ForzaCaste

Filippo Vladimir Scapin

matricola: 879809

Introduction

ForzaCaste is a Single Page Application (SPA) implemented in Angular on the frontend and a REST-style webserver written in typescript on the backend.
The application has several functionalities:

1. Play against a stranger or against a friend by inviting them to participate in a game.
2. Visualize the games in progress of strangers or friends, interacting with all the people connected to that game with the game chat
3. Play against a CPU with 3 difficulty levels
4. Social aspect: Messaging with friends, blocking them, viewing their statistics

Architecture

The frontend is a Single Page Application (SPA) implemented with Angular framework.

The backend runs on a Node.js 12.22 webserver written in typescript, Express is used as a framework for writing APIs, which respect the REST standard.

The Socket.io library was used for real-time communication between client and backend.

For the database we used MongoDB, a non-relational, document-oriented DBMS, to interface with it we used the Mongoose library in the server, allowing us to perform the various queries.

Development

The application allows you to play the game Connect 4.

It was created by dividing tasks among the various members of the group who published their additions/changes via a repository on GitHub.
The frontend and backend were created on Docker containers that were intended to help us launch our application quickly and install all the components needed to start it.
Our Docker container contained inside: an image of Node 12.22, the javascript package manager npm, a TypeScript compiler (programming language used), the Angular/cli framework that allowed us to create the frontend of the application, every change that took place in this container also took place externally, therefore also to the folders to which it was connected, so there was correspondence between the various changes.
For the testing phase of all the functions implemented in the backend we used Postman.
To save all the information of the users and the game, we used the database MondoDB Atlas for hosting; to display all our models and the correctness with which they were inserted, we used a special client: MongoDBCompass.
In the end, in order to distribute the application and not just run it locally, we decided to use Heroku to distribute it completely.

Data Model

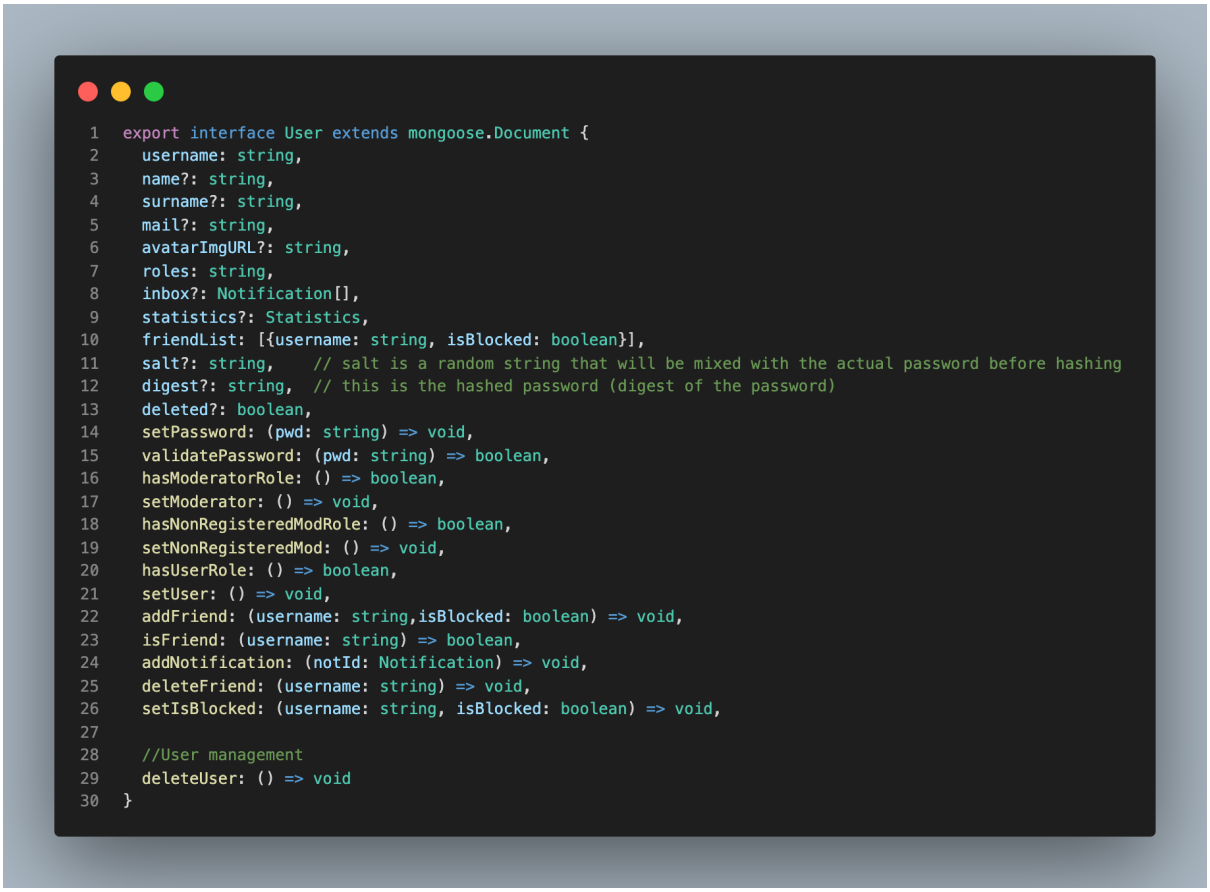
User

The template includes all the basic information that we can ask a user for in order to recognise them within our application, but some points are more important than others:

-digest: this is the password encrypted using the crypto library and the createHmac function (also used to verify a password)

-salt: random string of 16 bytes created with the crypto library, important because it makes the encrypted app difficult to reverse engineer.
-statistic: contains the statistics template for storing information about the games played by the user
-inbox: List of the Notification model, which lets you know how many notifications are still available to the user
-friendList: List of username pairs and isBlocked to identify who is a friend of the user and if they are blocked
-deleted: allows you to identify a user who has been deleted from a user who is still

active, the deletion is only logical, so the user will no longer be able to access but all his information will remain in the db

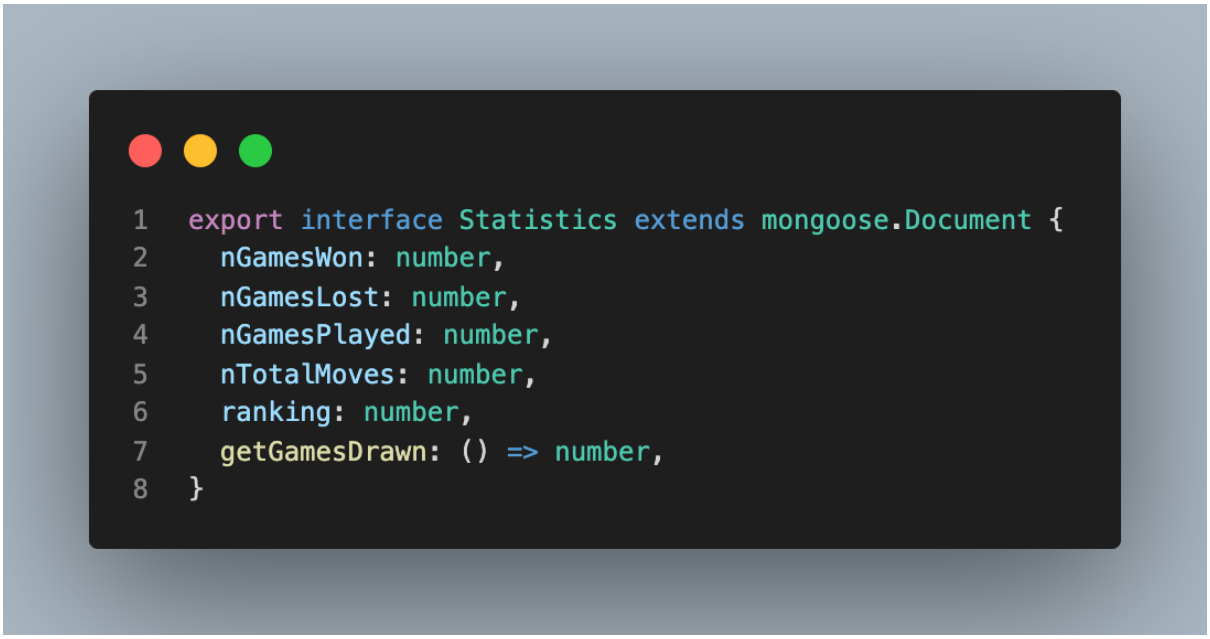


```
export interface User extends mongoose.Document {
  username: string,
  name?: string,
  surname?: string,
  mail?: string,
  avatarImgURL?: string,
  roles: string,
  inbox?: Notification[],
  statistics?: Statistics,
  friendList: [{username: string, isBlocked: boolean}],
  salt?: string,    //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

Statistics(used inside User model)

The purpose of the model is to summarise the user's history, thus making it possible to know the user's current ranking, number of total moves, number of games played, games lost, and games won.



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

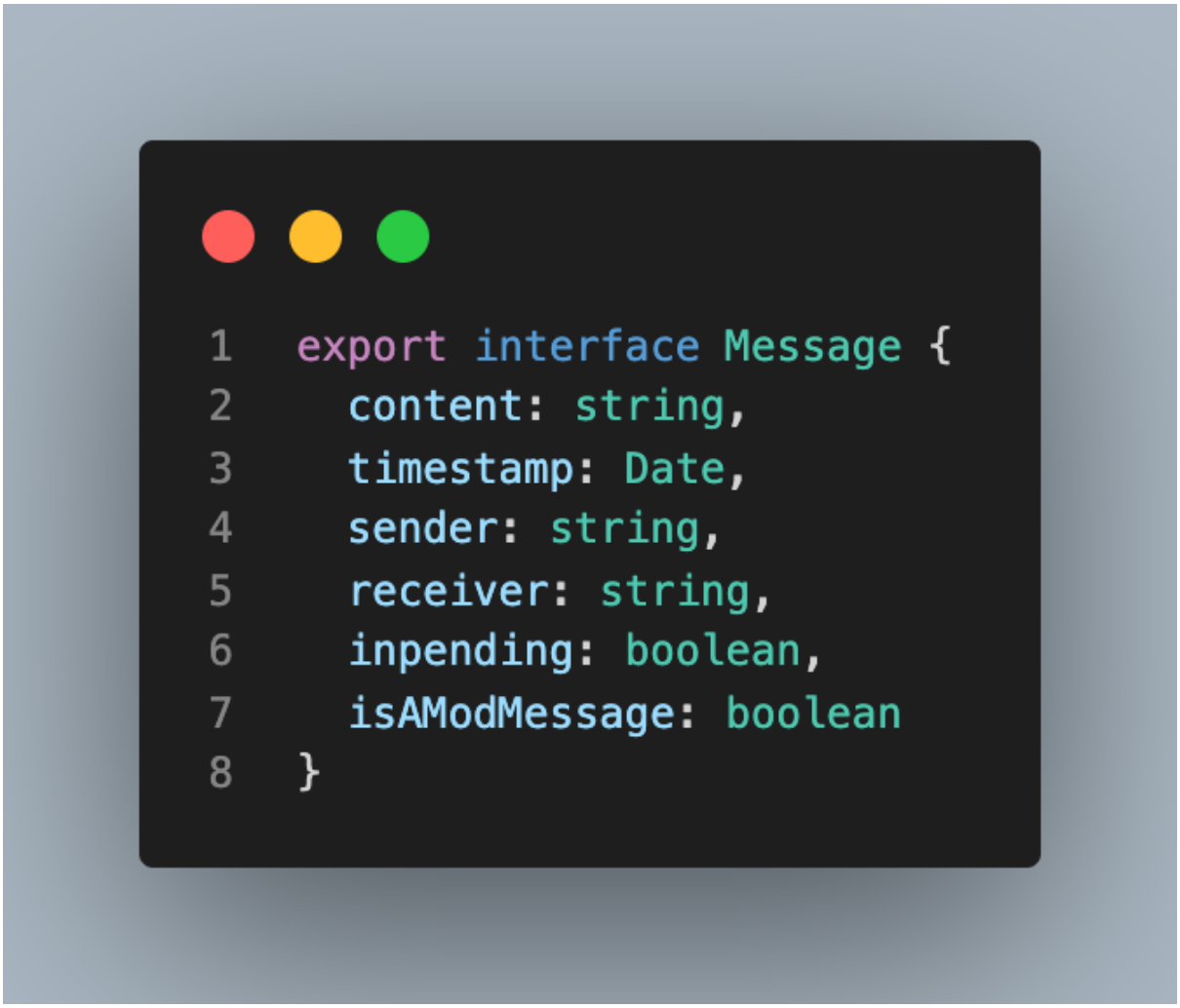
Notification

The template aims to display the notifications that can be viewed in the application in detail:
-text: contains the content of the notification
-type: identifies the type of notification
-sender: who sent that notification
-receiver: who is the recipient of the notification (if any)
-deleted: to understand if the notification should no longer be displayed because it has been removed by the user (only logically)
-inpending: to understand if that notification has already been viewed by the user

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

Message

The template is intended to display messages, these can be used between 2 users, or during game chats or while watching a match specifically:
-content: contains the content of the message
-timestamp: the date on which that message was created/sent
-sender: identifies who sent the message
-receiver: identifies who the message is intended for
-inpending: if that message has yet to be seen by the user it will be true
-isAModMessage: identifies the messages that take place within the chat between user and moderator



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

Match

The model aims to identify a match and to remember its history, i.e. everything that happens during a normal match, in detail:
-inProgress: identifies whether the match is still in progress or not.
-player1: identifies the first player.
-player2: identifies the second player, in case the game is played with the CPU, "CPU" will be put here.
-winner: identifies the name of the winner once someone has made the final move.
-playground: identifies the playing field, allowing you to reconstruct the playing field and thus all the moves made up to that moment (if it is still in progress) or all the moves made until the end of the game.
-chat: identifies the list of messages sent during the game by players or spectators.
-nTurns: identifies the number of turns that took place during the game.

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

Endpoint

The **endpoints** have been developed according to the REST API guidelines allowing communication with the frontend.

If you want to go further into the details of the endpoints and Listerners, in the project folder the file called **endpoint_documentation.pdf** provides more information.
The document was created and generated with the Postman tools, which allow us to document the various endpoints as we like.

Endpoint	Attributes	Method	Description
/	-	GET	Returns the version and a list of available endpoints
/login	-	GET	Login an existing user, returning a JWT

Endpoint	Attributes	Method	Description
/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 user
/users/online	-	GET	Return a list of online player 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 user 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
/game/cpu	-	POST	Create a match against CPU. Furthermore a user can enter in a game as observer
/game	-	PUT	Accept a friendly game request
/game	-	DELETE	Used by a player in order to delete a started game or to delete a game request

Endpoint	Attributes	Method	Description
/gameMessage	-	POST	Send a message in the game chat
/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	?inpending=true	GET	Return all the notification of the specified user. This endpoint returns all the notification that are not read
/notification	? makeNotificationRead=true	GET	Return all the notification of the specified user. This endpoint mark all the notification as read
/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	?modMessage=true	GET	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

Endpoint	Attributes	Method	Description
/friend	-	GET	Return the friendlist of the current logged user
/friend	-	PUT	Change the attribute isBlocked of the specified user in the friendlist
/friend/:username	-	DELETE	Deletion of a friends in the friendlist of the current logged user

Listeners

Listening on	Received body	Description
gameReady	{ 'gameReady': true, 'opponentPlayer': ""}	Inform both clients that game requests have been accepted and the match will start soon
move	{ 'yourTurn': bool }	Inform the user if it's his turn (true) or if it's opponent turn (false)
gameStatus	{playerTurn: ""}	Informs the observators which player will do the first move
gameRequest	{type: "friendlyGame", player: ""}	Where 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 play the next move and the status of the playground at the moment
result	{'winner': "", 'message': "Opposite player have left the game"}	You can receive that body during a game, with the information of who win, if the opponent left the game you'll be informed
move	{ "error": true, "codeError": 3, "errorMessage": "Wrong turn" }	If the player is trying to play even if it is not his turn

Listening on	Received body	Description
move	{ "error": false, "codeError": null, "errorMessage": null }	If the player’s move is correctly added
move	{ "error": true, "codeError": 1, "errorMessage": "The column is full" } **or** { "error": true, "codeError": 2, "errorMessage": "Move not allowed, out of playground" }	If the player is trying to insert an invalid move
gameStatus		match observers, who are joining the match room identified by the string made up of player's username concatenated to the string Watchers*(i.e. usernameWatchers),* receive { player:'player username or cpu', move: index, nextTurn: 'player username or cpu' } which represents the player who inserted disk into column index and the player who has to play next turn
result	{ "winner": null }	players receive the body if the game ended up in a draw
result	{ winner: true }	the winner receive the body
result	{winner:false}	the loser receive the body
result	{ "rank": rank }	players receive the body representing the points they have lost or gained after this match
result	{ winner: 'username of the player who won'}	match observers, who are joining the match room, receive the body
gameChat	{'_id' : "", 'content' : "", 'sender' : "", 'receiver' : "", 'timestamp' : {"$date" : ""}}	all the users involved in the match, so the players and the observators, receive the body that correspond to the match document in the DB.

Listening on	Received body	Description
newNotification	{ sender: "", type: "friendRequest" }	the recipient of the request receive the body that inform the destination user of a new friend request
request	{newFriend: ""}	the creator of the request receive the body that inform that the other user has accepted the friend request. If the newFriend field is null it means that the friend request has been refused
friendDeleted	{deletedFriend: ""}	the friend of the user receive the body that inform the user that he has lost a friend
friendBlocked	{blocked: bool}	the friend that has been blocked or unblocked receive the body that inform if he has been blocked if it is true, otherwise if it is false it means he has been unblocked
message	{'_id' : "", 'content' : "", 'sender' : "", 'receiver' : "", 'timestamp' : {"$date" : ""}, inpending: '', isAModMessage: true}	the target user of the message receive the body that correspond to the message document in the DB.

Common Response

This table makes it possible to identify the various types of response received from the backend, with their code and description of what they mean.

StatusCode	Error Message
200	**OK** - The request has succeeded
400	**BAD REQUEST** - The server cannot or will not process the request due to something 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

StatusCode	Error Message
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

Authentication is based on JWT which is provided by the server when you login, based on Basic Authentication.

We used Passport, it is a middleware for Node.js, that makes it easy to implement authentication and authorization, we used it during the Basic Authentication phase

At the beginning, after sending the login request to the endpoint, the endpoint will verify the credentials entered with those that are saved in the database, if there is a match a token will be created.

This Token will contain the information of the user for its fast identification, for the creation of the Token was used the library **jsonwebtoken** that will allow us, thanks to a secret key stored in the file .env, to verify that the user who makes a certain request has permission, the latter also allows us to set a period that keeps the Token valid.

We have decided to put a time limit of 1h, when we move around the pages of the application each time the /whoami endpoint is called which if there are 5 minutes left before the JWT expires, the first request made will have its period of validity extended.

If all goes well, the JWT will be returned to the client, and will be saved to be reused for the various requests, but if authentication fails, an error will be returned to the client.

Frontend

The frontend we created is located within the folder "taw", each component is located within the folders "src" and then "app", here the structure will be: all page components will have a folder, except for _services (contains the toast service), while the remaining "free" files will be all service components or modules.

The graphical interface was based on Bootstrap 5 components, which allowed us to make the application responsive with little effort, and to adapt it quickly to a large number of screens.

For the graphic management of the toasts we used ng-bootstrap, for the charts ng2-chart, and for the display of notifications with the number of them Angular Material.

For further customization and adaptation we used the .css files of the various components.

Service

For the frontend we used 3 main services, which were responsible for much of the application logic.

1. **user-http.service**
   It contains all the methods for interacting with the server regarding actions that the user can do, it makes all the http calls to the requested endpoint and returns the response to the component that requests its use, it also has the task of saving some variables that can be used by various components.
2. **socketio.service**
   Contains all the methods for sending or receiving messages from the server in real time, connecting to the Listeners with .on, or sending something to the server socket with .emit.
   Also contains some variables that are needed during the use of sockets by the user, especially during game/watch/cpu.
   It transforms the responses it receives into Observables.
3. **toast.service** (inside the _services folder)
   Contains methods to display and remove toast notifications.

Components

1. **sidebar component**
   This component is visible in all pages of the application and adapts dynamically depending on how you are authenticated, in fact if you are a moderator you will see the button to go to the all-user component, new-moderator and mod-chat, it contains the menu to navigate quickly.
   It also has the methods and display of the Floatingbutton "FriendList" which is a quick shortcut to display the friends list and interact with them.
2. **toast component**
   Methods and properties for displaying the Toast notification within the programme, which will disappear after 5 seconds. The text that appears will be dynamic, depending on what you want to display.

3. **all-user component**
   This component is displayed by only those who are moderators and allows you to view the list of all registered users with all moderators, each user is linked to various actions: logical deletion account, send message, display statistics.
   For users who are moderators, however, it is not possible for another moderator to delete the account, in fact the button to do so will not be visible.

4. **cpu component**
   This component has the task of displaying the game screen against the CPU, allowing you to perform all the moves, display those of the CPU and also ask for a suggestion on what move to make (only if you have an adequate ranking), it adapts dynamically depending on the difficulty chosen in the Homepage.

5. **friend-chat component**
   This component aims to show the user the chat between him and another user, you can see all the history of personal messages between the 2 users, in case one of the 2 users, or both, have blocked the other, the ability to send messages will be disabled.

6. **friend-stats component**
   The purpose of this component is to display the statistics of a user who is on the friends list. This component also displays other information about the user, such as their profile picture, e-mail address and their role in the application.

7. **game component**
   The purpose of this component is to display the game screen against another user, whether unknown or invited with a game request, and allows you to make moves when it is your turn.
   It also allows you to ask for hints (only if you have an adequate ranking) and also to send messages to your opponent during the game, these messages will also be displayed by observers, but players will not receive messages from observers.
   When the game ends, you will receive a message informing you who the winner is and how many points you got from playing that game.

8. **homepage component**
   The purpose of this component is to display the main page of the application, where you can choose to play or watch other players play.
   The game possibilities are divided into: play against the cpu with 3 difficulty levels to choose from (the default is AVERAGE), play against a stranger and at that point it will put us in a waiting point, and the possibility of inviting a user who

is among our friends list to play only if he is also online at that moment.
The spectator options are: to view the game of any user who is currently playing (whether against another user or against the CPU), while when we want to view the game of a user who is in our friends list we can choose from a list of games in progress in which they are participating and view it (whether against another user or against the CPU).

9. **mod-chat component**
   This component has the purpose of displaying to all users a list with all available moderators (excluding himself, if he is also a moderator) allowing to start a chat with that particular moderator, this chat will also be real-time (via sockets) if both are online and writing at that time.

10. **new-moderator component**
    This component has the purpose of displaying only to the moderators the page to create a new moderator with a username and password (temporary), which at the time of the first login will ask you to change, also entering the information to have a complete profile

11. **user-login component**
    The purpose of this component is to display the page for the user to enter the data to access the application, if there are any errors with authentication this will display an error message.
    If all goes well, the user will be redirected to the Homepage.

12. **user-logout component**
    This component does not have any display purpose, but only has the task of disconnecting the user from the saved session, and then allow him to start a new one and redirect him to the login page.

13. **user-profile component**
    This component has the purpose of displaying the summary of the profile of the connected user, allowing him to see a small graph with the history of his ranking, the details entered previously at the time of registration, being able to choose to change some details of his account.

14. **user-signin component**
    This component displays a user's registration screen, allowing the user to enter all their personal details, a password, email, username and even a link to a picture as their profile picture.

15. **watch component**
    The purpose of this component is to display to users who want to spectate a match the playing field between the user and another player (another user or the CPU), it will also display the chat between the 2 players, and all spectators can interact by sending messages to each other (players will not see these messages) through the chat, when the game ends you will receive a message informing you who will be the winner of the game.

Routes

1. **PATH**: ” ”

   redirectTo: “login”

   patchMatch:”/full”
2. **PATH**: ”**”

   redirectTo: “login”

   patchMatch:”/full”
3. **PATH**: ”/login ”

   COMPONENT: UserLoginComponent
4. **PATH**: ”/home”

   COMPONENT: HomepageComponent
5. **PATH**: ”/logout”

   COMPONENT: UserLogoutComponent
6. **PATH**: ”/signin”

   COMPONENT: UserSigninComponent
7. **PATH**: ”/profile”

   COMPONENT: UserProfileComponent
8. **PATH**: ”/new-mod”

   COMPONENT: NewModeratorComponent
9. **PATH**: ”/all-user”

   COMPONENT: AllUserComponent
10. **PATH**: ”/game”

COMPONENT: GameComponent

11. **PATH**: ”/watch”

    COMPONENT: “WatchComponent

12. **PATH**: ”cpu”

    COMPONENT: CpuComponent

13. **PATH**: ”/user-stats/:friend”

    COMPONENT: FriendStatsComponent

14. **PATH**: ”/friend-chat/:friend”

    COMPONENT: FriendChatComponent

15. **PATH**: ”/mod-chat/:user”

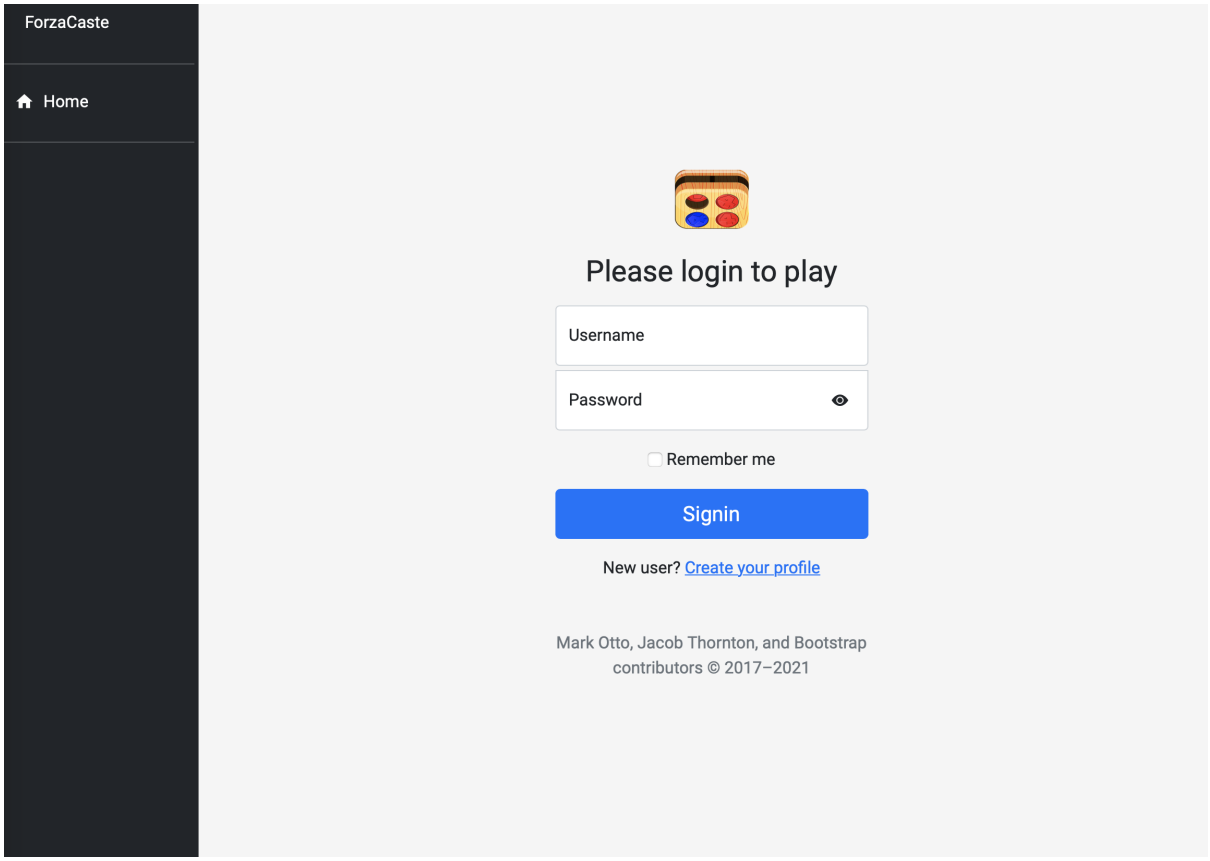
    COMPONENT: ModChatComponent

Application workflow

This section shows how the application works by means of screenshots that will allow you to understand in general how to use it and what functions it implements.

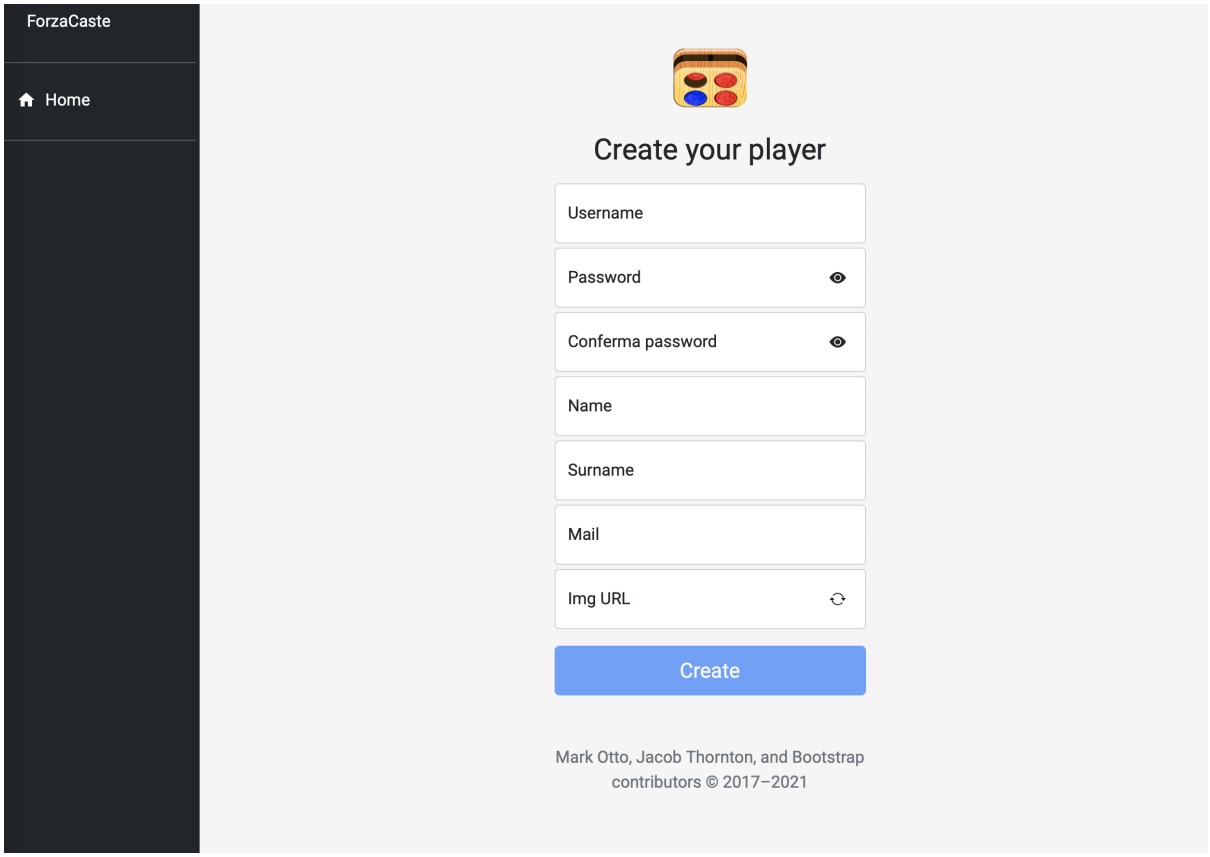
You can also test a live version at: forzacaste.live

LOGIN PAGE



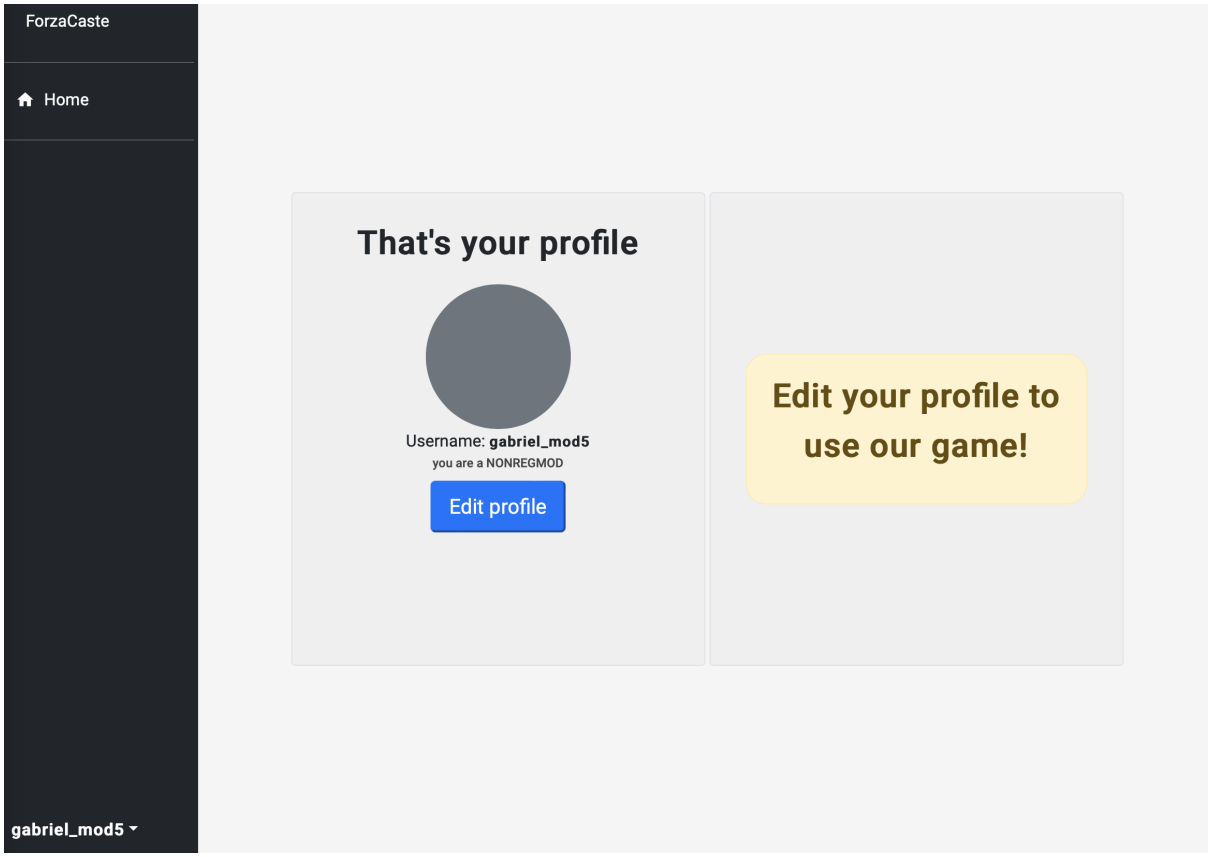
> You can login with your account and remember it, or create a new one

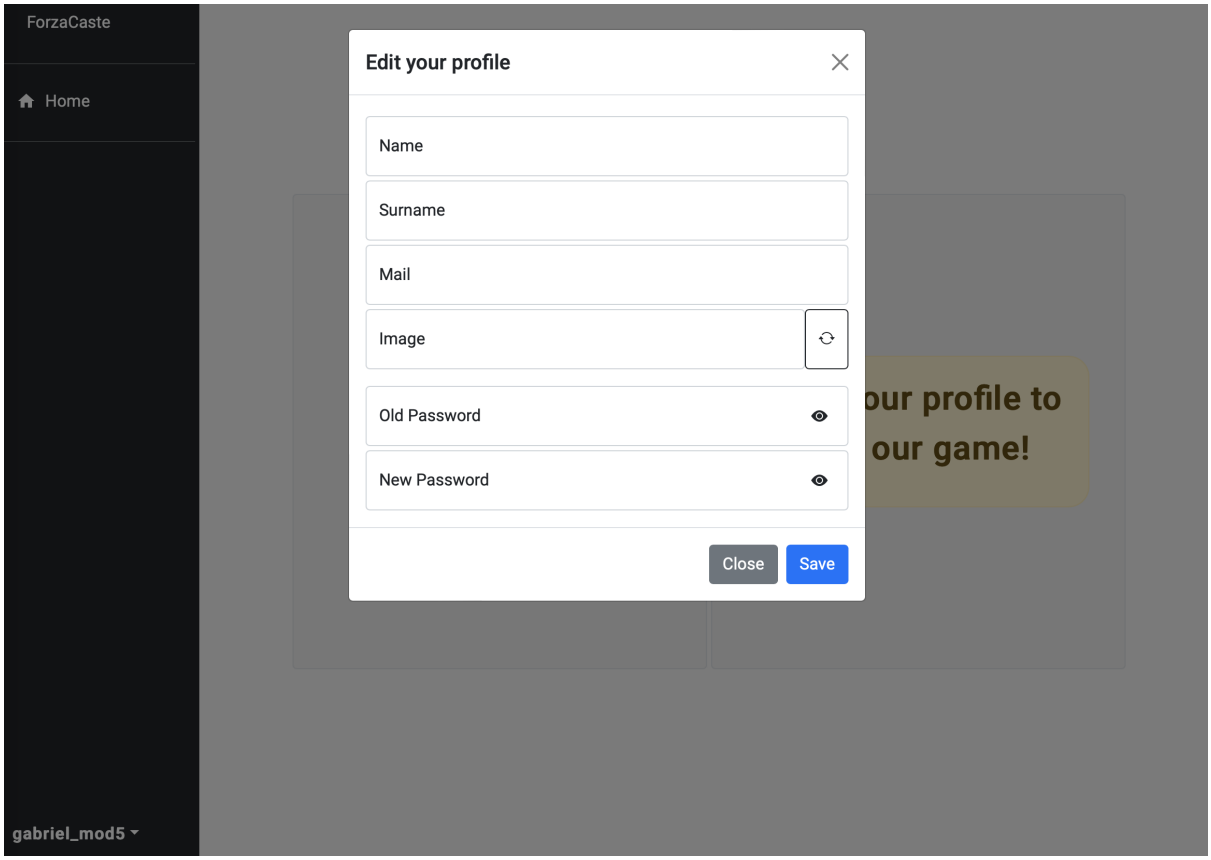
REGISTRATION PAGE



> You can register with you email and your credential, you can also enter additional information to help you recognise yourself in the application, such as a profile picture.

FIRST LOGIN AS MODERATOR

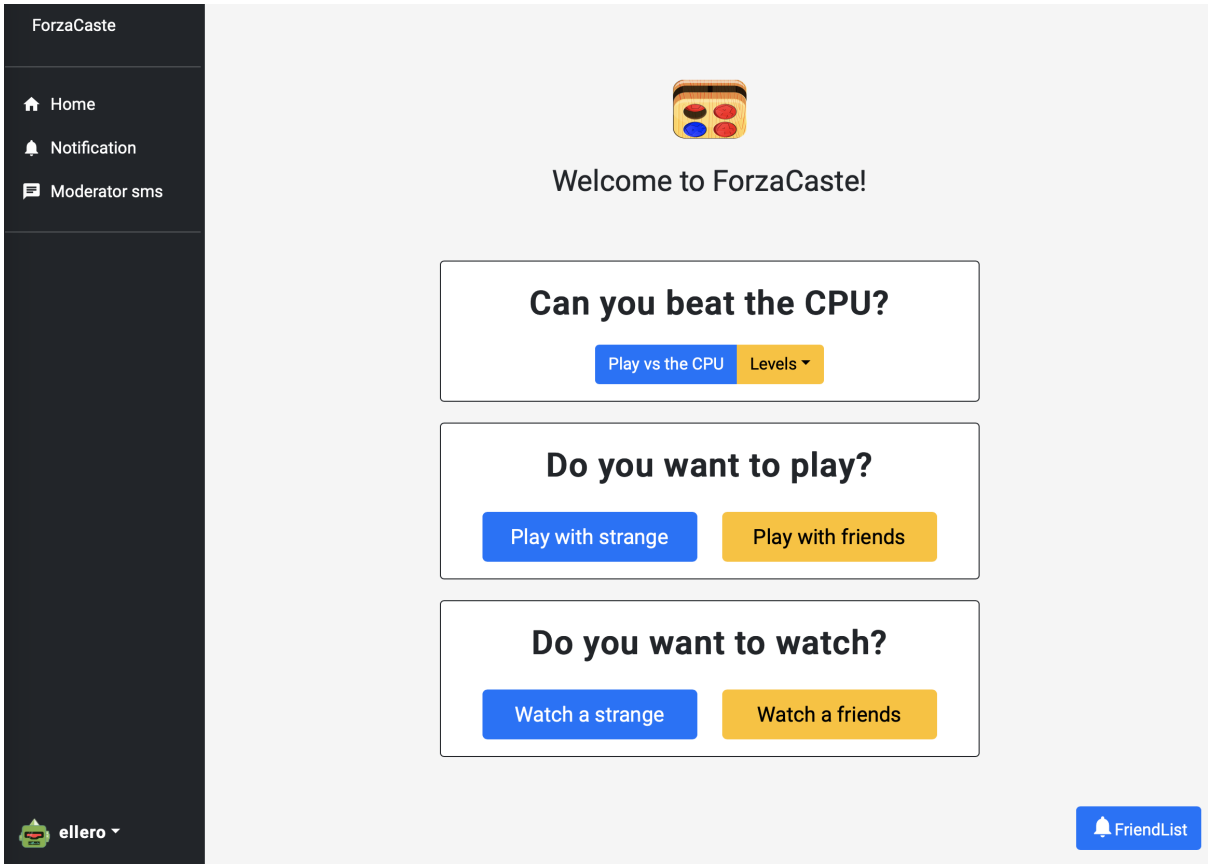

ForzaCaste
Home
That's your profile
Username: gabriel_mod5
you are a NONREGMOD
Edit profile
Edit your profile to use our game!
gabriel_mod5


ForzaCaste
Home
Edit your profile
Name
Surname
Mail
Image
Old Password
New Password
Close
Save
gabriel_mod5

> If a new moderator is created and logs in for the first time, the first page he/she will see is the edit profile page where he/she can enter all his/her personal data and change the password.

Homepage

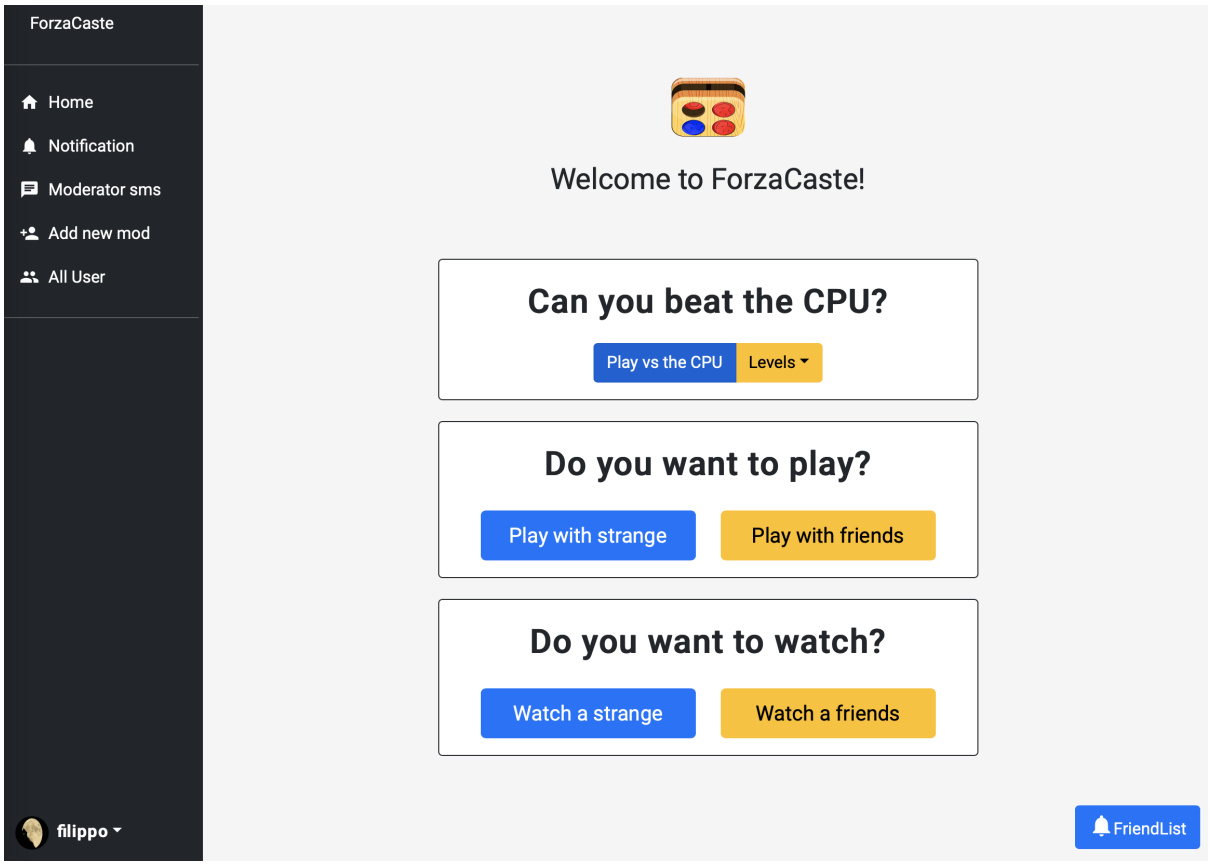
HOMEPAGE



> On the homepage you can decide what to do, play with the CPU in the default mode or by selecting the difficulty level, with a stranger or with your friend, or watch your friend's or a stranger's game.
> You can also go to your profile and see your stats, or start a conversation with your friend with the friend list button or see their profile and stats.

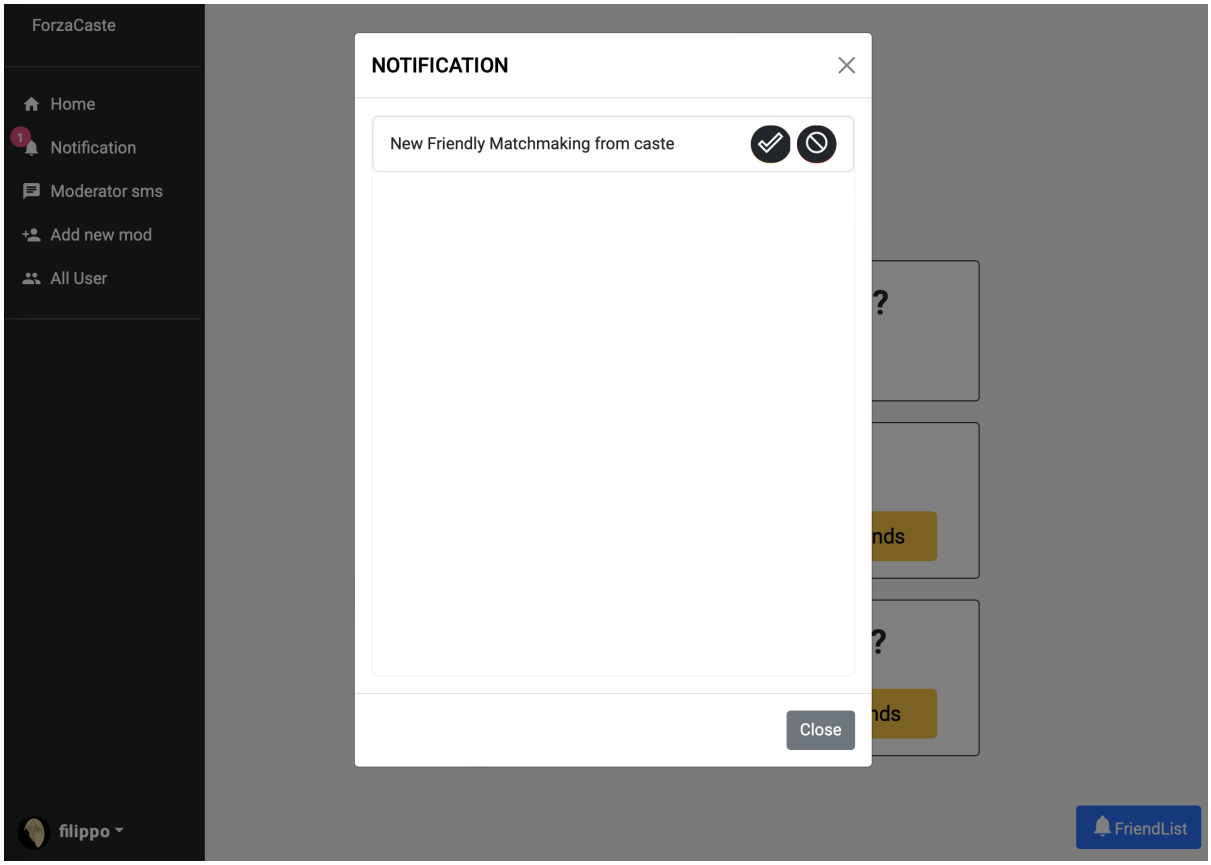
> If you need to, you can start a conversation with a moderator by clicking the button on the sidebar.
> You can also respond to notifications you receive by accepting or deleting them from the sidebar.

HOMEPAGE AS MODERATOR



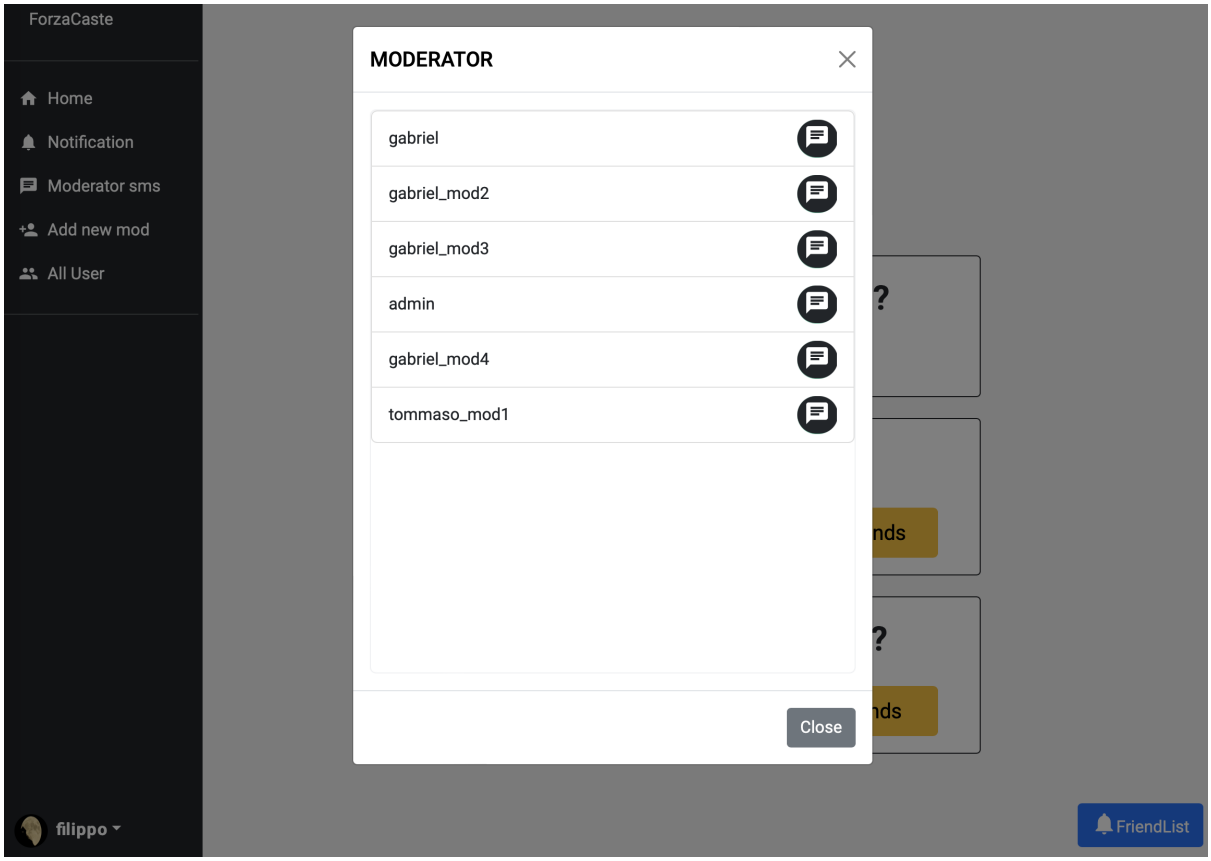
> If you are a moderator, the homepage screen will be slightly different. In fact, in the sidebar we will have two new buttons which will allow us to create a new moderator and to display all the users who are registered on the platform and to perform actions on them.

NOTIFICATION



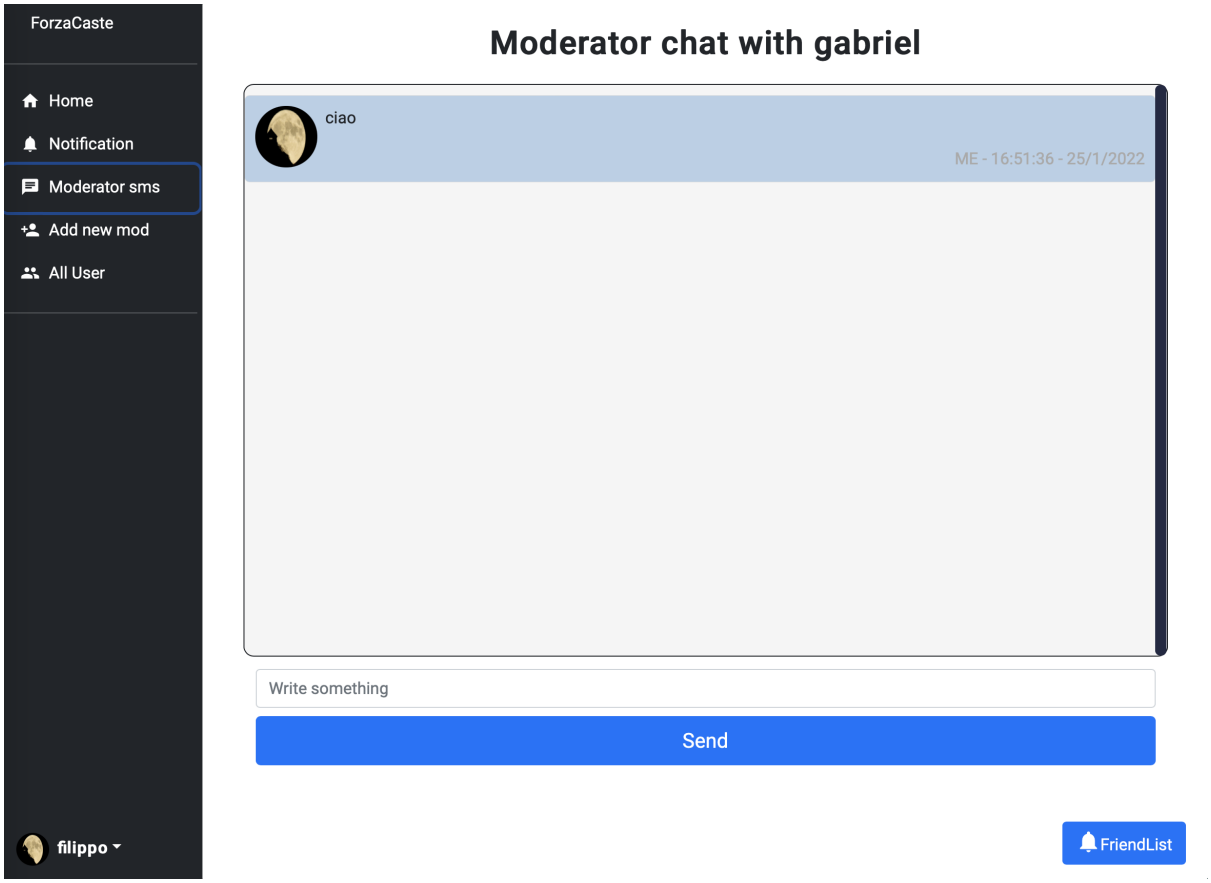
> Clicking on the notification button in the sidebar will bring up a dialogue with a list of all the notifications you have received and on which you have not yet taken an action, here you can receive notifications about friendship requests or game requests from a friend

MODERATOR SMS



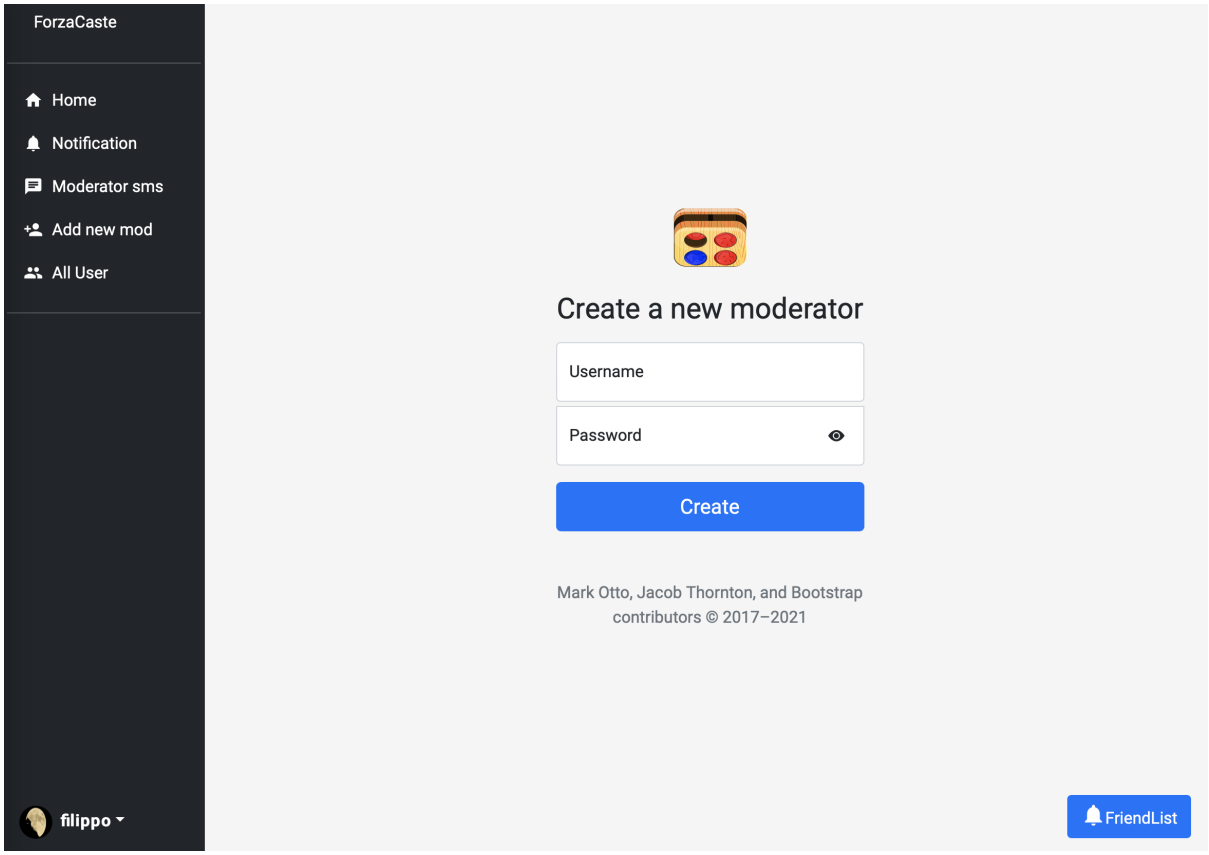
> If you click on the moderator sms button in the sidebar, a dialogue will appear with a list of all the application's moderators with whom you can start a conversation.

MODERATOR CHAT



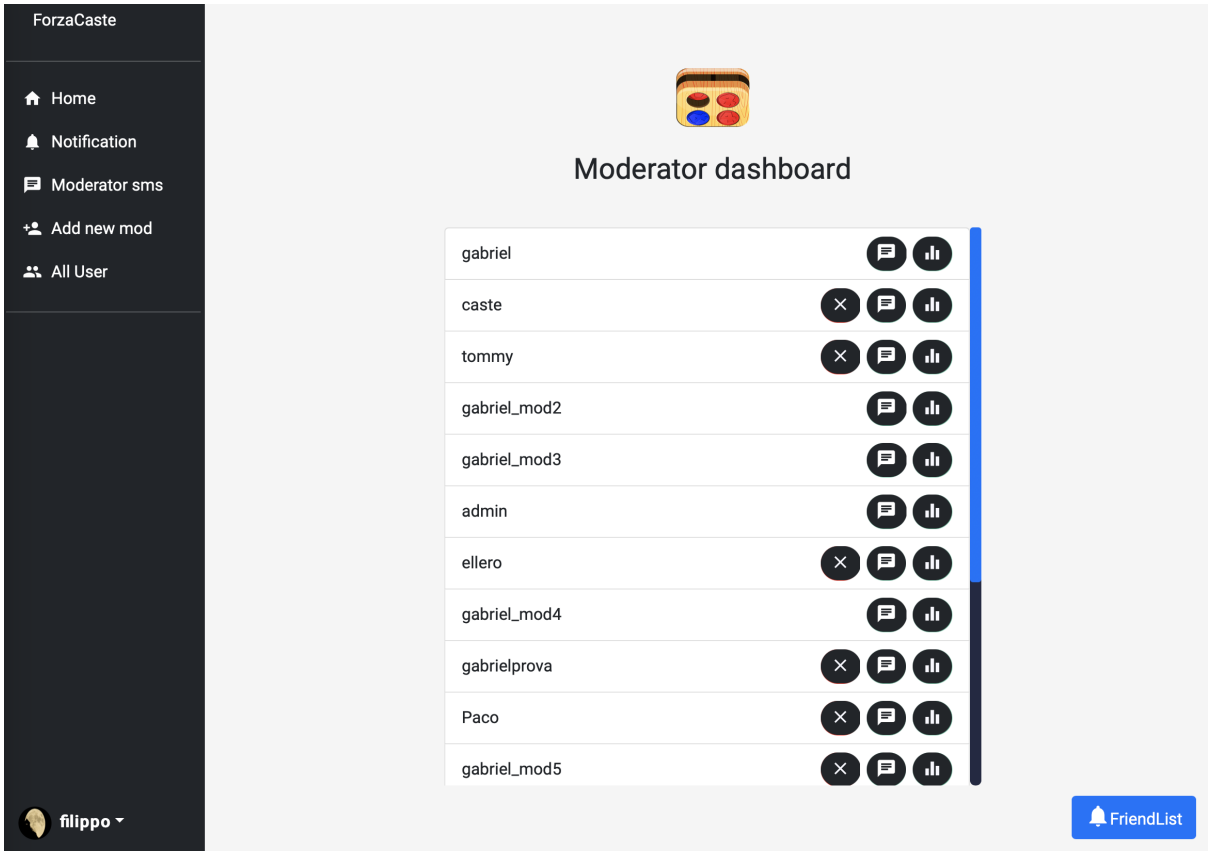
> When a moderator decides to start a chat with a registered user, he or she will have a chat facility which, if both are online, will be real-time, otherwise the next time the user logs in he or she will see all the messages.

ADD NEW MODERATOR (ONLY MODERATOR)



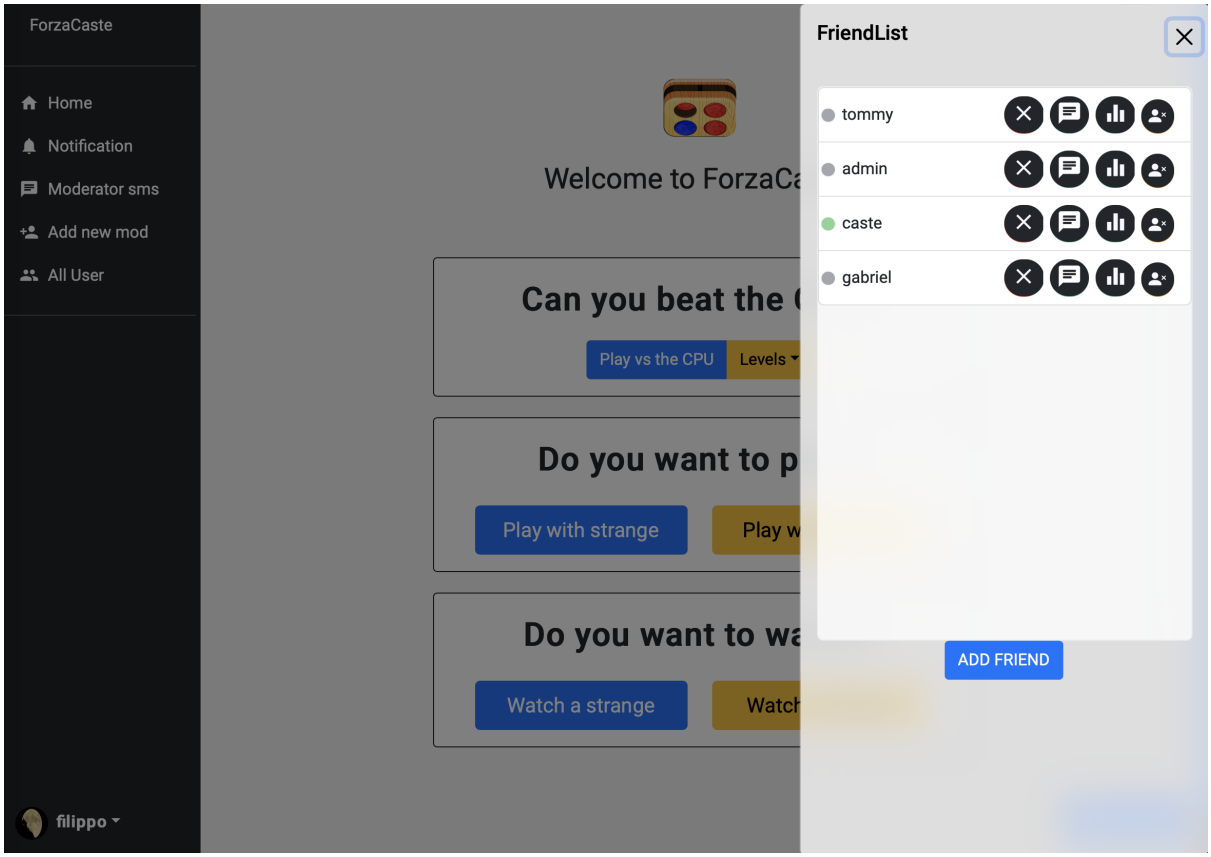
> Clicking on the add new mod button in the sidebar will take us to a new page where we can add the minimum details for adding a new moderator, only moderators can do this.

ALL USER (ONLY MODERATOR)



> Clicking on the all user button on the sidebar will take us to a new page where we can perform various actions for each user, such as deleting them (logically), viewing their statistics or starting a chat with that user, only moderators can access this page and no moderator can delete another moderator, in fact the button for that particular action will not be visible

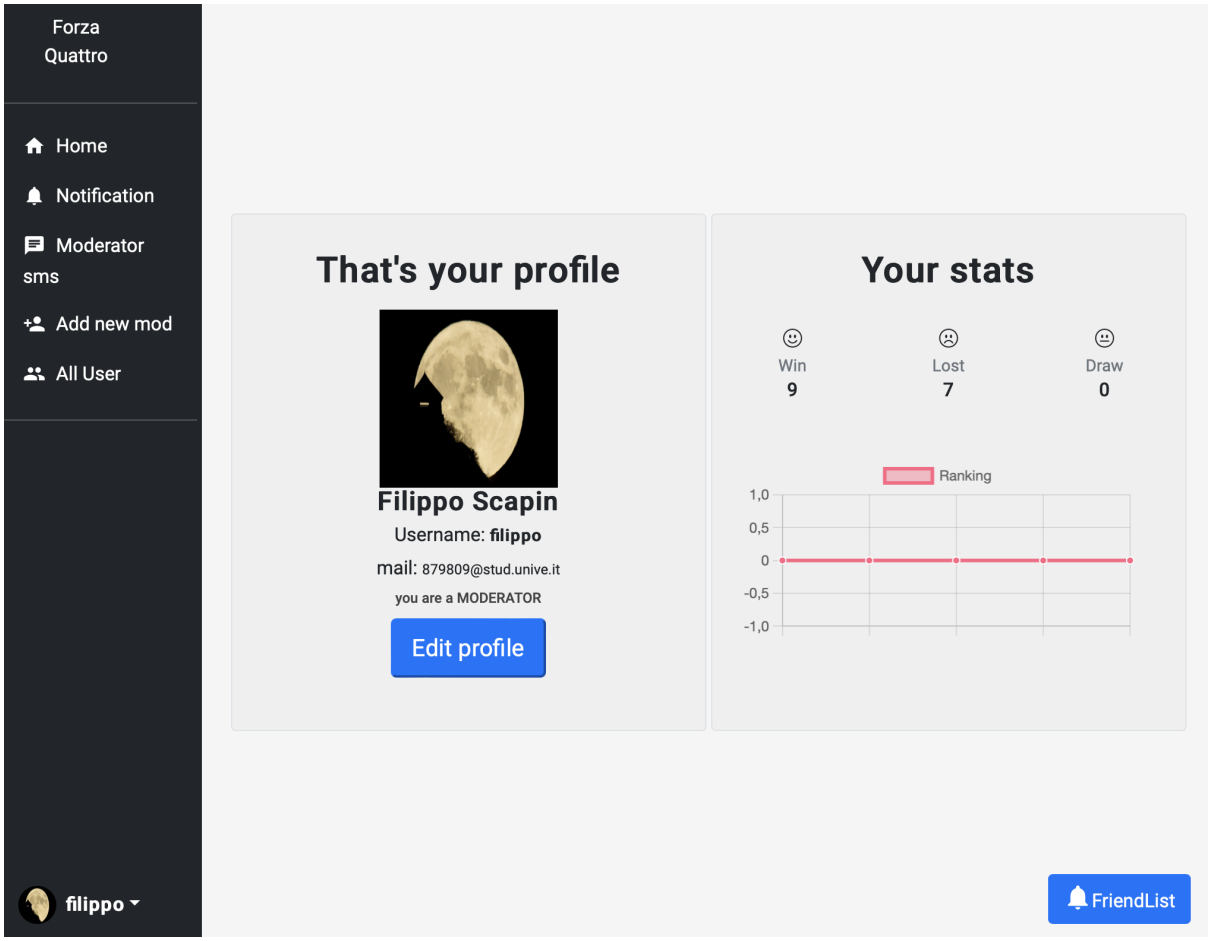
FRIENDLIST



> By clicking on the button at the bottom right (not visible during a game), you will be able to view the user's entire friend list and also see notifications from each of them.
> You will also see the user's status, whether they are online or offline, and you will be able to access a summary page for that friend to view some of their information, such as their statistics.

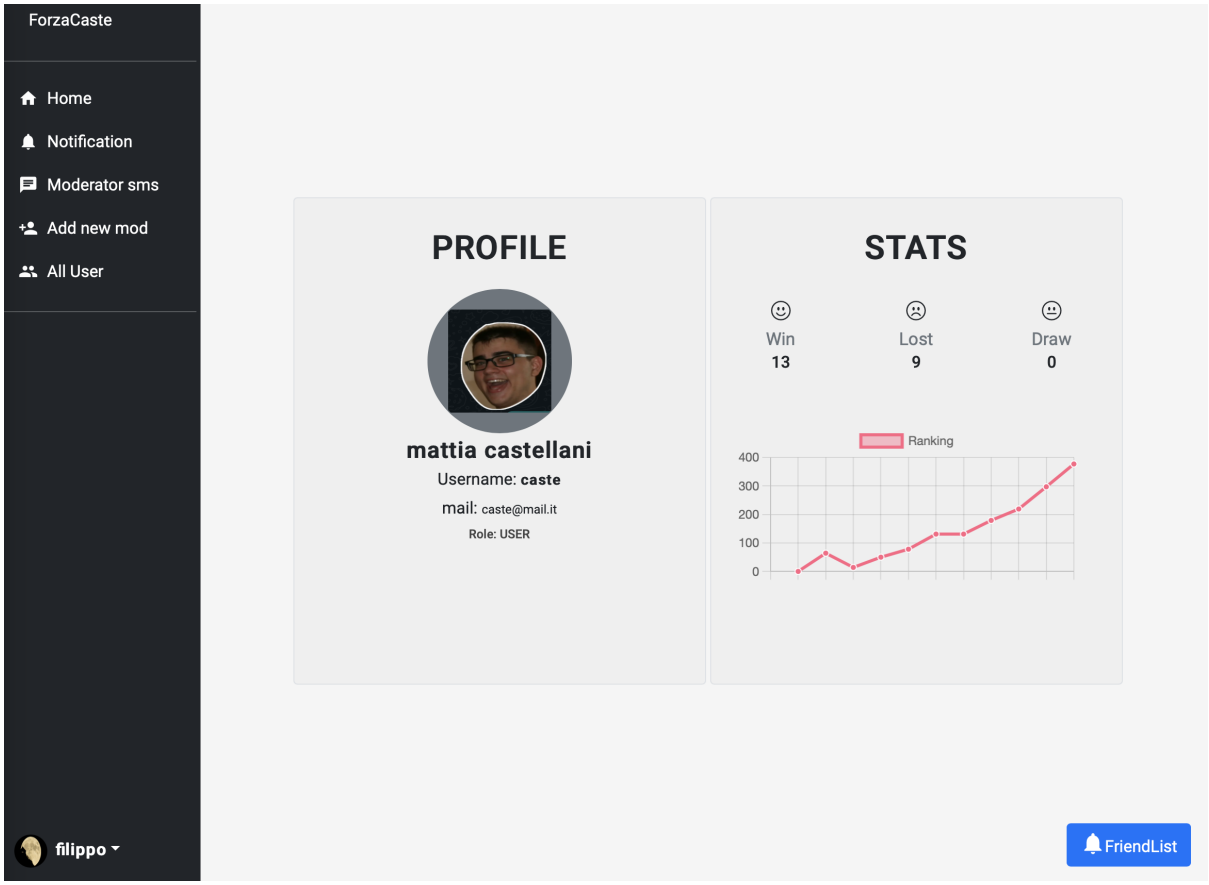
PROFILE

PROFILE PAGE



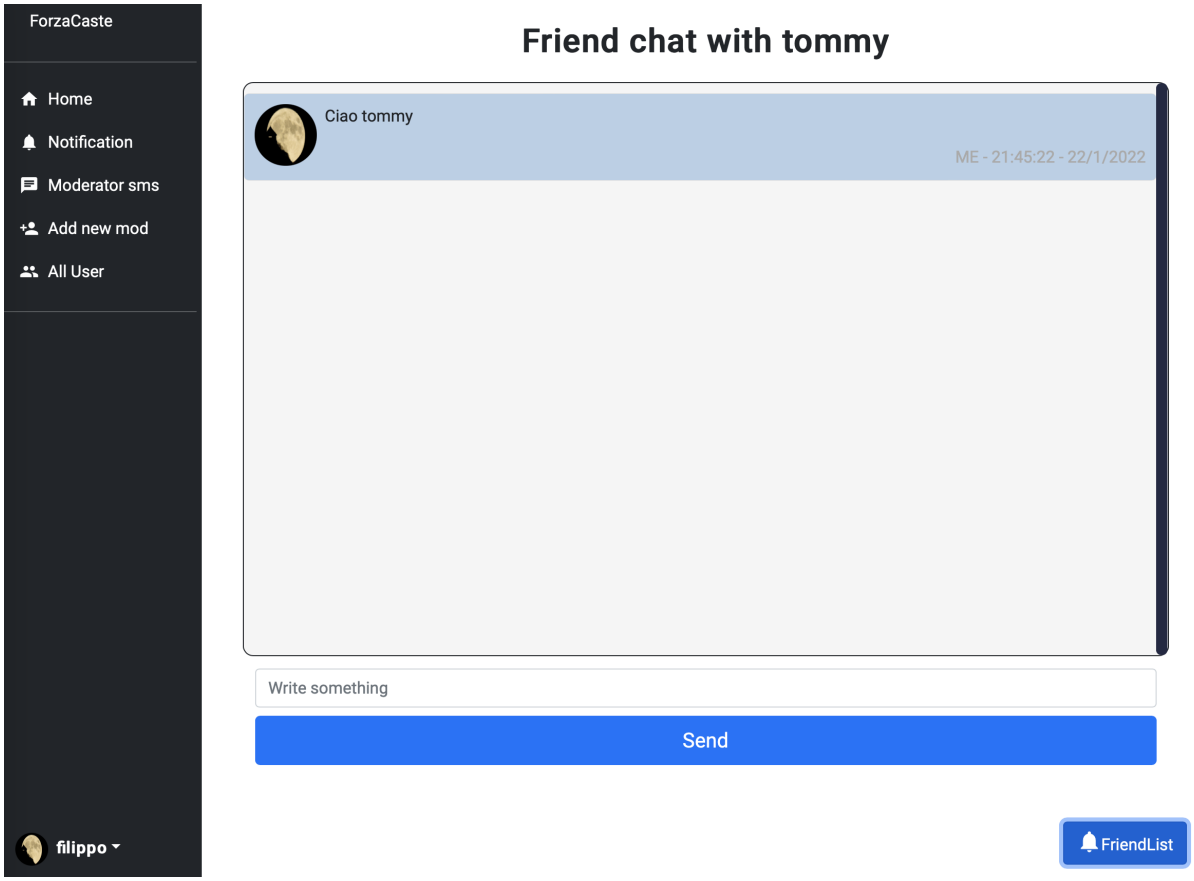
> When you click on our name on the sidebar and click on "Profile", from here you will access the profile of the user who is logged in at that moment, from this page the user will be able to see a summary of his account, such as statistics and a graph of the progress of his last matches; he can also edit his profile information by clicking on "Edit profile".

PROFILE VIEW MODE



> When we are on Friendlist and we want to view a friend's profile, we will click the third button from the left and we will be directed to a new page where we can see a quick summary of that user's profile.

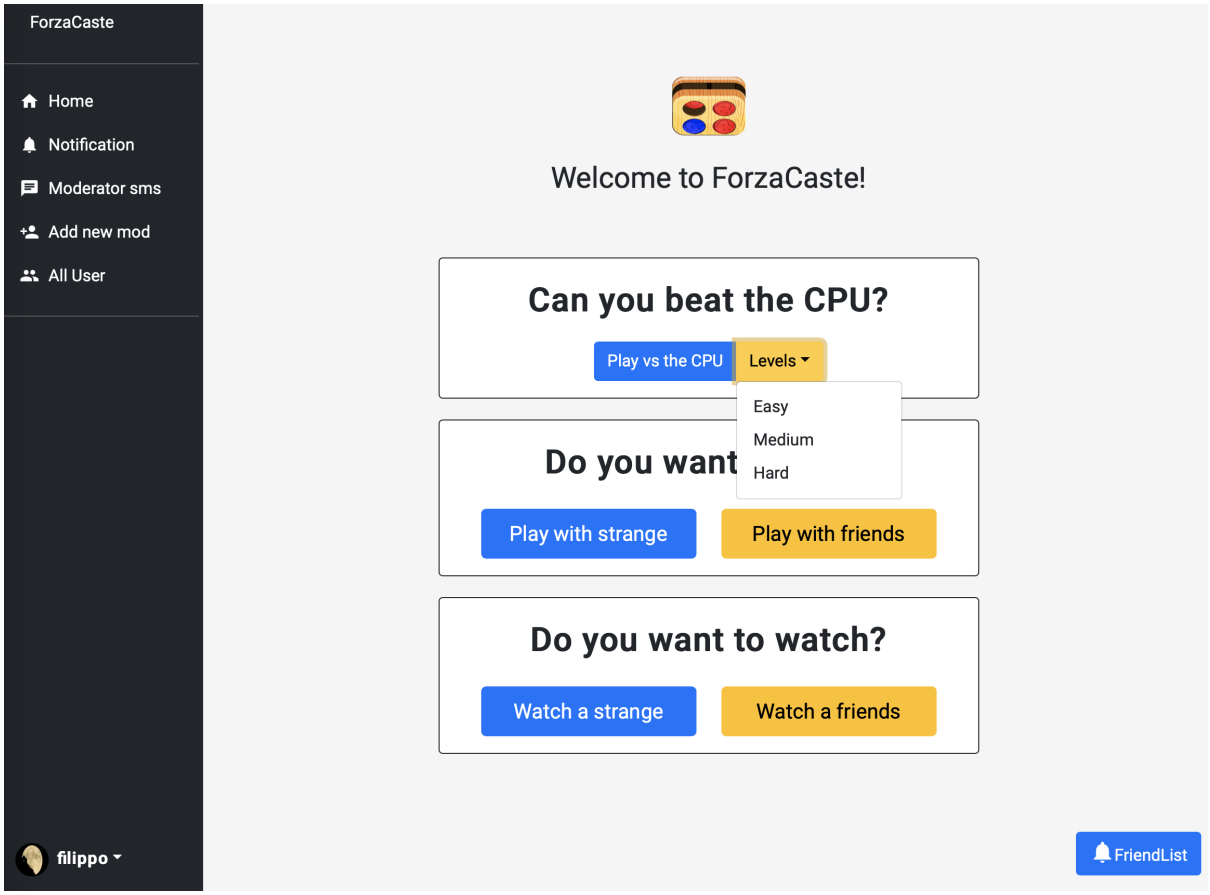
Chat with friend



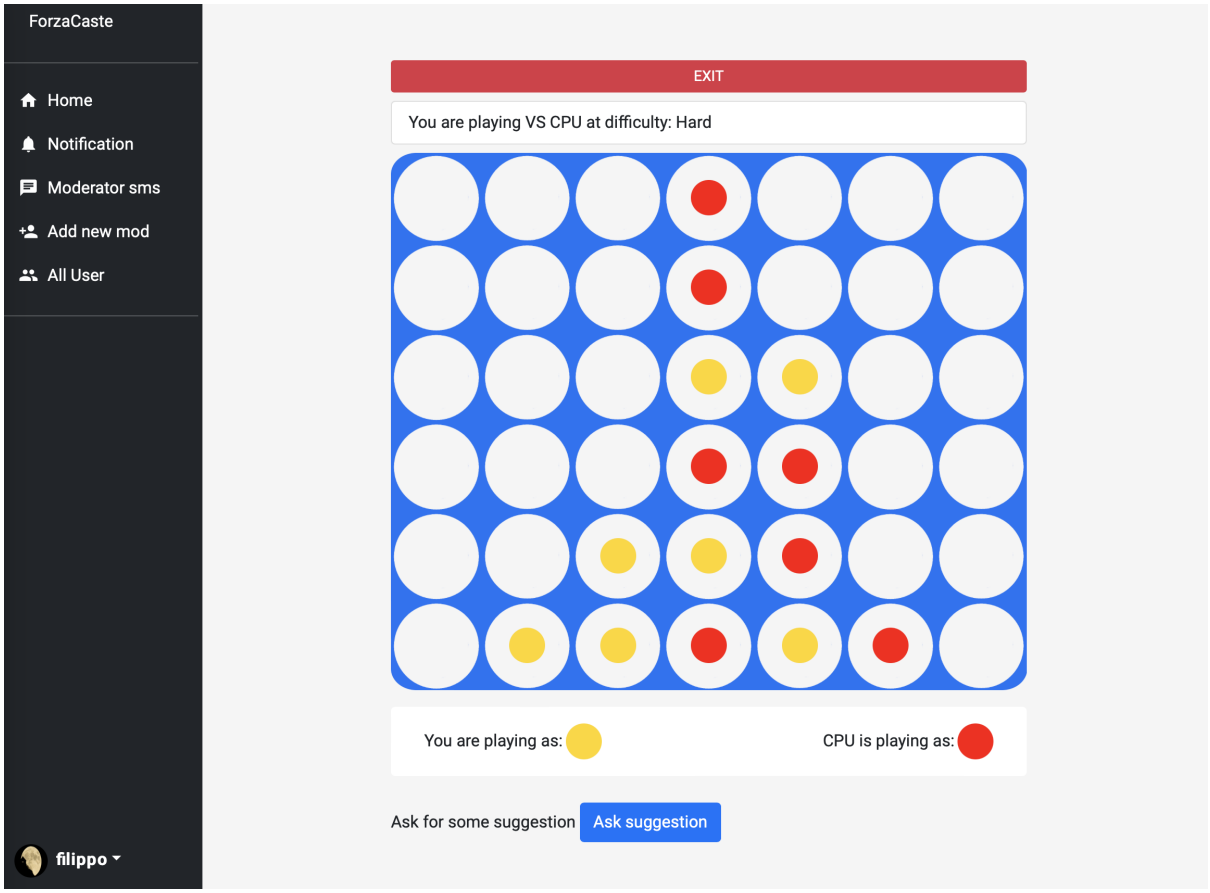
> When we decide to start a conversation with one of our friends, or resume one already started, by reading the messages sent by one of our friends, we will have to click on the FriendList button and click on the second button on the left, which will take us to a new page where we can interact with our friend, sending messages that he or she will be able to see at the next access or in real time.

Game

GAME WITH CPU

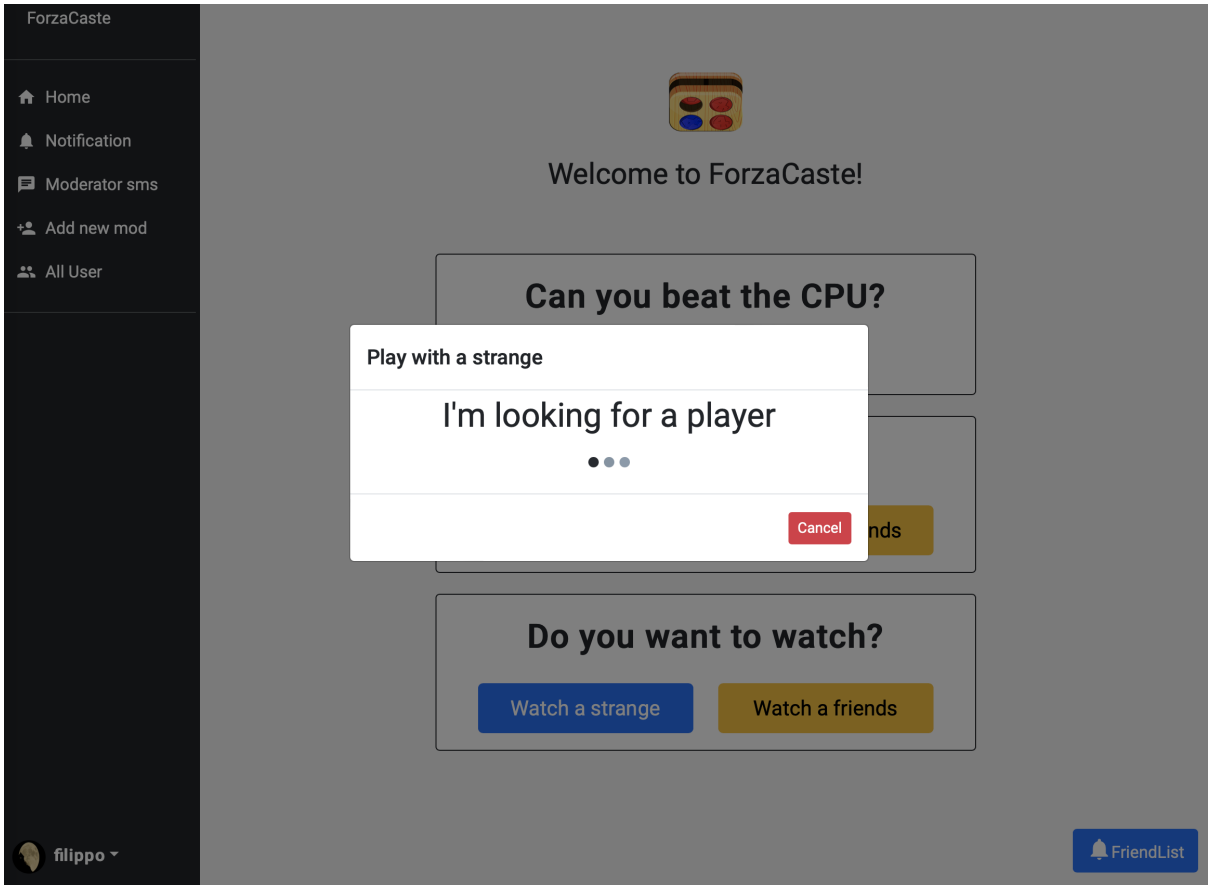


> When you want to start a game against the cpu you have to click on the appropriate button on the homepage, which will start a standard game with a medium difficulty, in case you want to choose the difficulty, there are 3 levels of difficulty: Easy, Medium, Hard, you must click on the button immediately to the side and it will start automatically, taking us to the game page



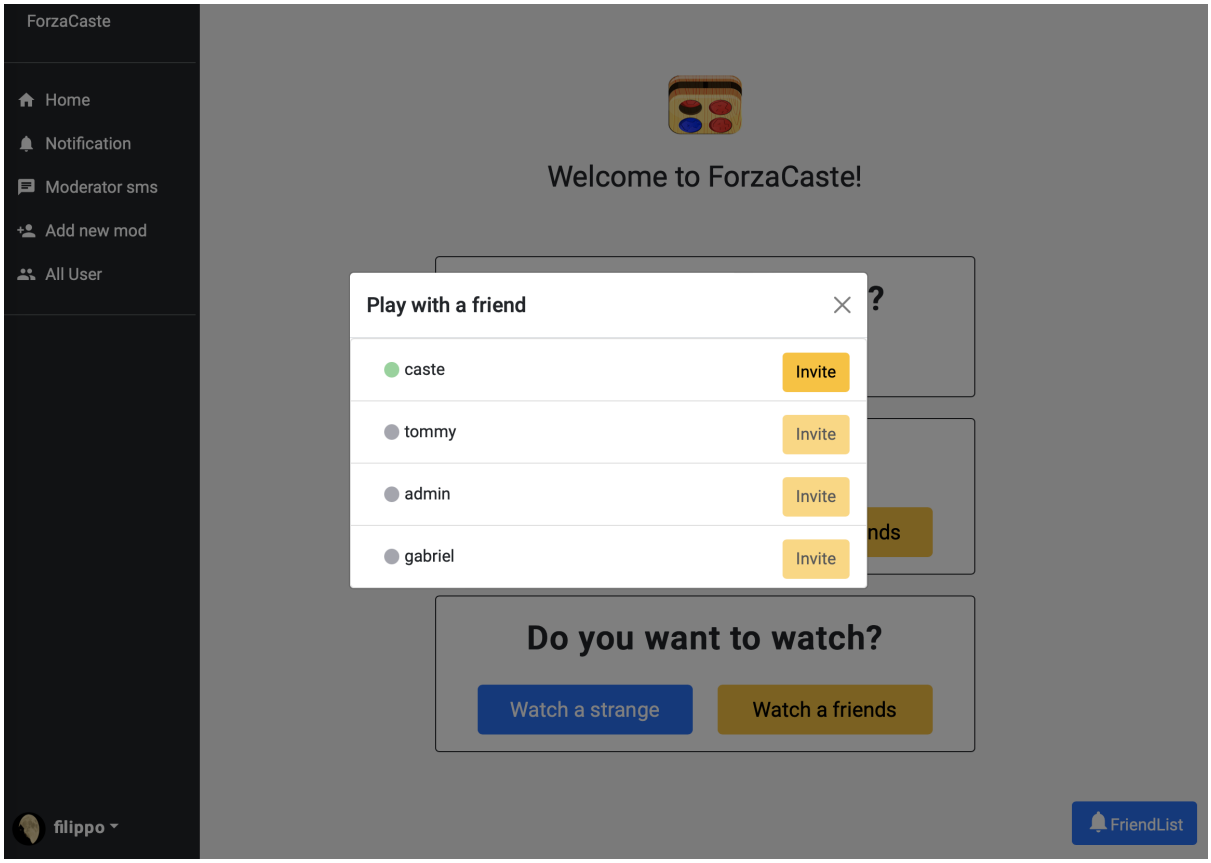
> Once the game has started we can make our moves and see who we are playing against, the CPU, what level, Hard, and the colour of our pawns, yellow for us, and red for the CPU, if you have a high enough ranking (100+) then you can ask for a suggestion on what move to make.

GAME WITH STRANGE

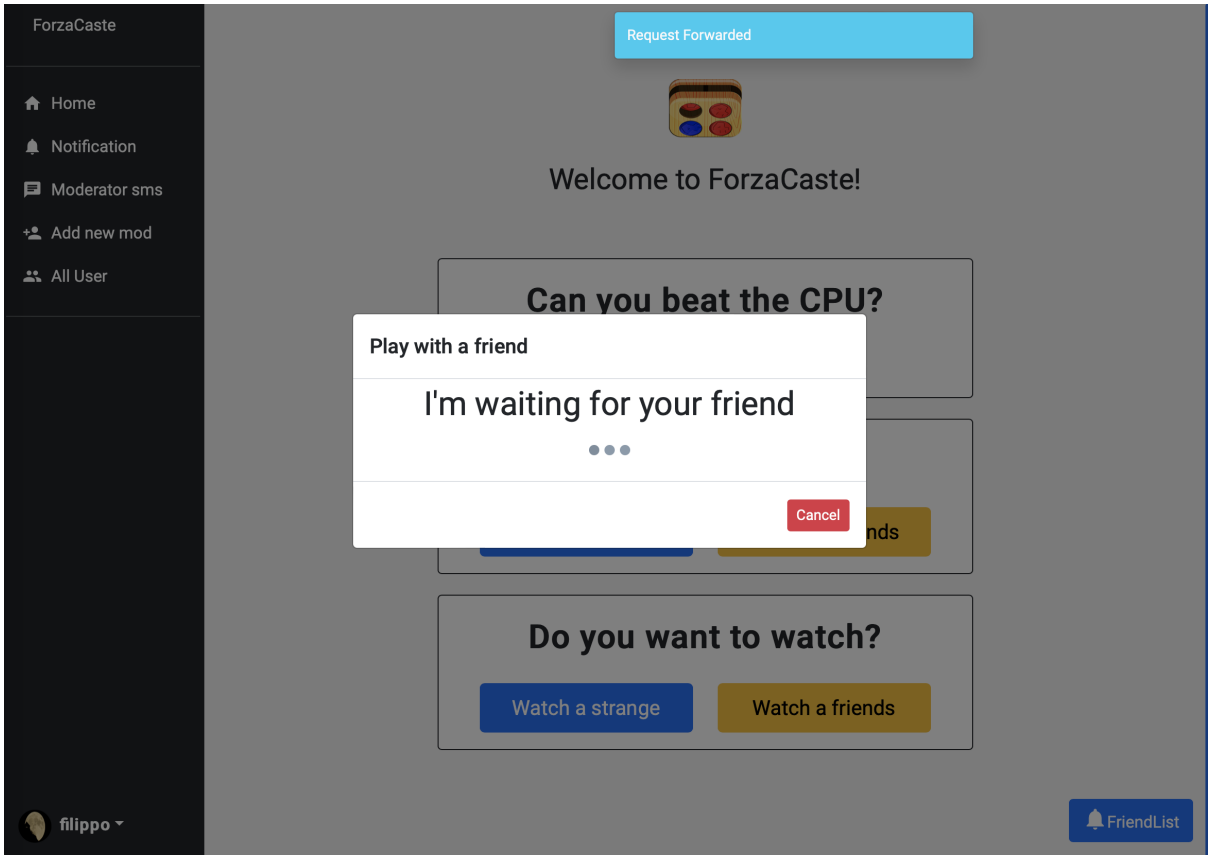


> When you want to start a match against a stranger, you need to click on the Play with strange button on the homepage, which will open a dialogue that will put the user on hold until they find a match with another user. Matchmaking depends on the user's ranking, so they will tend to look for another user with a similar ranking to them.
> If the user gets tired of waiting, they can click the cancel button, which will close the matchmaking phase and allow the user to perform other actions.
> Should a match be found, the user will be redirected to the game screen.

GAME WITH FRIEND

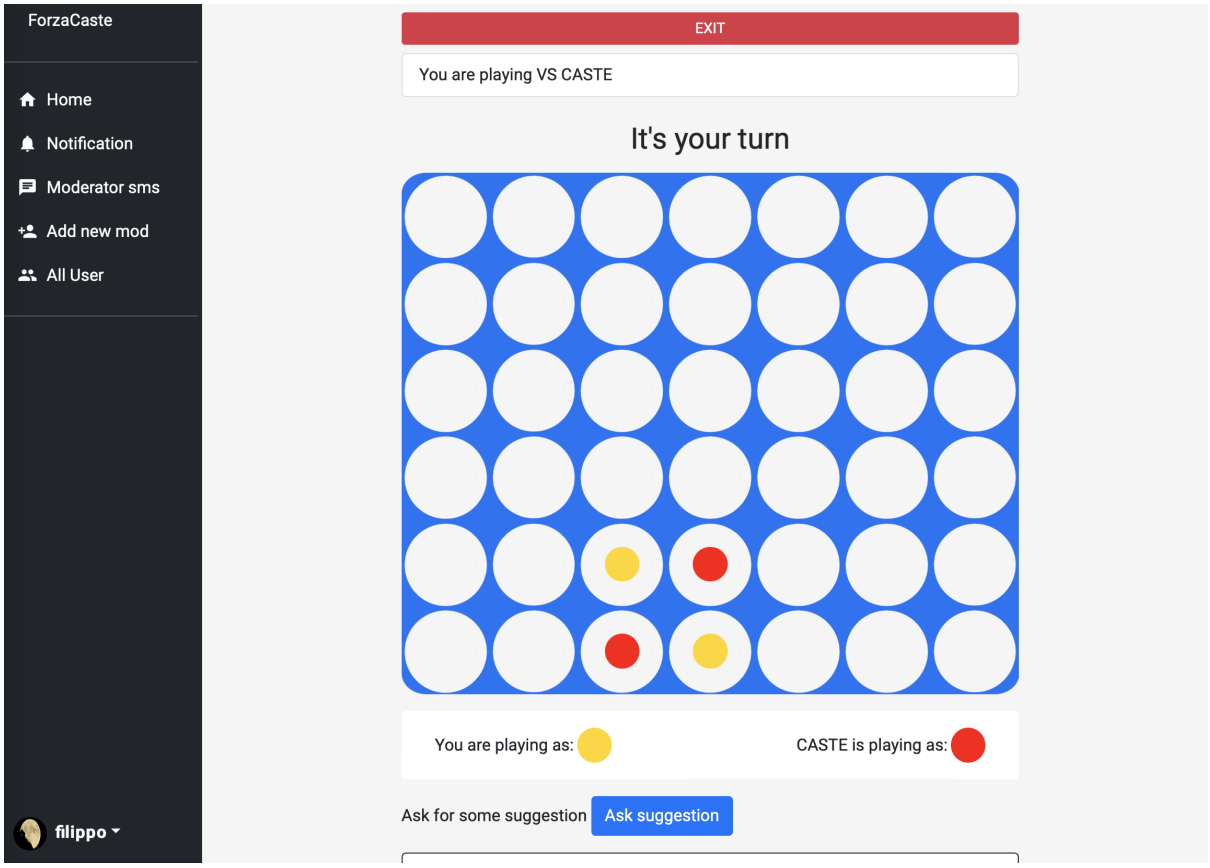


> When you want to invite a friend to take part in a game, click on the Play with friend button and a dialogue will open with a list of friends to invite. Only online friends can be invited, all other users will not have the invite button enabled.

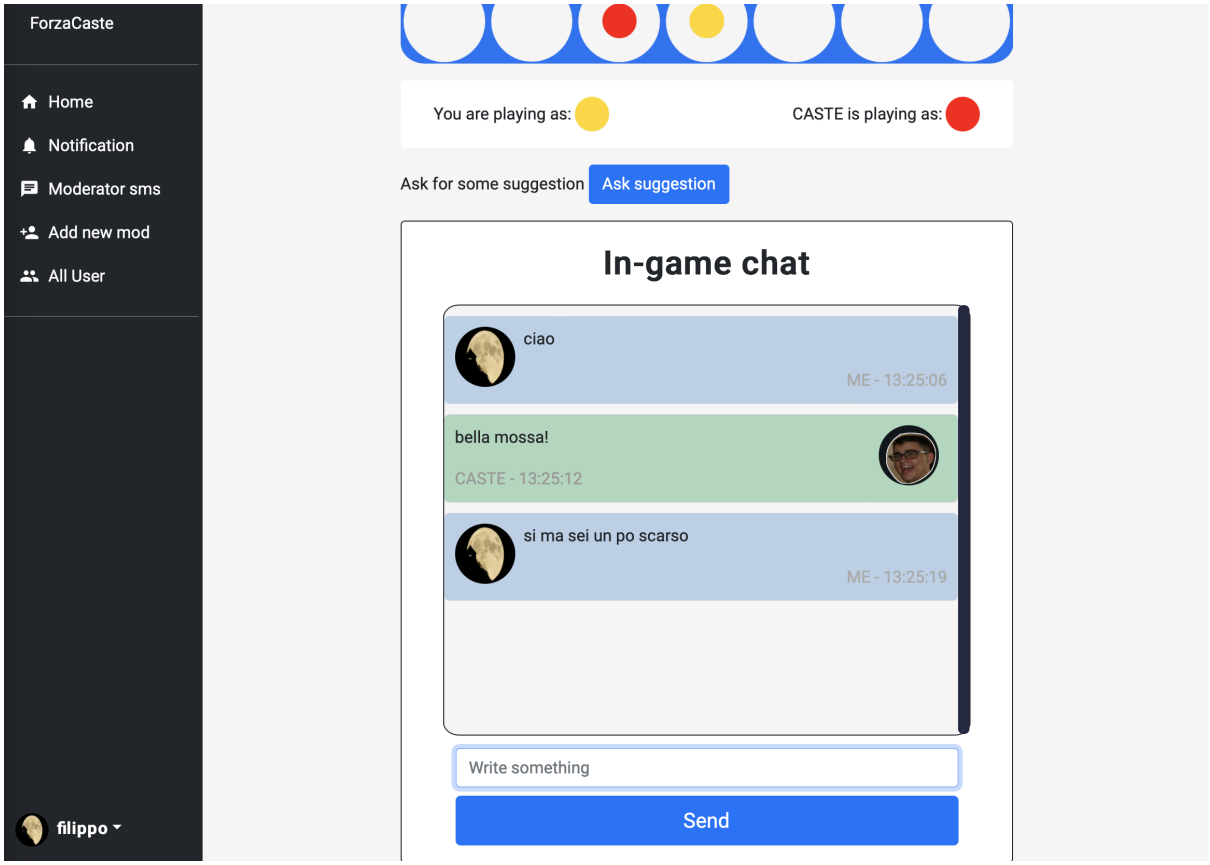


> Once the user clicks on the invite button, the user will be put on hold until the other user has taken an action, either accepting or rejecting the request to play, in which case the user will be denied the request, the screen will be closed and the user will be able to take any action they wish.
> If the user gets tired of waiting, they can click the cancel button, which will close the Matchmaking phase and allow the user to perform other actions.
> If the request is accepted, the user will be redirected to the game screen.

GAMEBOARD (same for strange and friend)



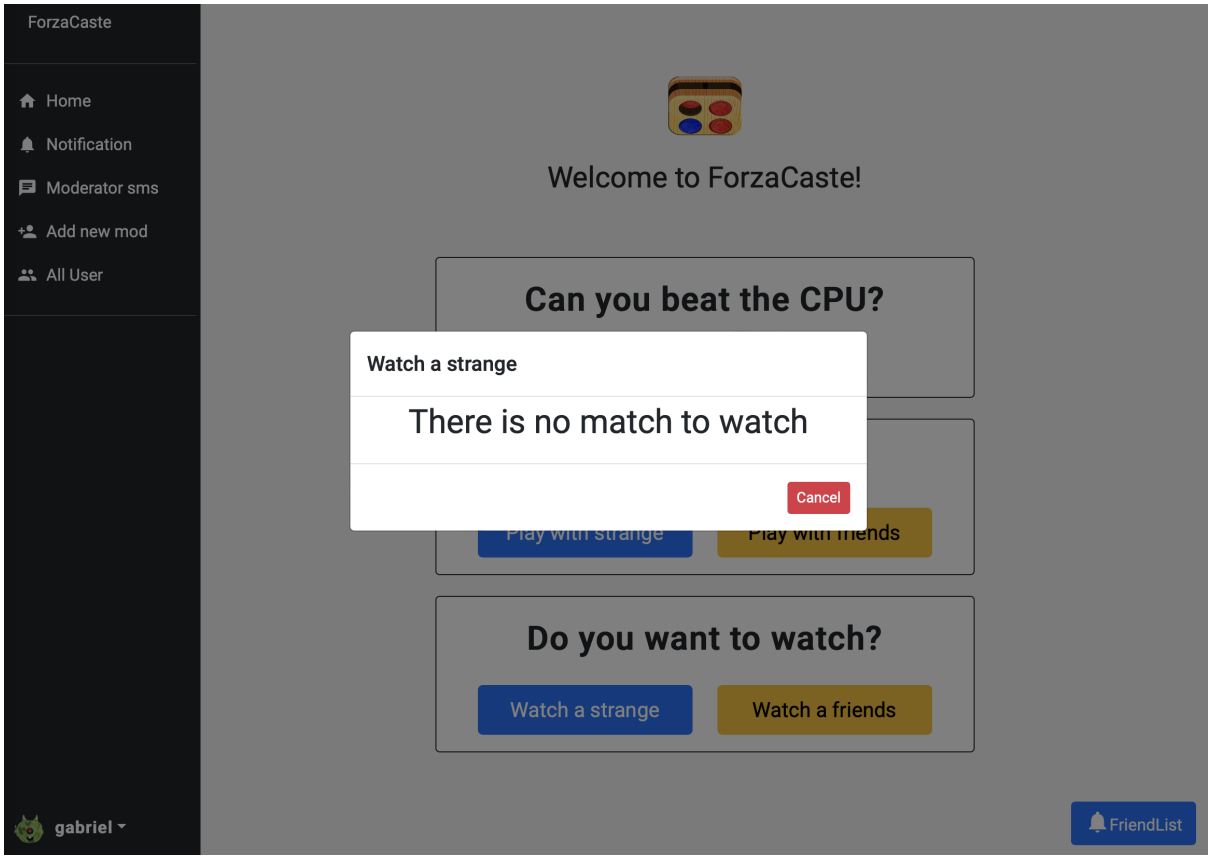
> Once you start a game against a stranger or against a friend the game screen will be identical for both cases, only if the other player is not our friend will appear a button that will allow us to ask for friendship, in fact we can see who we are playing against, Caste, the colour of our pawns, yellow for me, red for caste, and if we have a high enough ranking (100+) we can ask for a suggestion on what move to make, disabled when it is not our turn.



> In case you want to interact with the other player during the game, you can send messages in real time, writing whatever you want, in case there are observers at the game the 2 players will not receive messages from them.

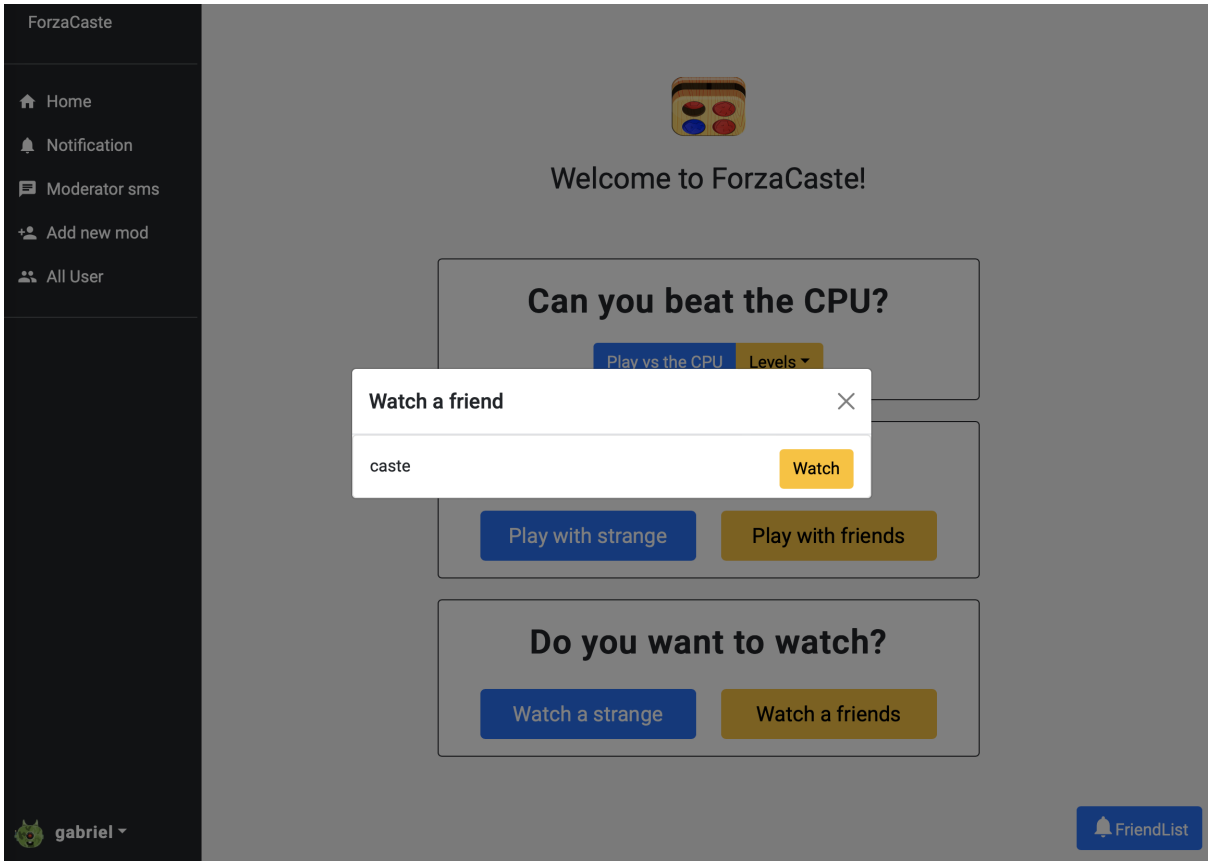
Watch

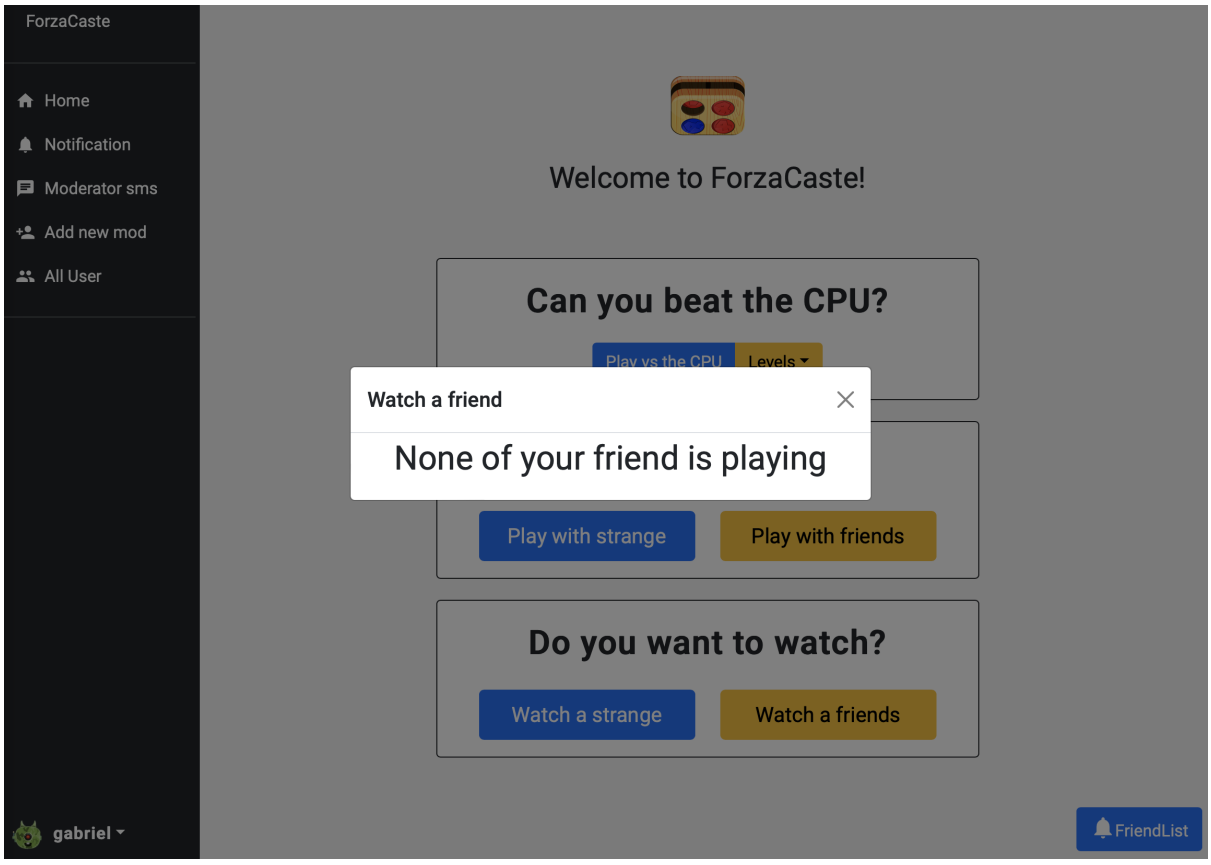
WATCH A STRANGE



> When you want to watch the match of 2 other players you have to click on the Watch a massacre button on the homepage screen, which will open a dialog that will load all the matches in progress and choose one at random to access, if there are no matches in progress will inform the user with a message, when you access a match to watch, the user will be sent to the watch page.

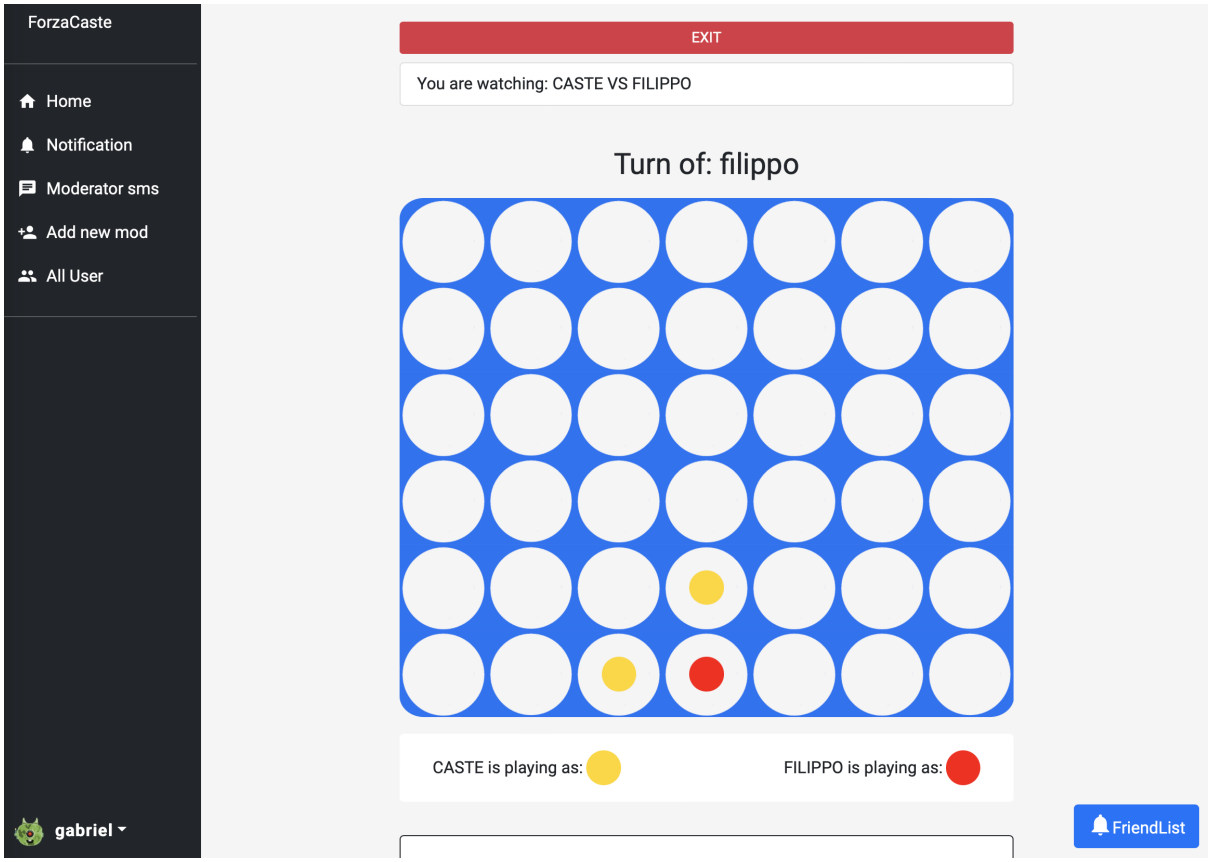
WATCH A FRIEND


ForzaCaste
Home
Notification
Moderator sms
Add new mod
All User
gabriel
Welcome to ForzaCaste!
Can you beat the CPU?
Play vs the CPU
Levels
Watch a friend
caste
Watch
Play with strange
Play with friends
Do you want to watch?
Watch a strange
Watch a friends
FriendList


ForzaCaste
Home
Notification
Moderator sms
Add new mod
All User
gabriel
Welcome to ForzaCaste!
Can you beat the CPU?
Play vs the CPU
Levels
Watch a friend
None of your friend is playing
Play with strange
Play with friends
Do you want to watch?
Watch a strange
Watch a friends
FriendList

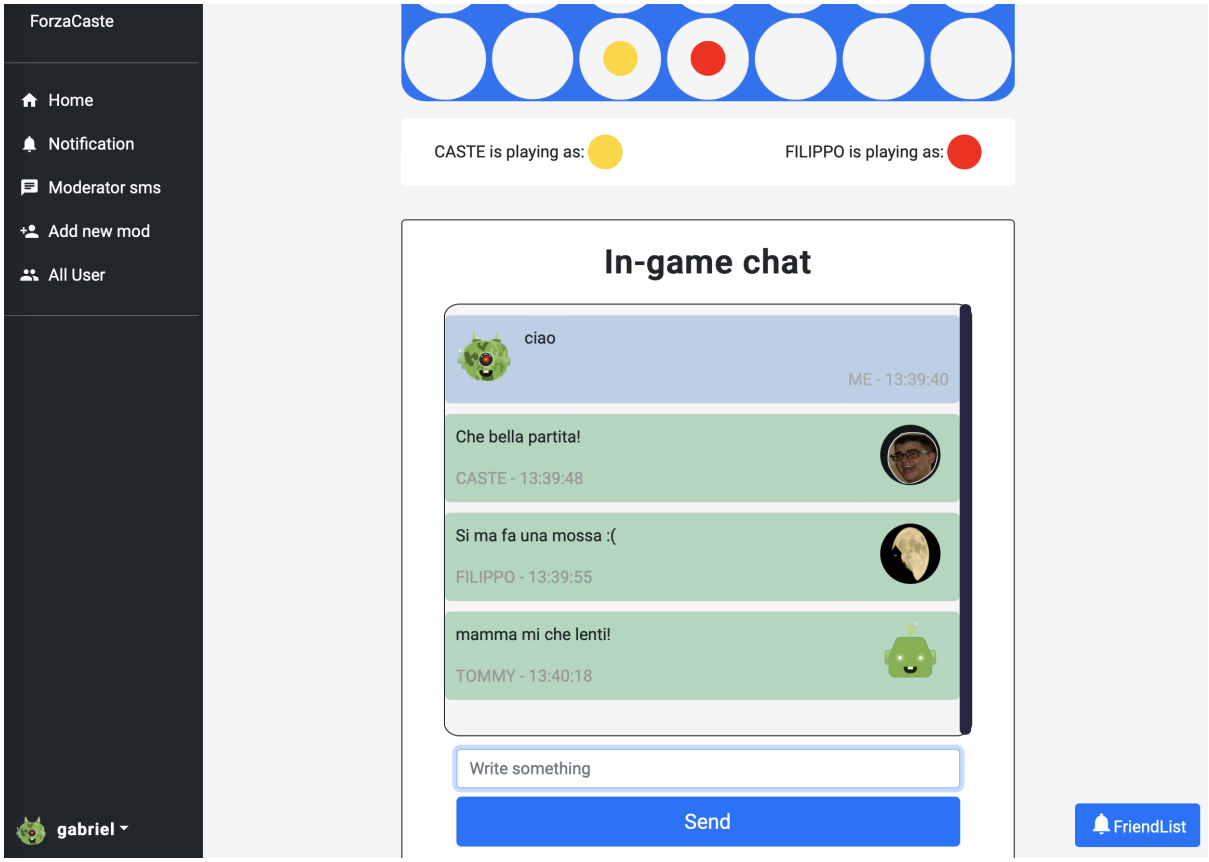
> When you want to watch a friend's game, click on the Watch a friend button on the homepage screen, this will open a dialogue that will load all of your friends' games in progress, if there are no games in progress, you will be informed, if there are, then you can decide to log in by clicking on the watch button from the list of friends who are playing, when the game is loaded you will be redirected to the game screen.

WATCH MODE GAMEBOARD



> When the game is loaded with all the moves made up to that moment, you will be able to view the playing field that will allow you to exit and view all the moves of the 2 players in real time, here you can also watch a game between a player and the cpu, so we can see who the 2 players are, whose turn it is, the moves

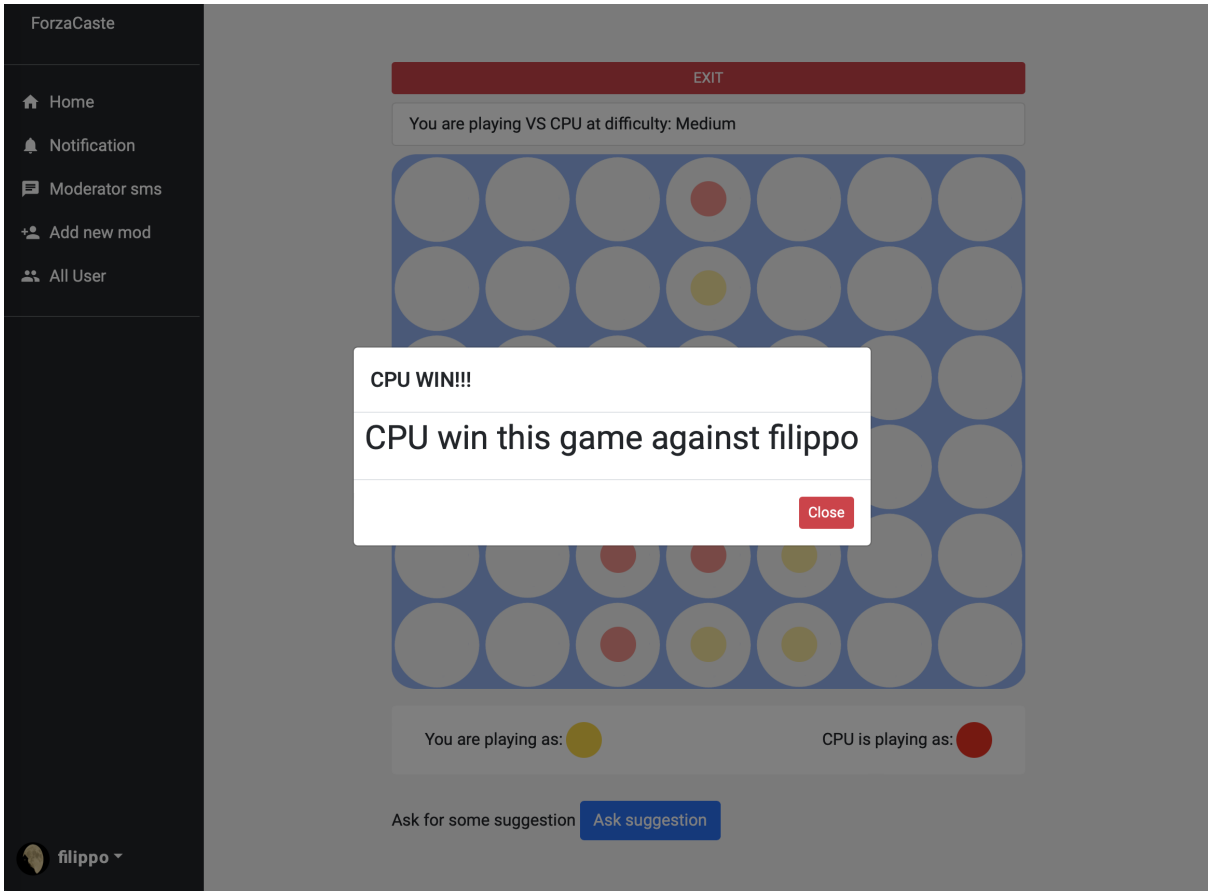
> made on the playing field, and the colours of the tokens of the respective players.



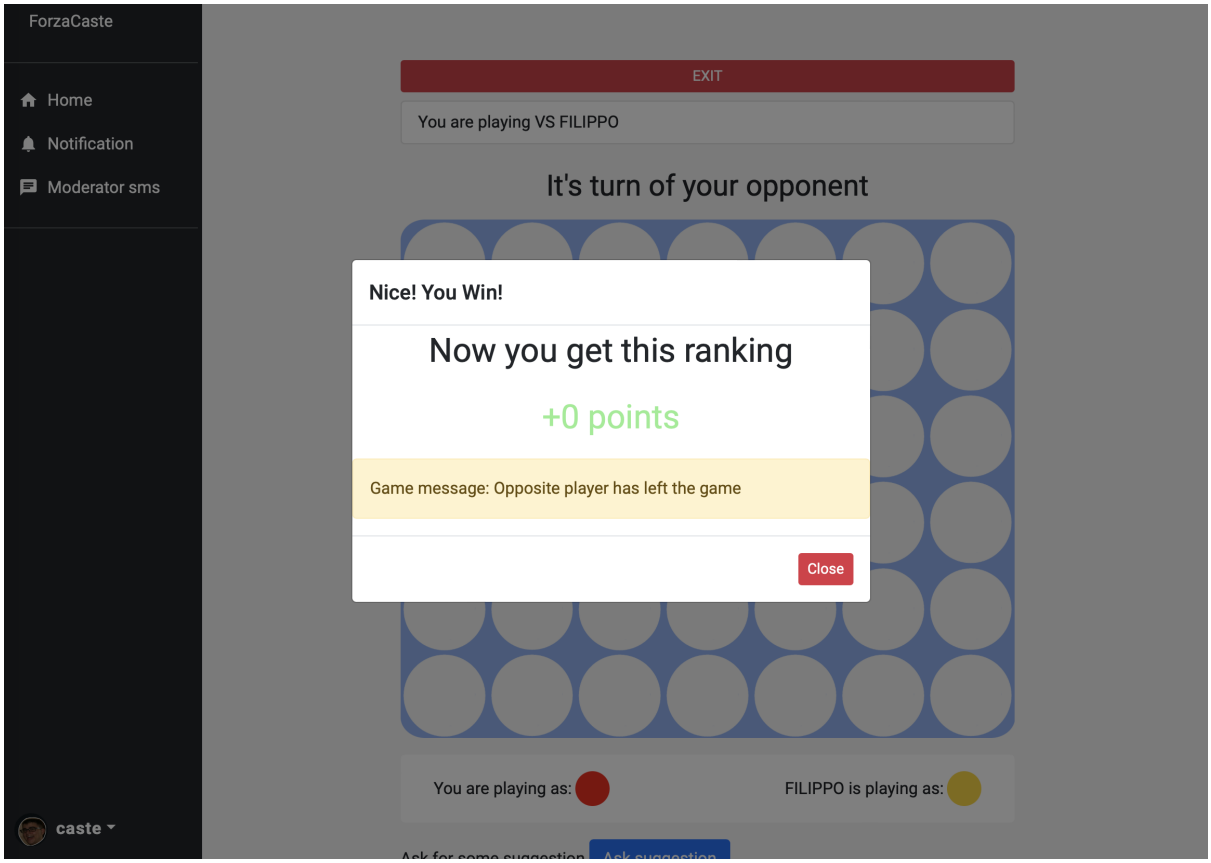
> If you want to comment on the game, you can send a message in the game chat, in which the messages of the 2 players who are playing the game and of all the observers who are viewing that game will appear.

Ending game

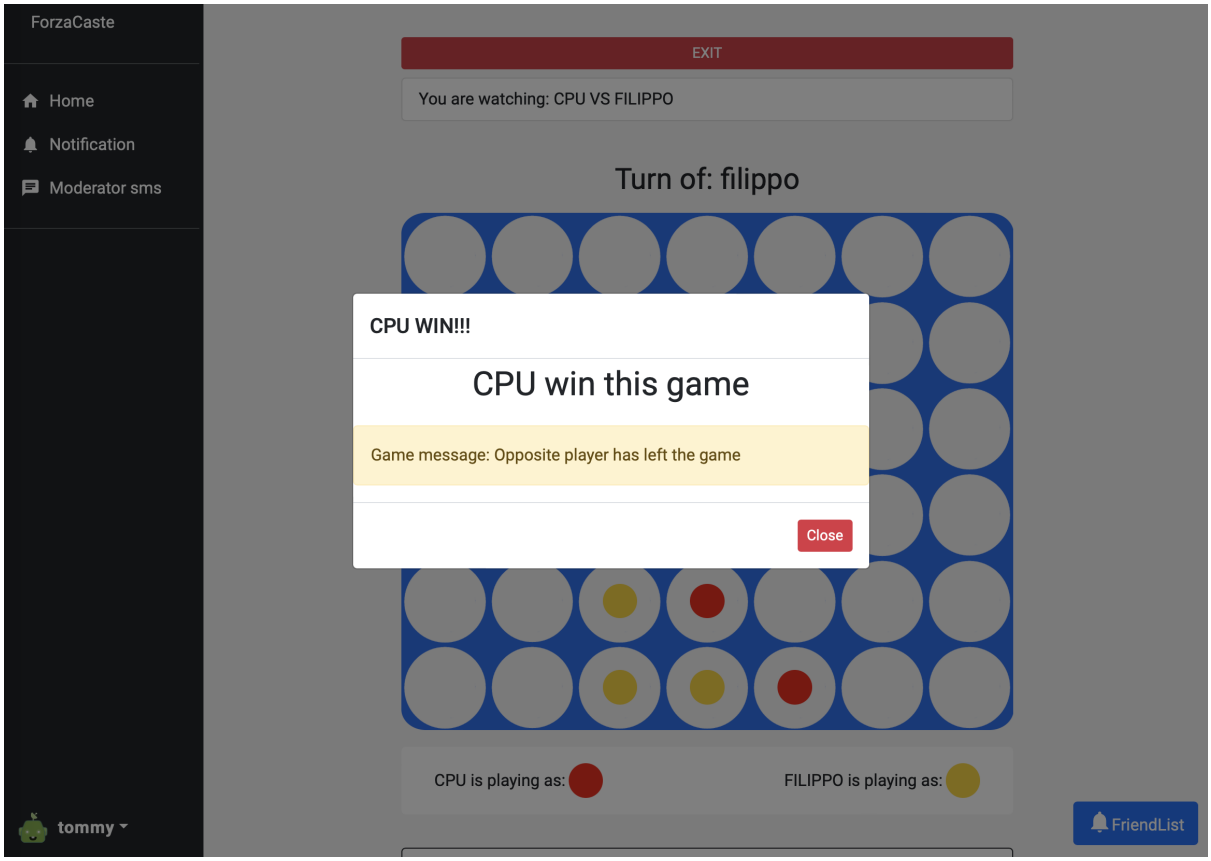
Quando una partita termina perché qualcuno vince, perde o pareggia verranno visualizzate queste schermate che permettono di capire come è finita la partita, per ogni modalità c'è una schermata diversa che però hanno la stessa funzione, non si potrà tornare alla gameboard, una volta visualizzato il messaggio e si ha intenzione di uscire dalla partita basterà cliccare exit e così si tornerà alla homepage



> Lose game vs cpu



> Winning game vs Player



> Losing game vs CPU in watch mode