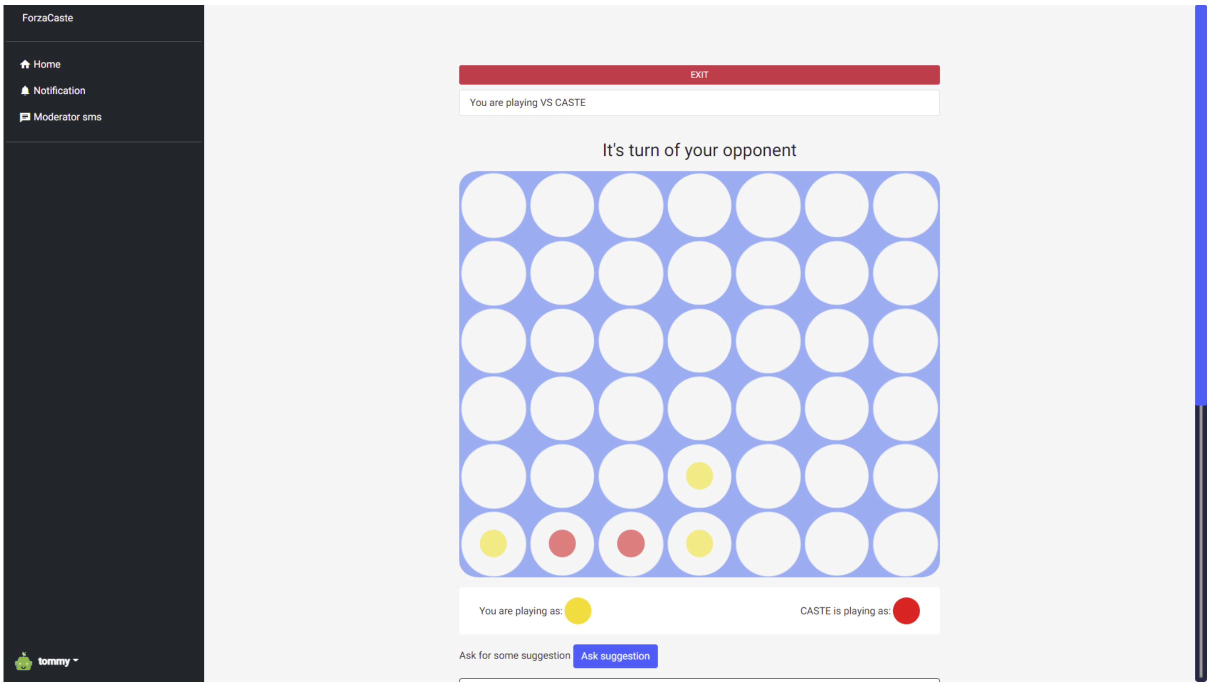


Play the move

Once a causal player has been found to play against, the friend has accepted the request to play, or the game against the AI has started, the game interface is accessed, where the user, respecting his and the opponent's turn, can play the moves.



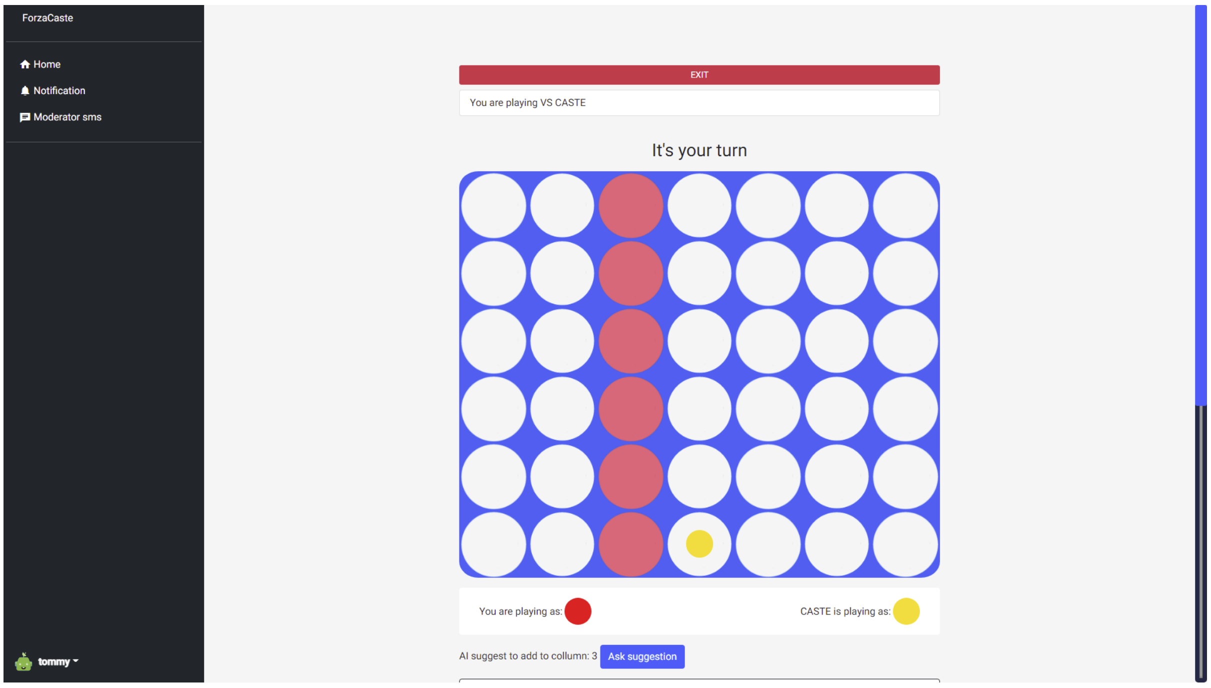
It's the user's turn, so he can play the move.



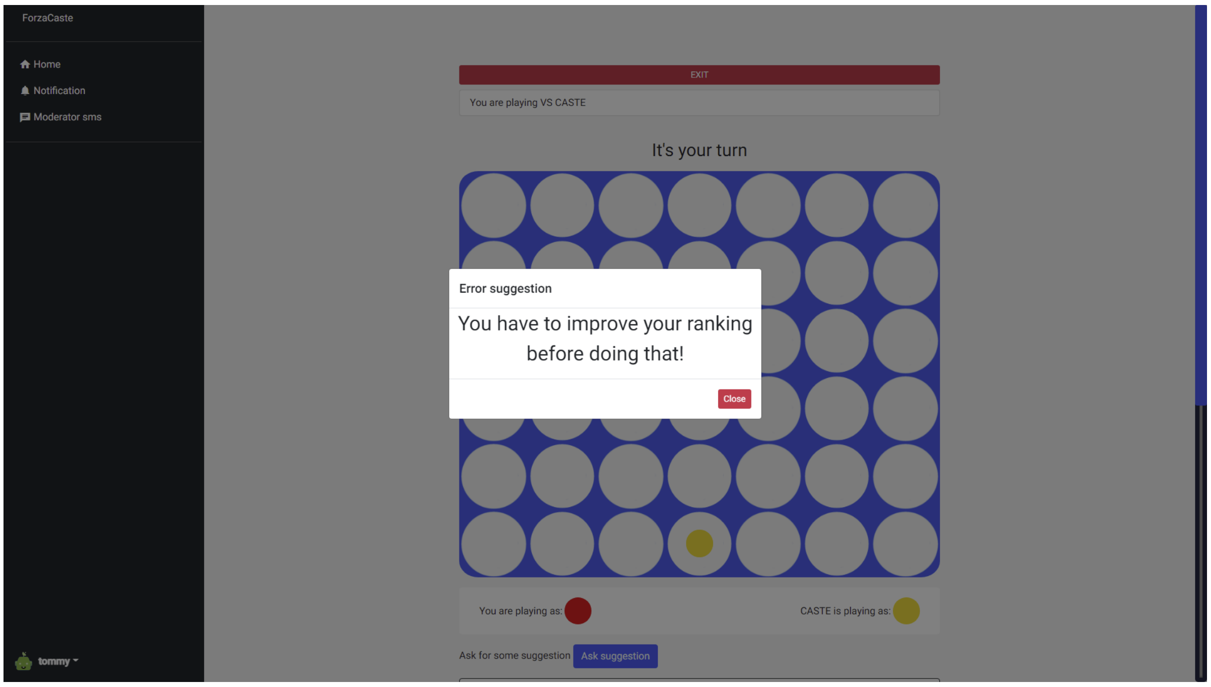
It is the opponent's turn, so the user must wait for him to play the move.

Ask for a move suggestion

When a user has a high enough ranking (greater than 100) he can ask the AI to suggest a move to him.



The AI suggested the move to the user.



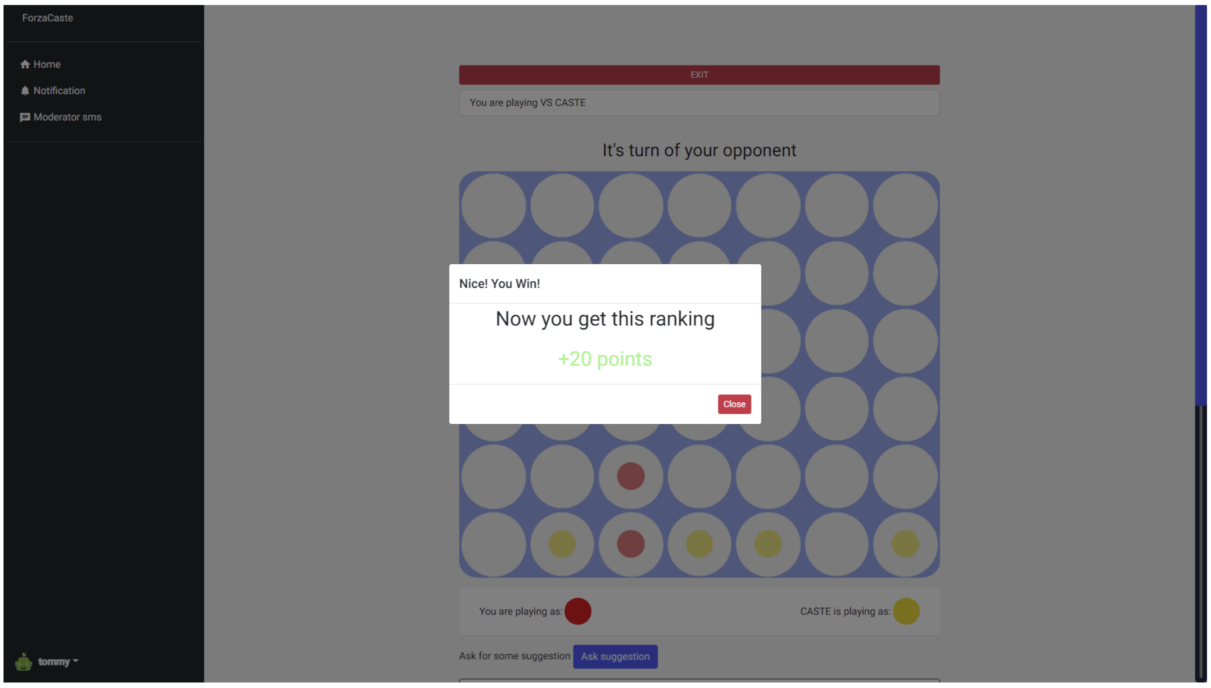
Error suggestion

You have to improve your ranking before doing that!

In this case the user does not have a high enough ranking, so the move suggestion is not available.

Match finished

When the game ends, a message is displayed to inform the players.

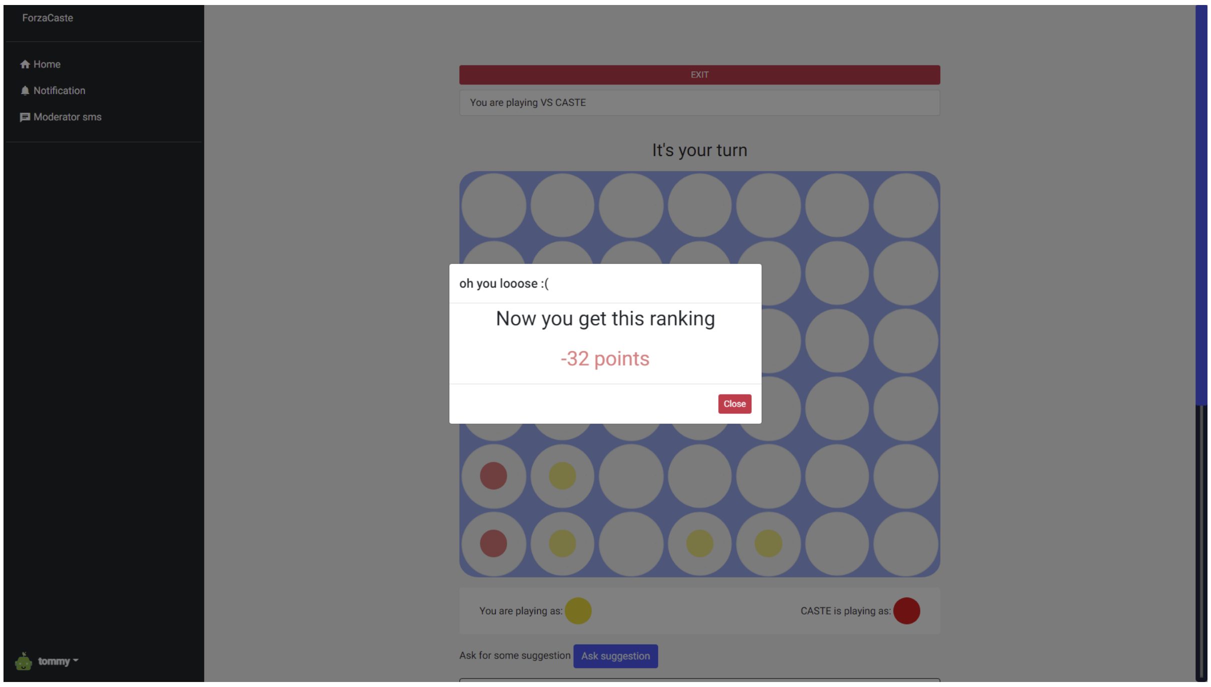


Nice! You Win!

Now you get this ranking

+20 points

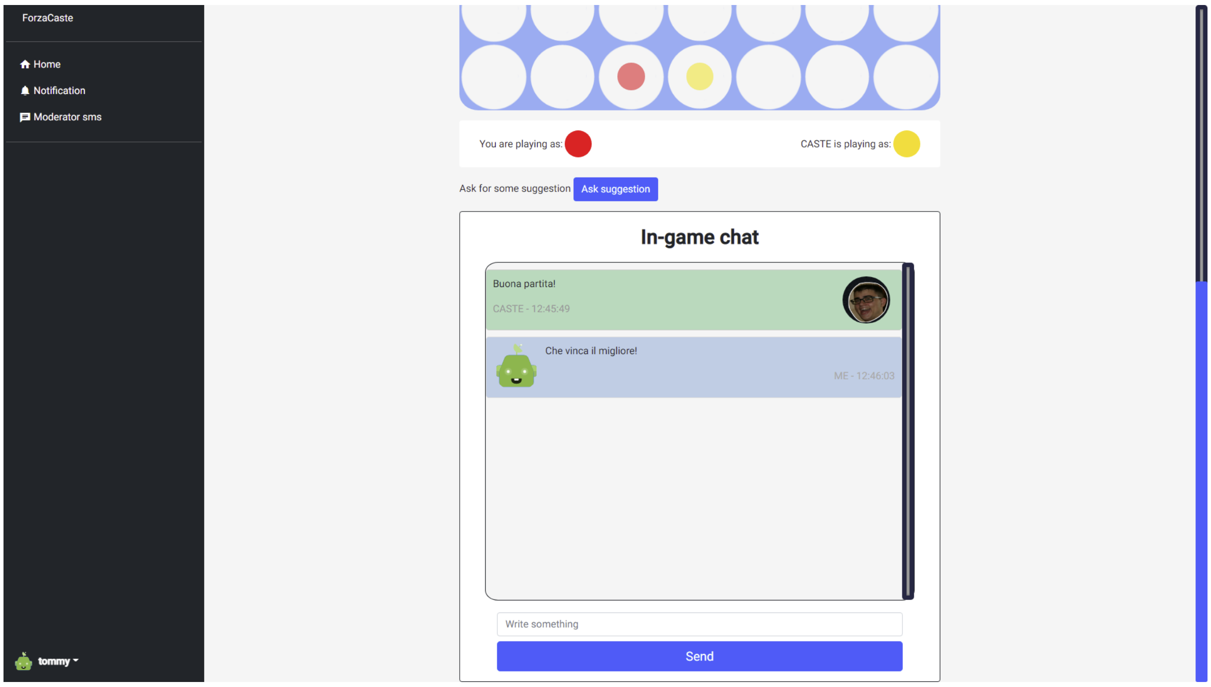
In this case the user has won.



In this case the user has lost.

Game chat

Here is shown the game chat where players and observers can read, write and send messages.



Buona partita!
CASTE - 12:45:49

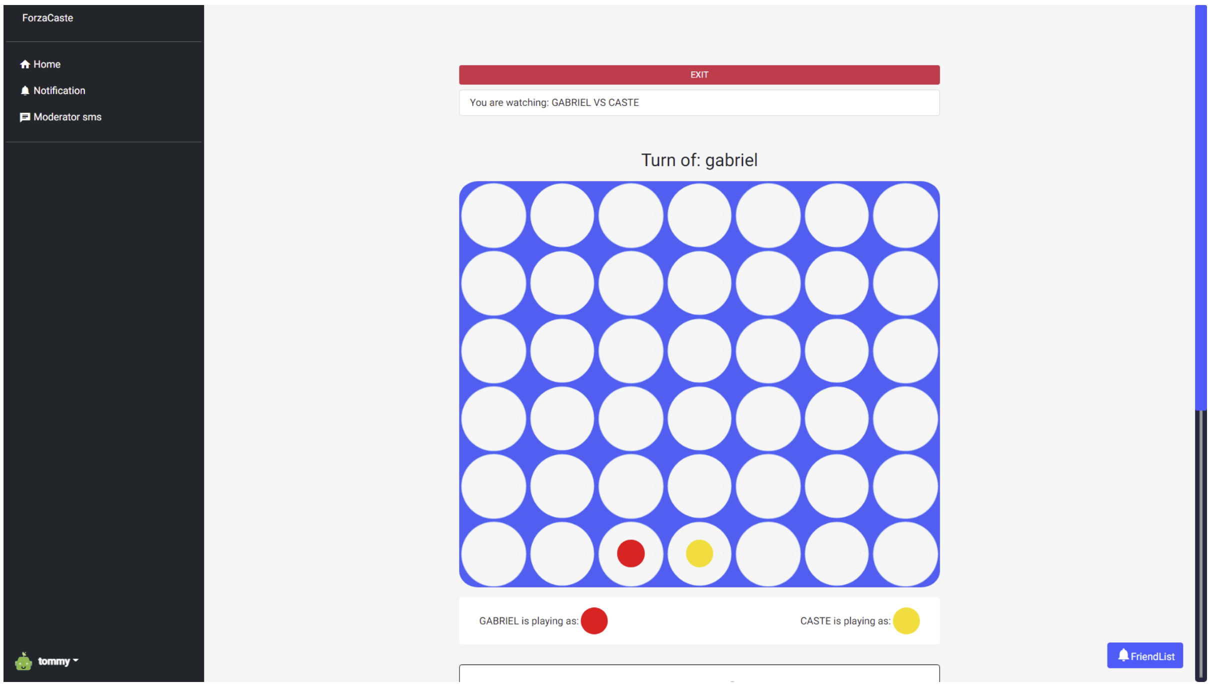
Che vinca il migliore!
ME - 12:46:03

Observer

The following shows the user interfaces and interactions if the user wants to watch a friend's game.

Watching game

The observer watches the game between the two users and in particular he can see the field updating with the moves made by the players and is told whose turn it is.



You are watching: GABRIEL VS CASTE

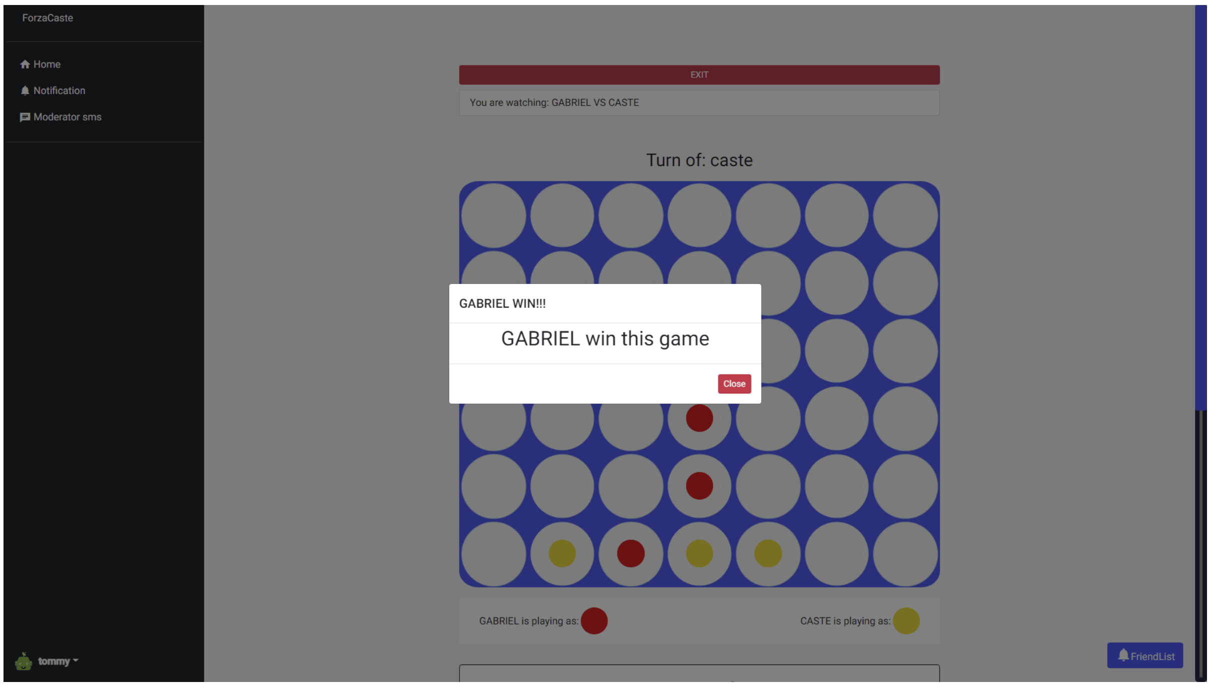
Turn of: gabriel

GABRIEL is playing as: 🔴

CASTE is playing as: 🟡

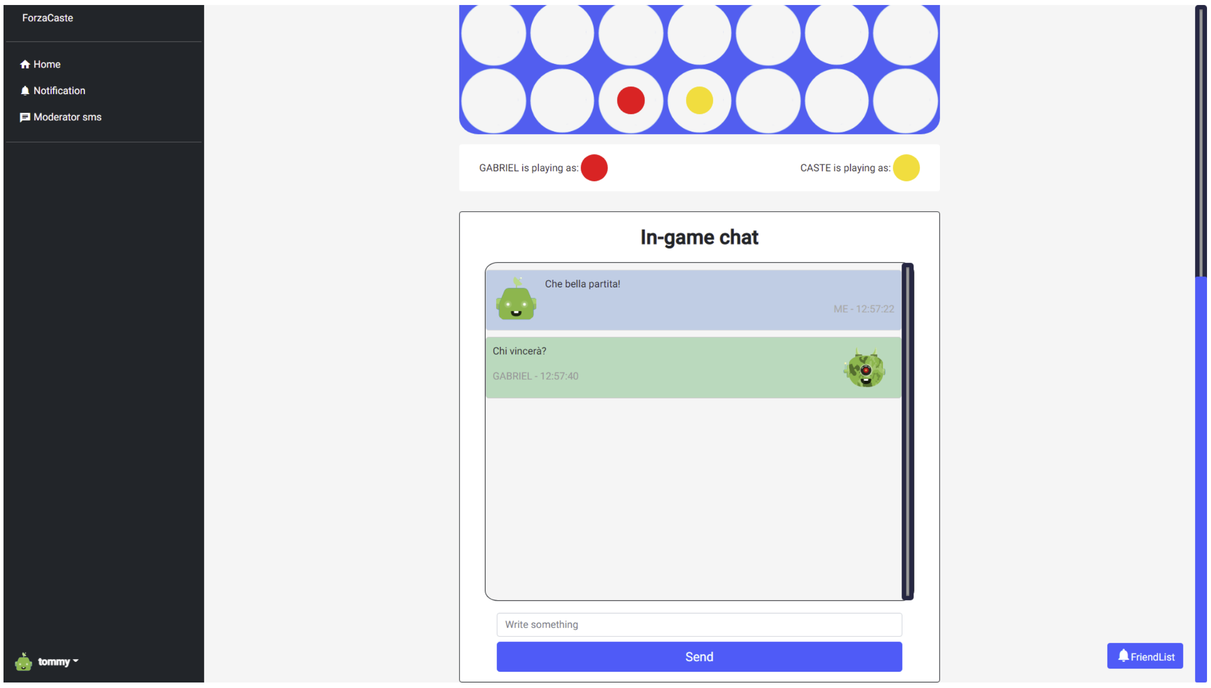
Message game ended

When the game ends a message is displayed to the observer informing them of the winner.



Observer game chat

Observers watching the game can send messages in the game chat. However, messages sent by observers are only viewed by observers, while those sent by players are received by everyone.



Che bella partita!
ME - 12:57:22

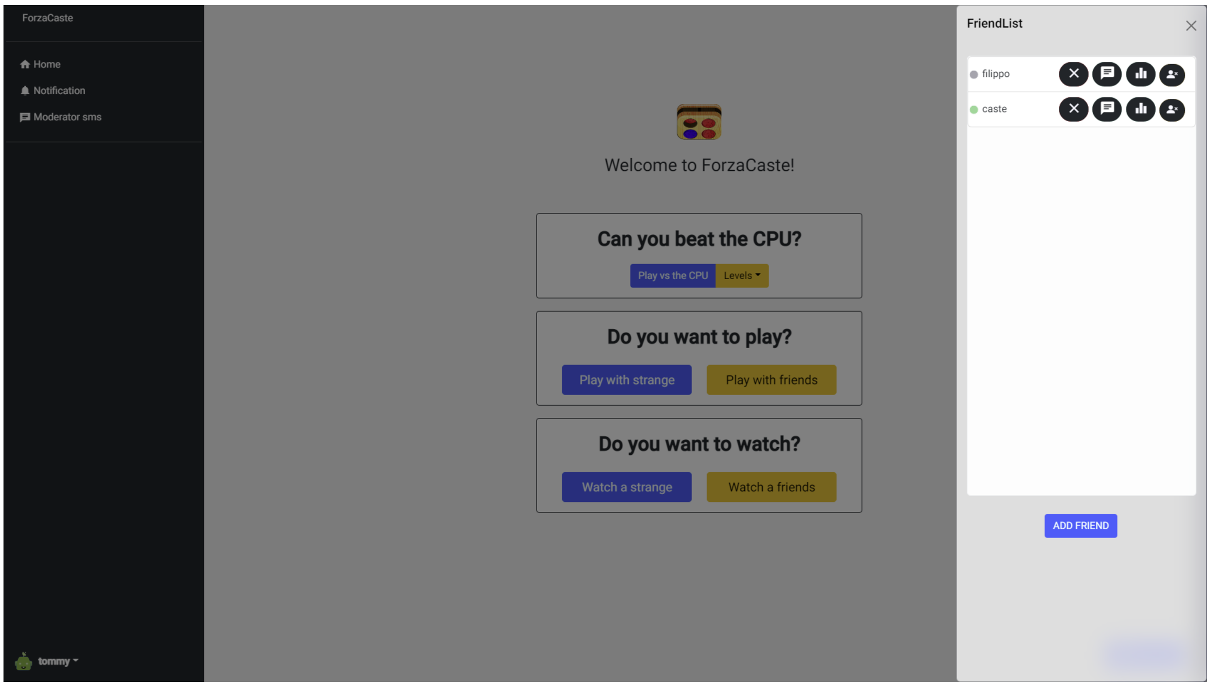
Chi vincerà?
GABRIEL - 12:57:40

Friend

This section shows all the views and workflows that take place in the interaction between friends.

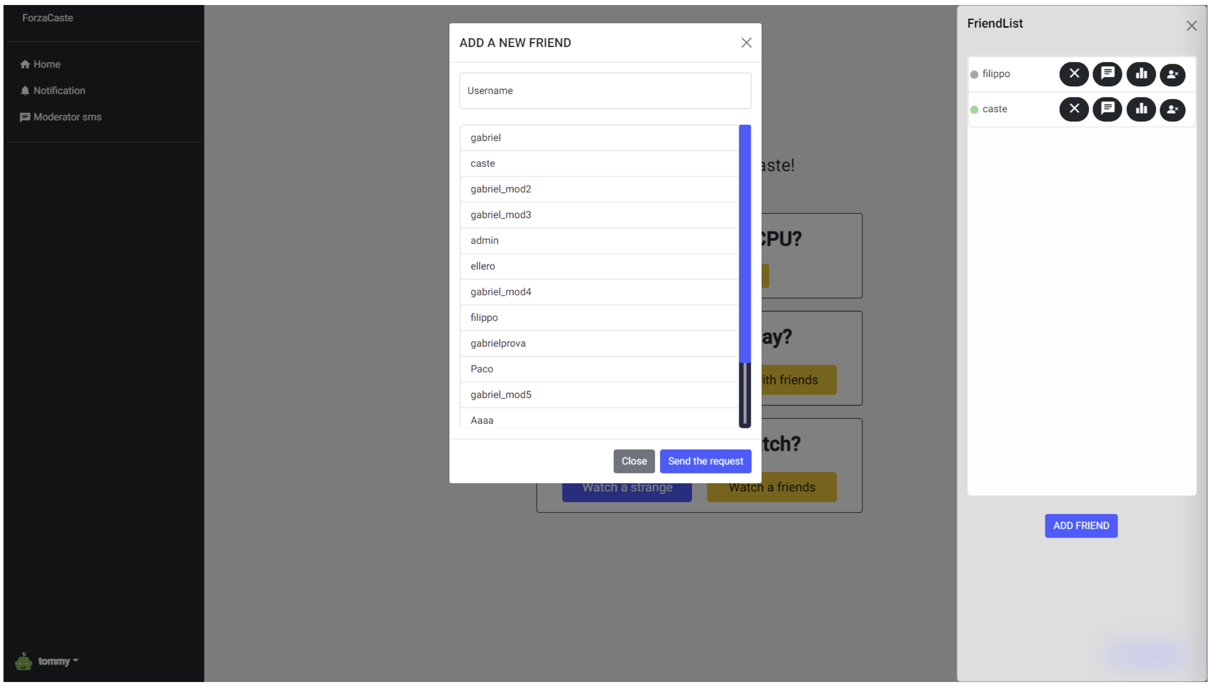
Friendlist

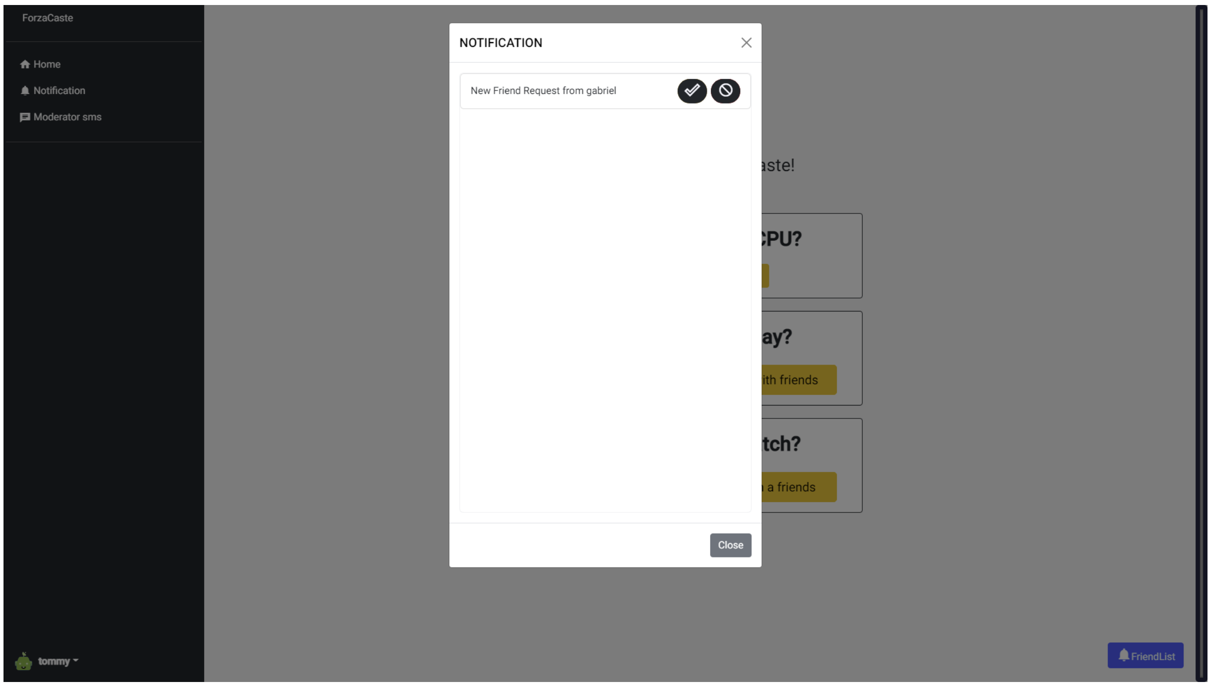
This is the list of friends with all the operations that can be performed. It is also possible to see which friends are online, in which case the dot next to the username is green.



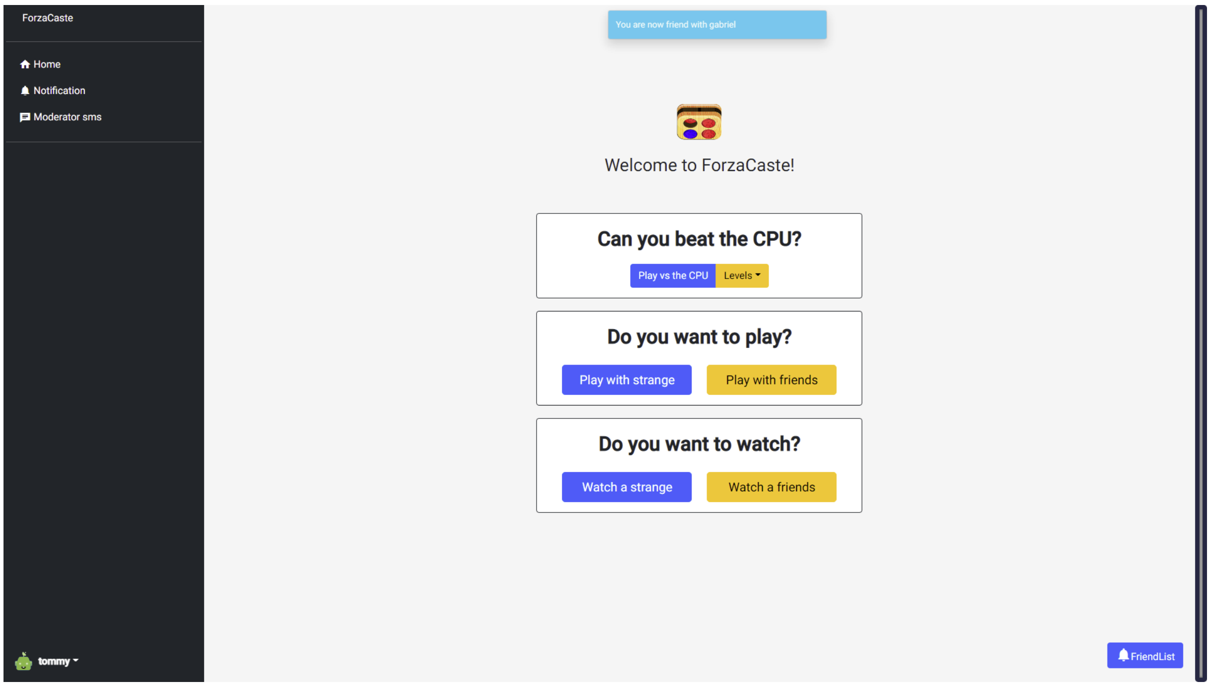
Add friend

By clicking on the 'Add friend' button you can then select one of the users present to send them a friend request.





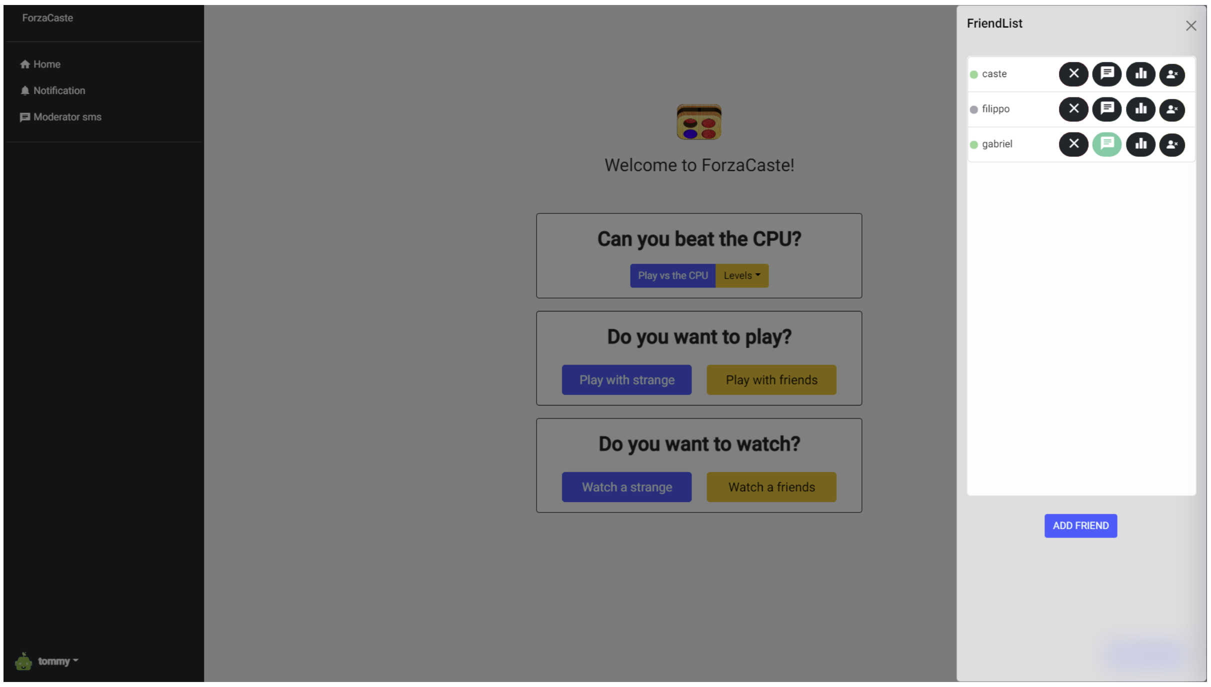
In the sidebar, under the 'Notification' section, you can view the friend requests received, and you can accept or reject them.

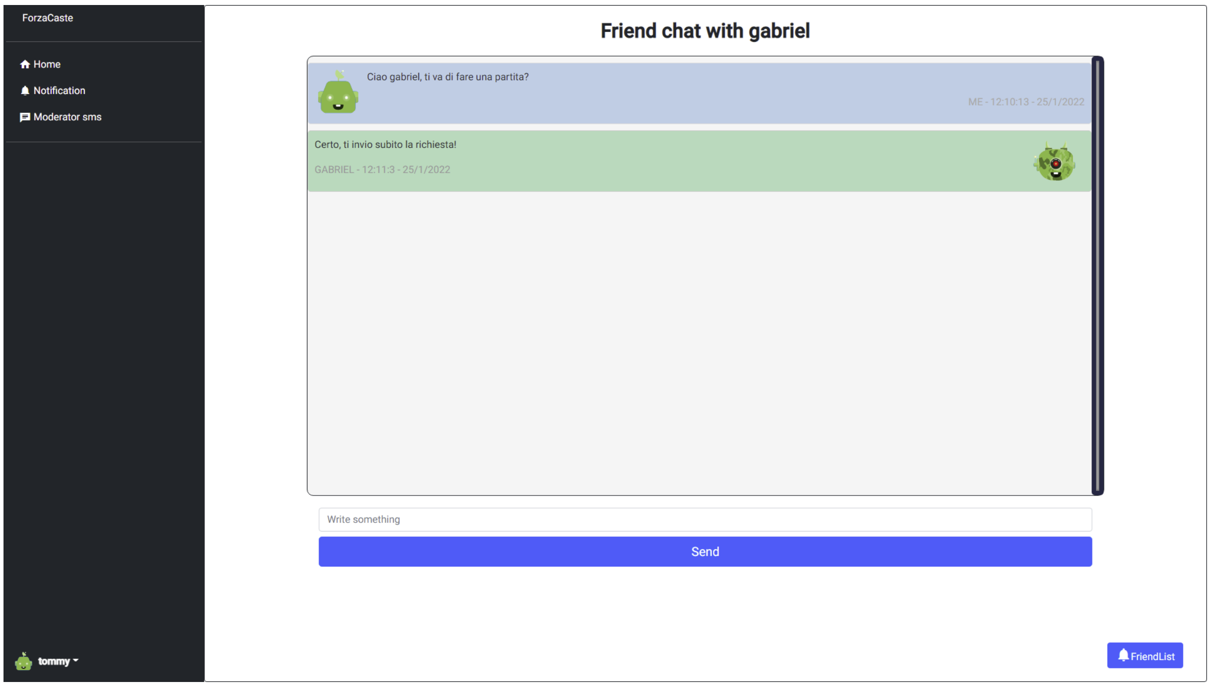


When a friend request is accepted or rejected, a toast informs the user.

Chat

The user who wants to chat with a friend can access the chat via the chat button. This will open the chat with the friend where you can read the messages received and write and send new messages.

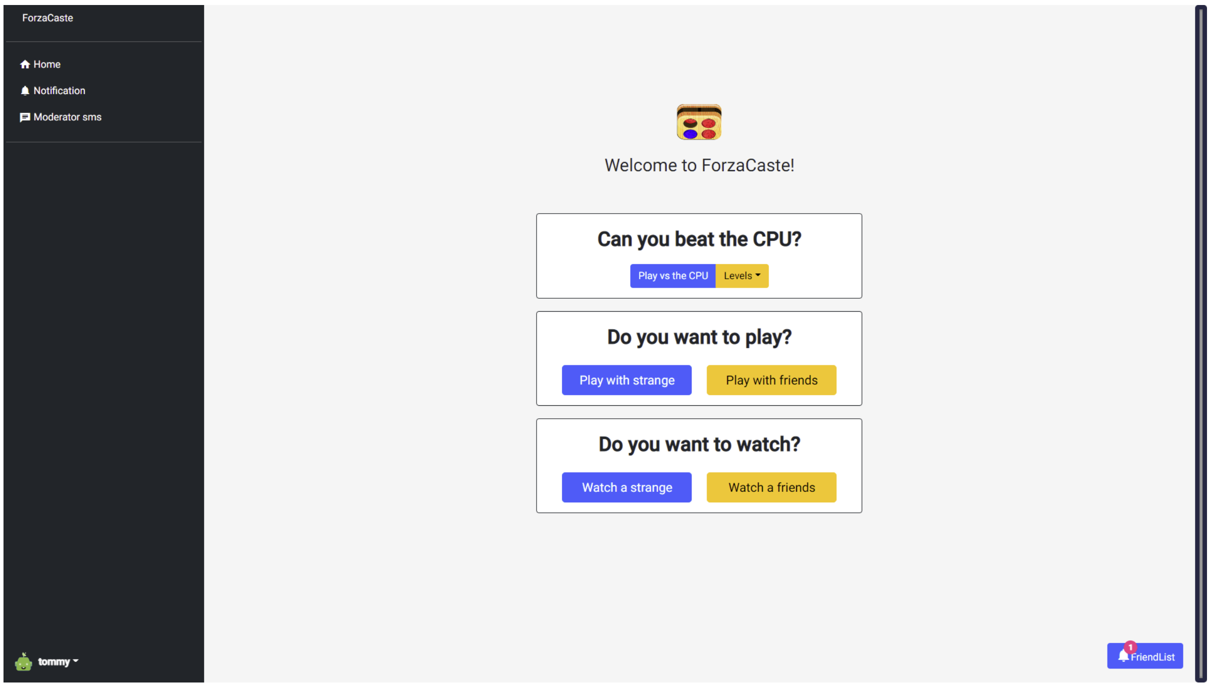


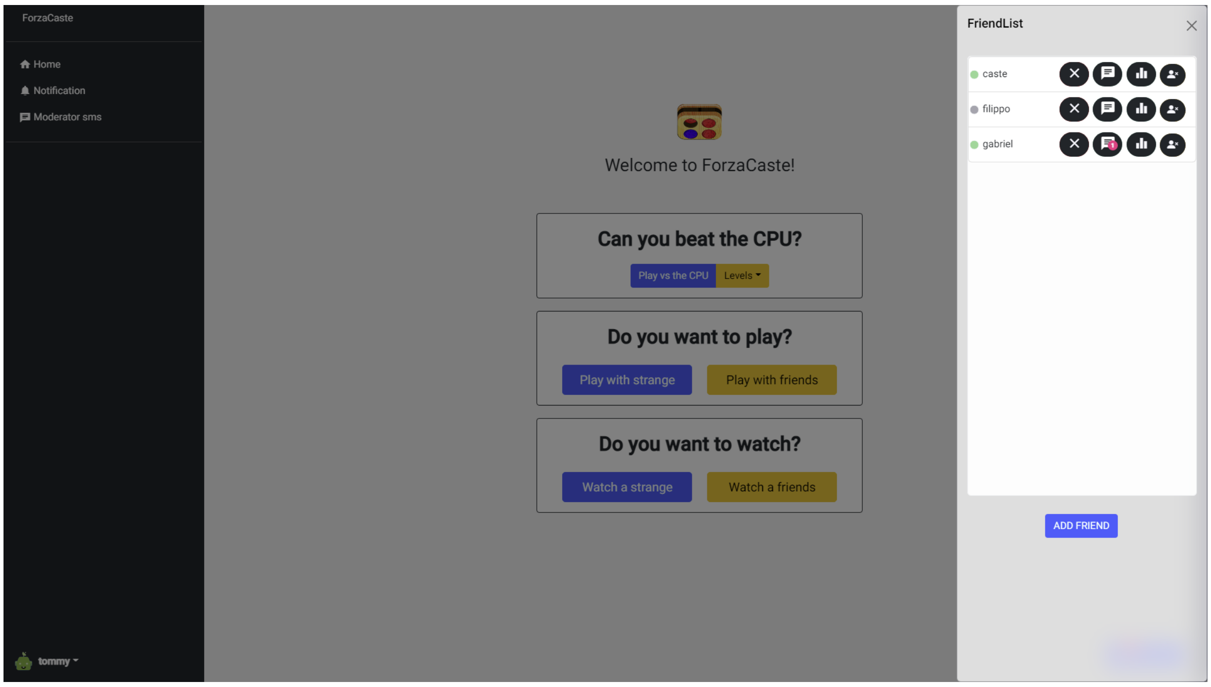


Ciao gabriel, ti va di fare una partita?
ME - 12:10:13 - 25/1/2022
Certo, ti invio subito la richiesta!
GABRIEL - 12:11:3 - 25/1/2022

Message notification

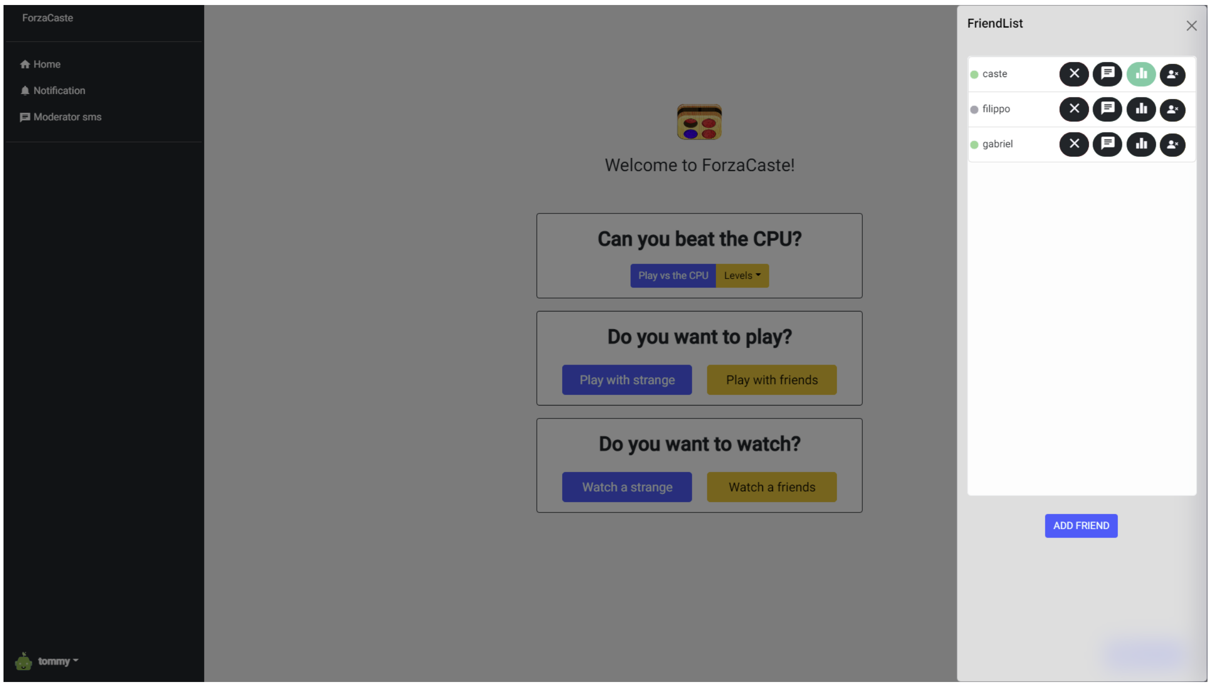
When a message is received from a friend, a badge appears in the 'FriendList' button. In addition, by accessing your friend list you can see from which user you received the message.

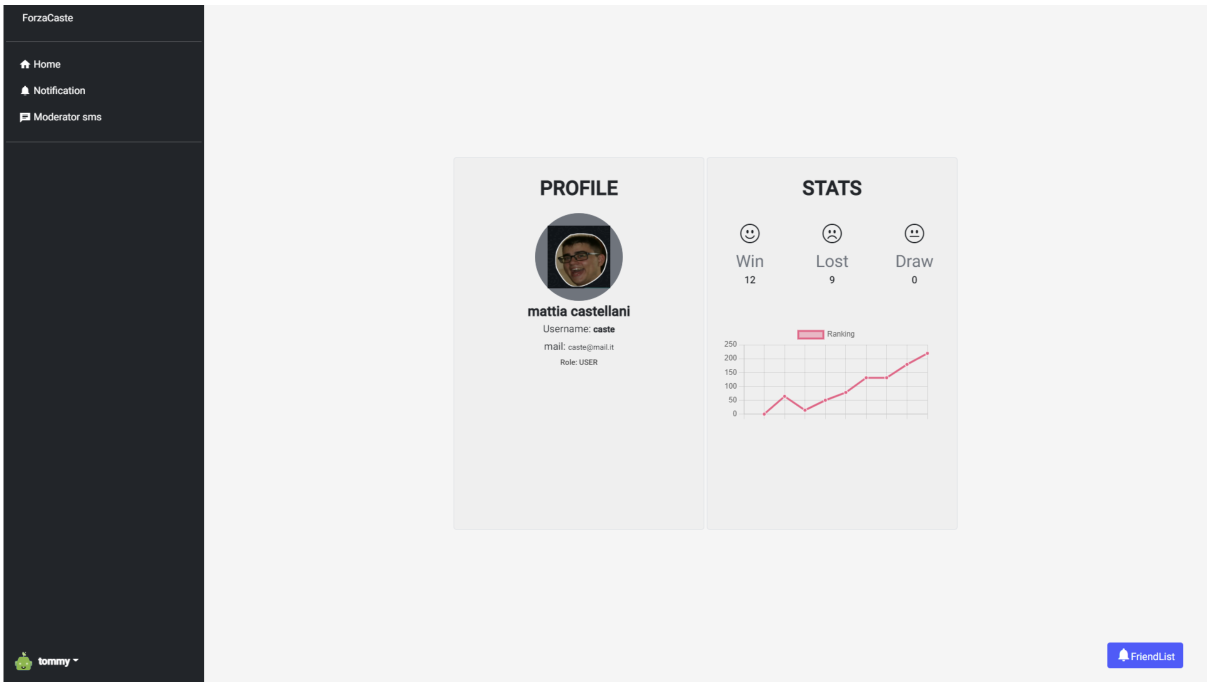




Friend profile

From the friends list you can access your friend's profile to view their information and game statistics.

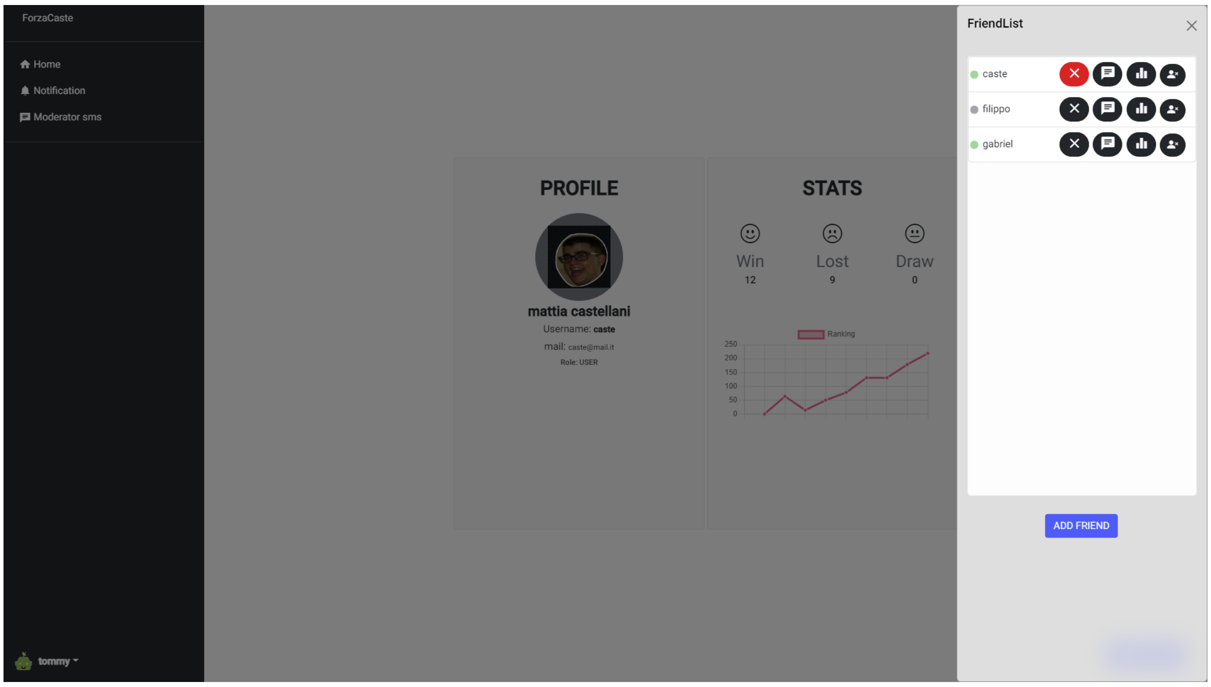




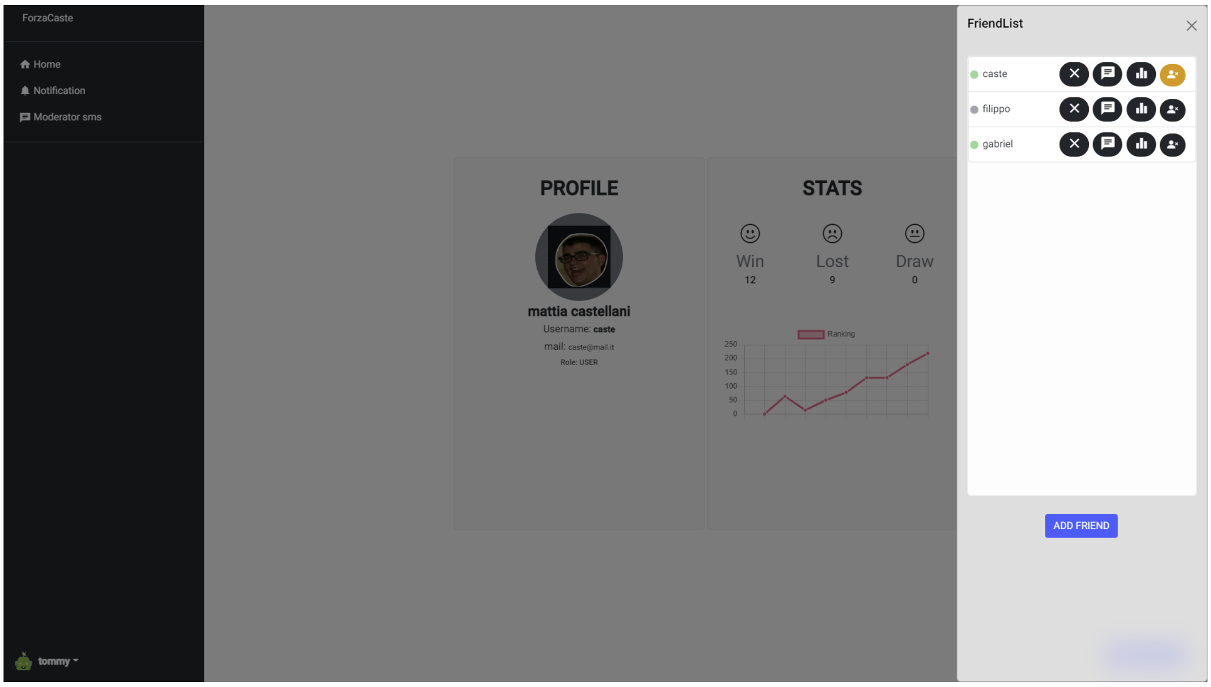
This is the friend’s profile.

Other operations

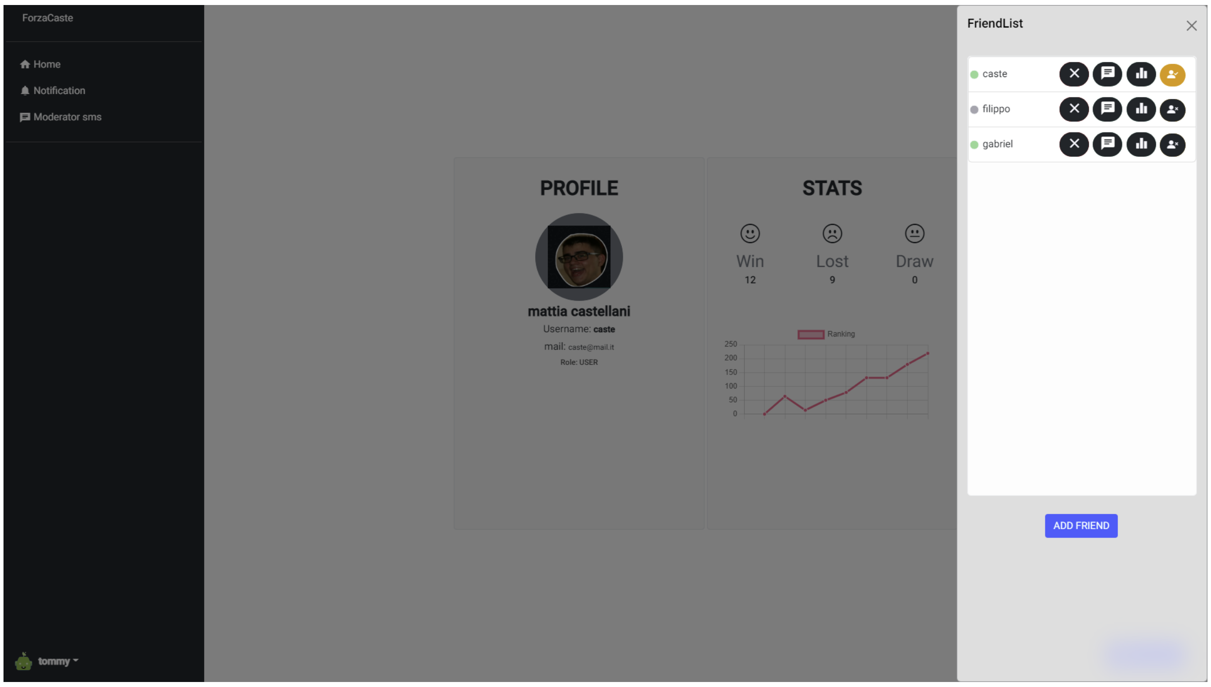
You can also delete a friend or block or unblock them.



When deleting a friend, a toast appears to the other user informing them that they have been deleted from a user's friend list.



Blocking a friend.



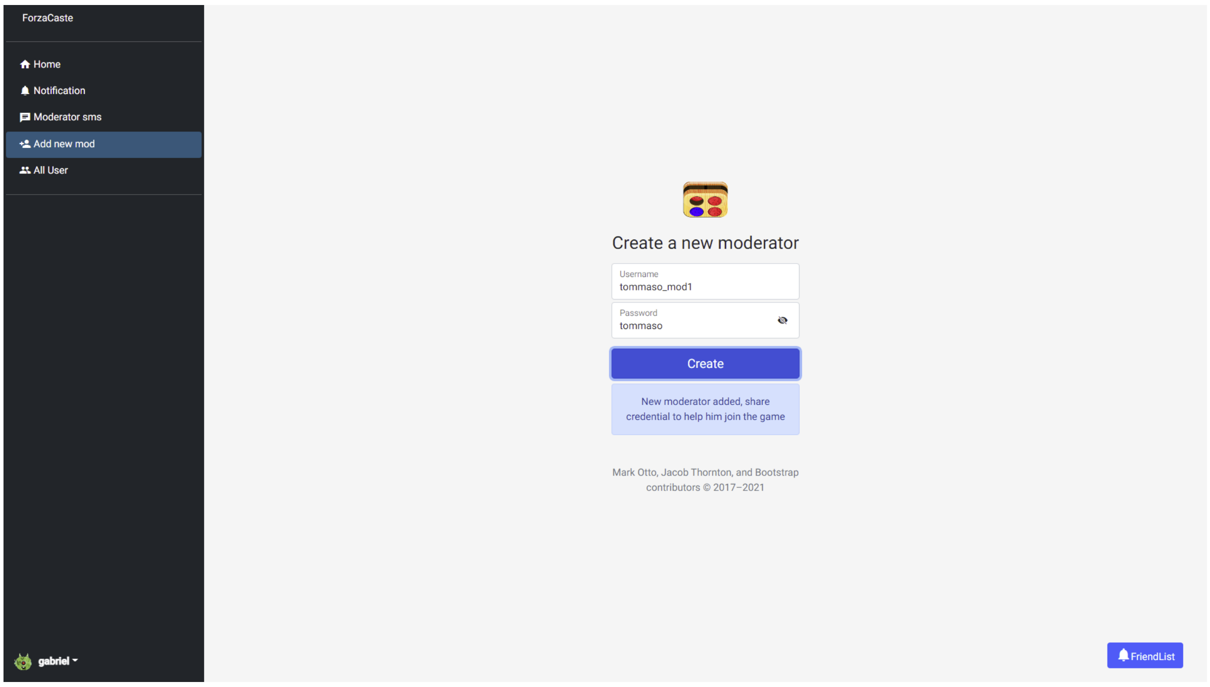
Unblocking a friend.

Moderator

Views and interactions involving moderators are shown below. In particular, more functionalities are provided than for a normal user.

Add new moderator

In the sidebar there is a button to add a new moderator. When a new moderator is created, temporary credentials are chosen, which the moderator will have to change the first time he/she logs in.



Create a new moderator

Username
tommaso_mod1

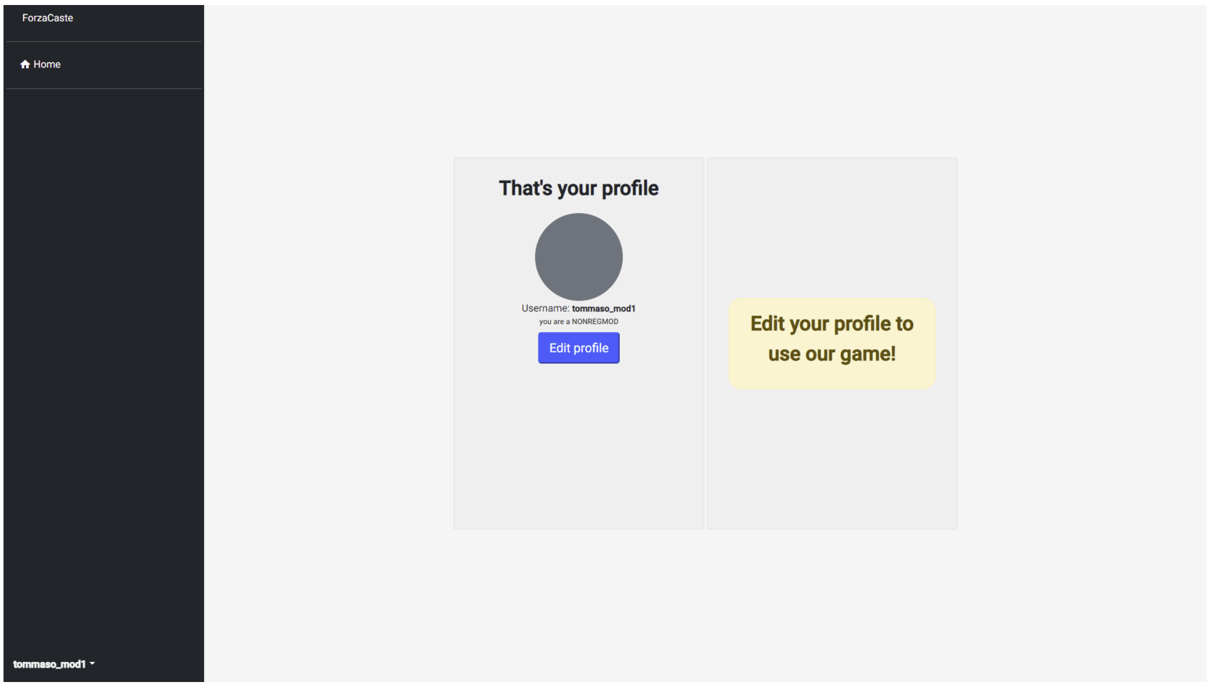
Password
tommaso

New moderator added, share credential to help him join the game

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

New moderator first access

As said previously, when logging in for the first time, the moderator must add his details and set his credentials.

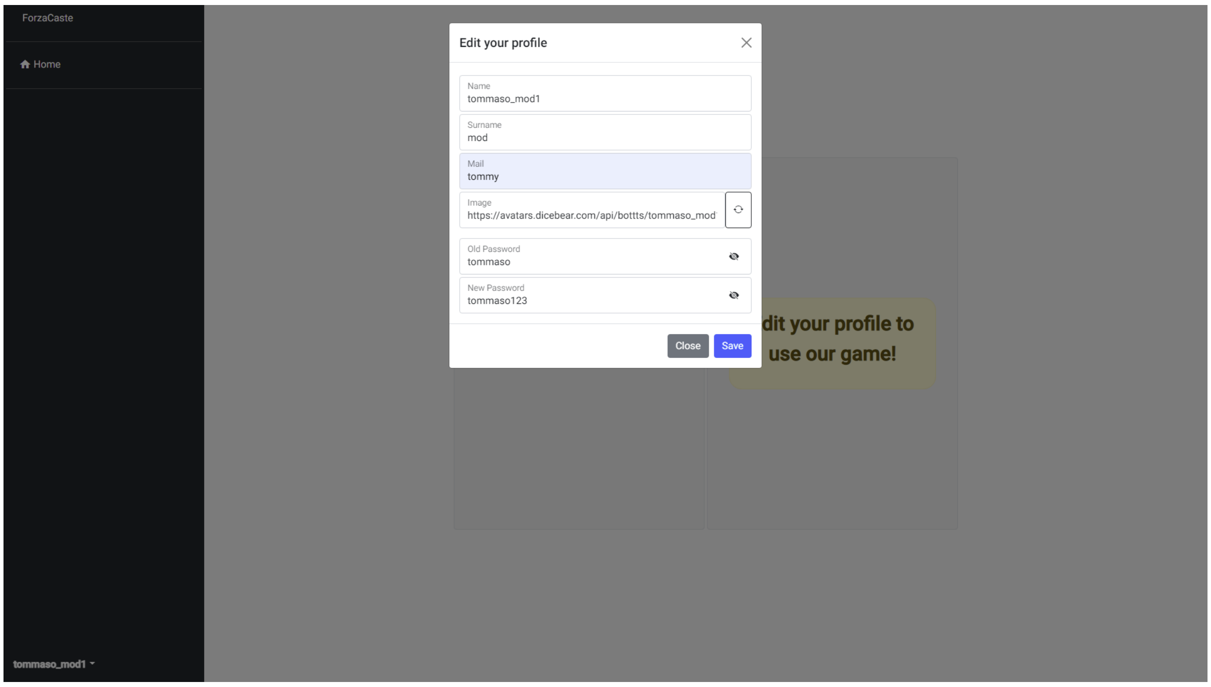


That's your profile

Username: tommaso_mod1

you are a NONREGMOD

Edit your profile to use our game!



Edit your profile

Name: tommaso_mod1

Surname: mod

Mail: tommy

Image: https://avatars.dicebear.com/api/bottts/tommaso_mod

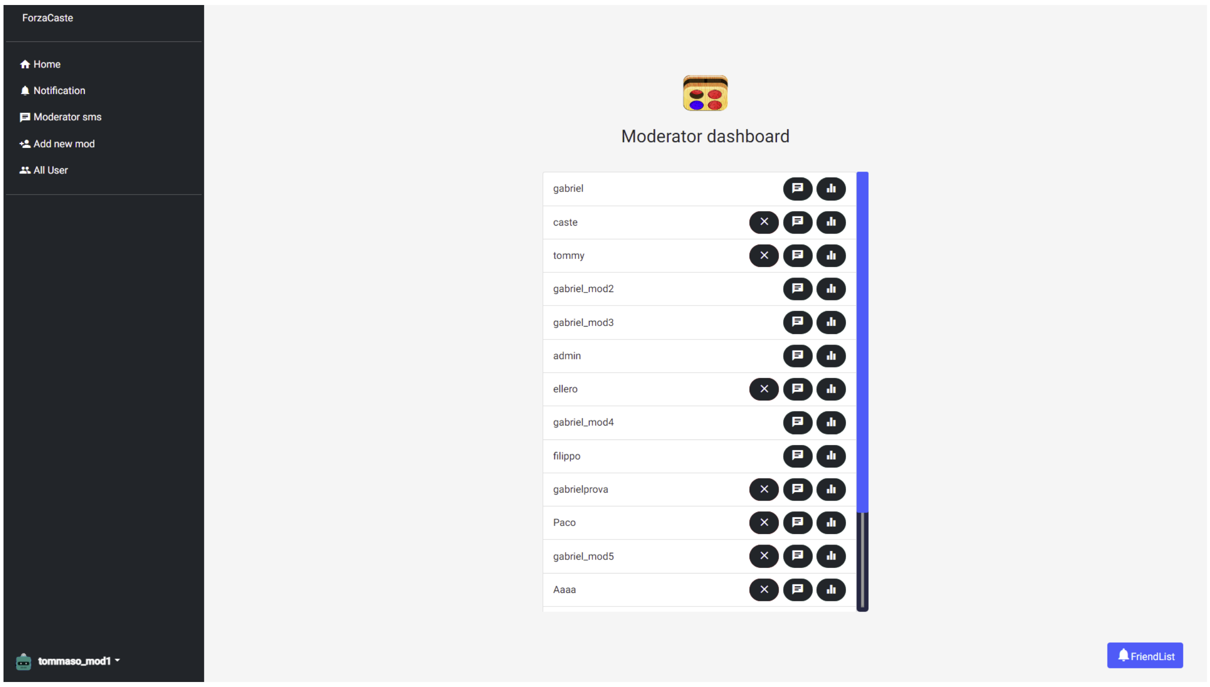
Old Password: tommaso

New Password: tommaso123

Close Save

View all user

Another additional functionality of the moderators, accessible from the sidebar, is the list of all registered users of the platform. In this section the moderator can delete a normal user, chat or view a user's profile and game statistics using the buttons for each user.



Moderator dashboard

gabriel
caste
tommy
gabriel_mod2
gabriel_mod3
admin
ellero
gabriel_mod4
filippo
gabrielprova
Paco
gabriel_mod5
Aaaa

Additional features

Our group decided to implement the following additional functionalities:

- Possibility of playing against the CPU, i.e. implementation of an AI capable of playing against a user
- Requesting an AI suggestion on the move to be performed
- As for matches against random users, matchmaking using an algorithm that searches for the opponent based on statistics and ranking
- Allow users to block friends
- Deploy of the web application on the web and possibility to download the PWA

Game against the CPU

The feature was implemented using the Minimax algorithm. In particular, this algorithm is able to predict the possible moves that the user might make and then chooses the best move to play. The difficulty level chosen by the user corresponds to the intelligence level of the AI, where intelligence level means the number of user-executable moves that the AI is able to predict. The number of moves it can predict corresponding to the chosen level of difficulty are:

- Easy → 2
- Medium → 5
- Difficult → 7

Move suggestion request

This feature was developed by exploiting the AI algorithm used for matches against the CPU. So this feature consists of asking the AI what move it would make and returning the result to the user. So it is up to the user to decide whether to follow the advice or execute a move themselves.

Matchmaking

We decided to develop matchmaking for games against random users not so that the user is chosen completely at random, but so that a player of a certain level competes against a user of the same level. We have therefore introduced a system that calculates the level of each user. The level of each user is calculated using the win rate (ratio of games won to total games played) and the number of moves the user performs on average in each game (we have considered that a user who performs fewer moves to win is probably good at the game). This system is not only used for matchmaking but also to calculate the points to be awarded or deducted from each user in case of victory or defeat. For example, the higher a player's level is, the fewer points he or she will lose in the event of a defeat.

Block and unblock friend

We have thought about giving the user the possibility to block, and in case later unblock, a friend. Blocking a friend means that the user will no longer receive requests from the friend, such as game requests, and means that the user will no longer receive messages from the friend.

Deploy

Finally, we decided to deploy the application on the web, making it accessible to everyone via the following link https://forzacaste.live. The web frontend has been hosted on GitHub, while the web backend has been hosted on Heroku.

It is also possible to download the ForzaCaste PWA.

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-connectfour-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