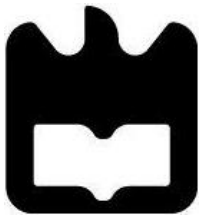
MathGames

Universidade de Aveiro

Departamento de Eletrónica, Telecomunicações e Informática

Projeto em Informática

Technical Report



Diogo Carvalho, Diogo Cunha, Pedro Amaral, Pedro Santos, Rafael Baptista, Ricardo Cruz

Advisors: Nuno Lau, Diogo Gomes

25 de junho de 2021

MathGames

Report of Projeto em Informática from Licenciatura em Engenharia Informática from Universidade de Aveiro written by Diogo Carvalho, Diogo Cunha, Pedro Amaral, Pedro Santos, Rafael Baptista, Ricardo Cruz under orientation of advisors Nuno Lau and Diogo Gomes.

Keywords

web game design, mathematical board games, 3D avatar, gamification, elo rating system, online tournaments

Abstract

This project was developed during the Projeto de Informática class of the degree course in computer engineering at the University of Aveiro.

In this project our team developed a desktop application, a web platform and a mobile application that allow users to play mathematical board games with a large variety of game modes. This concept emerged from a current lack of digital platforms where kids and teenagers are able to play these types of games.

The team is very glad about the final outcome since we responded to all the requirements made by our customer, and we were even able to add some additional features.

Table of Contents

Figure Index

Chapter 1. Introduction

At the beginning of the Projeto de Informática course, we were presented with a list of potential project proposals from which we should select the topic that we would cover during the semester. After analyzing all items on the list, we ended up selecting proposal number fifteen called “Jogos Matemáticos”.

In this project, we were challenged to develop a web and desktop portal and mobile app that allow users, mainly children, to play mathematical board games using their computers or mobile devices. These mathematical board games are games, usually played by more than one player, that involve strategy and logical reasoning with the goal of introducing players to basic mathematical concepts allowing children to exercise their cognitive abilities in an interactive way.

What motivated us to select this project, at a more professional level, was the anticipation of a probable contact with a vast set of new technologies in areas that are interesting to us, such as game development, mobile development, and artificial intelligence. On the other hand, if we have a look outside of a professional vision, the possibility to develop a platform where the main target is children and where we feel that our work is going to be used to fulfill someone’s needs motivates us to do our best in this project.

Besides the advantages of having a web and mobile platform where every child can access to play mathematical games, helping in the education of those children, the MathGames project is going to solve the difficulties that Fábrica Centro Ciência Viva de Aveiro is facing during this pandemic to play these games and host tournaments with their visitors, since physical travel to the factory is not always possible. In addition, our desktop app will allow users with no access to the internet to play games in the offline modes, which will be presented in the future.

It is also relevant to inform that this project was proposed through a partnership created between Universidade de Aveiro and the institution of Fábrica Centro Ciência Viva de Aveiro that played a fundamental role.

1.1. Objectives

Follows the list of objectives:

1. Games can be played by 2 players on the same device;
2. Games can be played by 2 players via the internet, online;
3. Association between players and skill levels in different games;
4. Users should be able to play against the computer;
5. AI Bots should have a configurable level
6. Possibility to create virtual tournaments;
7. A component that registers interaction between players and provides statistics regarding those interactions;

1.2. Report Structure

TO DO

Chapter 2. State of the Art

2.1. Digital Web Games

Digital web games are games played on the internet, usually through a bowser. These may include multiplayer or single player gameplay, and some include both options.

In order to better understand what had already been done, what worked and what did not in terms of functionalities, we researched some of the available services which had similar characteristics to those we are going to develop for our own app. We began by looking at apps that members of our team already knew and used, starting with some of the popular chess websites, like chess.com and lichess.org, because even though chess is not one of the games we are going to develop, all board games have some similar characteristics, therefore we could extract some knowledge from these popular services, we also analyzed other services recommended by our advisors like kaggle.com.

By analyzing those resources, we compiled a list of relevant features implemented and interesting methods used:

- The usage of uuids to represent users without accounts;
- A leaderboard page with all player's rankings separated by different game modes, therefore increasing competitiveness;
- Invitation of other players to a match through a link, so that users can easily invite someone with our without an account to play a match;
- A live chat functionality present during matches to increase interaction between players;
- During a match, highlighting the player's possible moves, as well as the last move played, to help beginner players not to lose track of the current state of the match.

2.2. Math Games

In order to better understand what and how currently recognized math games were played, we researched several resources available online and spoke with Professor Pedro Pombo, manager of Fábrica Centro Ciência Viva de Aveiro.

From our research we reached the conclusion that current math games[1], have not yet made the transition to the digital space, meaning they are solely played with a physical board or a paper with the board layout printed on it, which means that there is still a lot of room for improvement.

Every year, a Math Games tournament is hosted, where young students usually compete in person playing one or more of the available games. During the covid-19 pandemic, organizing these tournaments is no longer a possibility, therefore creating the need for a platform where players can easily compete against each other.

2.3. Selected Technologies

We also analyzed what current technologies are most popular for developing web applications/ web games. Most resources and websites only listed HTML, CSS, and JS, but we were looking for a framework which could simplify some of the code we needed to write. With the suggestion from our advisors, we looked at Phaser 3[2], a popular JS framework used in web game development. This framework turned out to be perfect for us since it had a lot of documentation, lots of game examples/ tutorials and it simplified the development of our games and their functionalities.

[1] A list with some of these games can be viewed at https://www.ua.pt/pt/fabrica/page/9833
[2] Official website: https://phaser.io/phaser3

Chapter 3. Conceptual Modeling

3.1. Actors

With the development of our system, we are mainly targeting kids and teenagers who will make use out of the several games we are going to provide. In both of our applications, desktop, web, and mobile, we will be creating a simple user interface since our users do not have much technological knowledge/experience.

To avoid the necessity of sharing personal information and the possibility of our users not having a personal email, we will give the opportunity to play our games to everyone, even without the need of creating an account. From this requirement, we can extract one of our actors, which is a user without an account. This type of user characterizes everyone that joins our platforms and plays matches (from all available games) with other users on the platform. They will also be able to access our offline tools, which are playing vs AI and playing with a friend on the same device. Their progress, however, will not be stored.

We hope to capture these users' attention by providing a fun, competitive, and engaging environment, which will hopefully lead to the creation of an account, generating our second actor, a user with an account. These users will have access to all of the functionalities stated above, while also having their progress stored so they do not risk losing it when transferring between different devices. The match history will also be available, they will have a rank (increased/ decreased by winning/ losing games), an account level (increased by playing games), and will have access to limited content unlocked by leveling up their account, for example, the avatar customization will have locked items that are only disabled once the user reaches a specific level. They will also have the ability to participate in tournaments.

Our third actor is a user with tournament privileges, these users have the additional functionality of creating and managing their tournaments, these privileges are granted by a system admin.

The actors stated above will have access to the offline functionalities through the desktop app, being the online functionalities available through all platforms, web app, desktop app, and mobile app.

We are also targeting the FCCV staff (system admins), by providing a way to organize tournaments through the internet without the need for the physical presence of participants, and, even if they are hosted at the FCCV, competitors could be provided with computers to play out the games. Organizers will also have the ability to manage users by banning those who are misbehaving or disrespecting rules. In addition, these users will also have a section with statistics regarding the use of our platform. These users will be mostly using our web and desktop app.

3.2. Personas

In this section we are going to present personas based on the actors defined in the section above.

Persona	**Motivation**
Rui is an 8-year-old shy kid who loves board games. He knows how to play chess and checkers, which he mostly played with his friends at school before the pandemic started.	Rui is looking for a website where he can play fun and interesting board games with other people. He does not have an email so he cannot use websites that require an account. Since he is shy, he prefers to learn how to play the games before challenging other people, so he also wants to be able to train alone.
Ricardo is a very competitive teenager, he excels in most of his classes and always works hard to accomplish his goals. Lately, he has been feeling like there is no competition for him at school, so he wants a different way to compete with others.	Ricardo wants an app where he can challenge and be challenged, he wants to be able to use it anywhere so that he can always feel the thrill of competition. He wants to be able to visually see his improvement while competing against similarly matched people.
Ana is a middle school teacher who wishes to organize a fun activity with her students,	Ana wants to be able to create and manage multiple board game tournaments

she is always looking for innovative ways of teaching and entertaining the minds of her students, and she believes board games are a great way of stimulating creativity and problem-solving skills.	intuitively. She also wants to invite as many or as few students as she would like.
Artur is one the members of Fábrica Centro Ciência Viva de Aveiro. He was unable to host the math games tournament this year due to regulations that took place due to the pandemic.	Artur wants to have control over an application where users can play the recognized math games, he also wishes to be able to organize official digital tournaments of those games. He also wants access to important statistics to view which games are the most popular.

3.3. Use Cases

In this section, we are going to present our use case diagrams of the whole system, where we can have a look into how our actors will interact with the system, we'll also describe each use case and its priority. We have divided the use cases in two sets of actors: Regular users (Figure 1) and Administrator users (Figure 2).

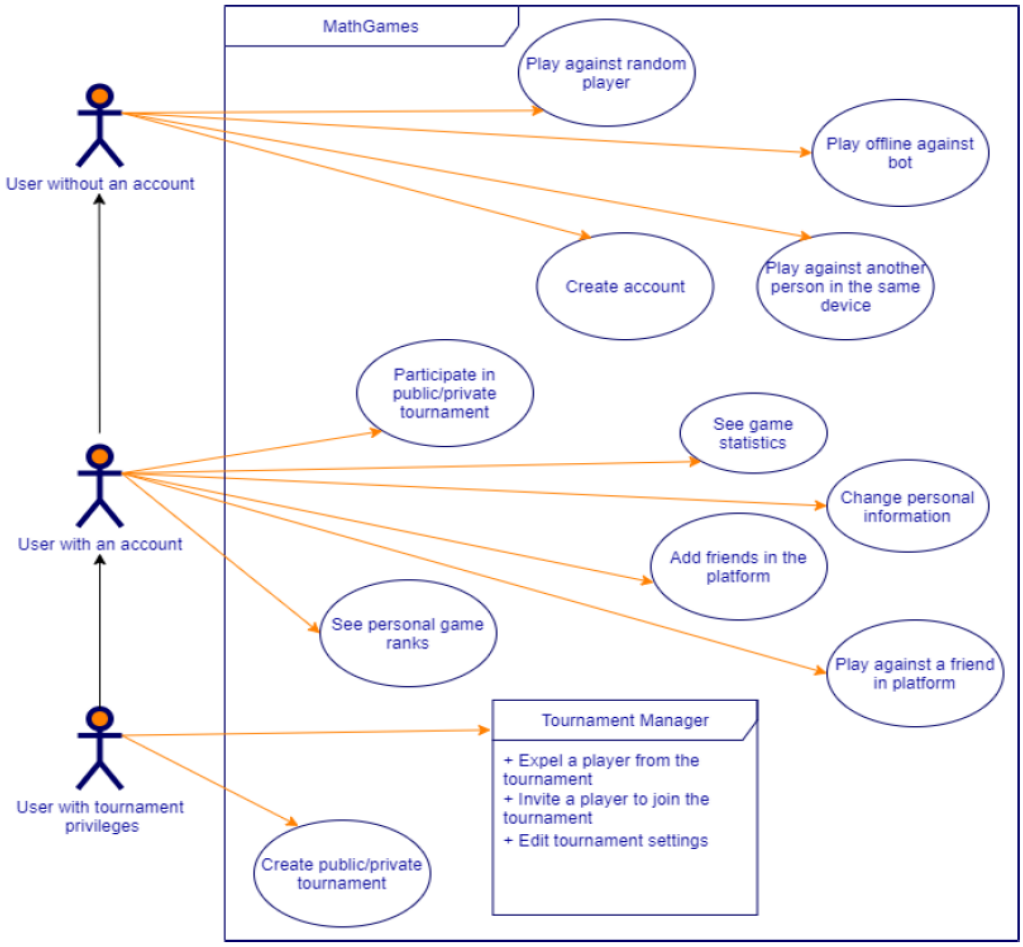


Figure 1 - Regular User Use Cases

- **Play Against Random Player**

Rui intends to play Rastros against a random person. To perform this he must first access our main page, he must then press a button present on that page or, through the side menu, access the game's option. After reaching the games page he will be able to choose what game he wants to play, which will then show the multiple game modes for that specific game, after choosing the mode for online versus another player, the matchmaking process will begin and when an opponent is available the game will begin.

Priority: High

- **Play Offline Against AI**

Rui wants to improve his skills in Rastros by playing against the computer. He must first access our main page, then he must press a button present on that page or, through the side menu, access the game's option. He can then choose the game he wants to play and, on the page of the game he chose, click on the respective mode and start playing.

Priority: High

- **Play Against Someone on the Same Device**

Rui wants to play Gatos e Cães against his brother on a computer without access to the internet. To do this, he must first access our desktop app, then press a button present on the home page or, through the side menu, access the game's option. He can then choose the game, which will show the different game modes available, after selecting the option for playing against a player on the same device, the game will start.

Priority: High

- **Play with a Friend on Different Devices**

Rui wants to challenge a friend in a game of Rastros, so he must first access our main page, he must then press a button present on that page or, through the side menu, access the game's option. Then choose the specific game he wants to play and finally, on the game page, he must choose the option to invite a friend. A link will then be generated, which can be sent to his friend. After accessing the link, the game will begin.

Priority: High

- **Create an Account**

Rui wants to create an account on the platform to be able to keep track of his evolution and get a classification regarding his skill in each game. If the users want to create an account on the platform, they must first access the login page, then click on the create account button, which will open a new page with the registration form, after

filling it, they will be able to submit it, and if the verification succeeds, the account will be created.

Priority: High

- **Add a Friend**

Ricardo wants to add his real life friend on the platform friend's list to invite him to play later on. To add a friend, the user must access the friend list menu, introduce the username of the account that he wants to add, and confirm the operation. In the alternative, the user can access the classification page, search for the friend username and click on the icon which represents adding a new friend. The friend who received the request notification must access his list of notifications so he can accept it.

Priority: High

- **Challenge a Friend**

Ricardo wishes to challenge one of his friends directly through the app, to do that he can access his friend list from the navbar, he can then press the challenge button next to the friend he wants to challenge. The friend who received the game invite notification must access his list of notifications so he can accept it.

Priority: Medium

- **Participate in a Tournament**

Ricardo wants to compete in a large tournament created by his school's teacher. Users wishing to participate in public or private tournaments must access the tournament list page through the side menu. Then choose the tournament in which they wish to participate and choose to sign up.

Priority: High

- **View Game Statistics**

Ricardo wants to view his progress over the last few matches. To accomplish that, users can access a page to view their match history and statistics of previously played games, this page can be accessed through the user's profile, which can be accessed through the navbar.

Priority: Medium

- **Change Avatar**

Ricardo wants to change information regarding his avatar (hat, jeans, …). He can accomplish that by accessing the user's profile page through the navbar. Navigate to the avatar customization section that appears in the side menu of the profile page and change the avatar components. To conclude, he must confirm the changes performed.

Priority: High

- **View Personal Rank of a Specific Game**

Ricardo wants to know his rank in a specific game, to find that out, a user should access the Games page through the side menu and then choose the game he is looking for which will redirect him to a page with the game’s information and the user’s rank.

Priority: Low

- **Create a Tournament**

Ana wants to create a tournament to give her students a way of having fun while still learning new skills. After requesting permission to create tournaments, she can access the tournaments page on the side menu, where she will have the option to create a new tournament, by filling the various settings that are required like max number of students and whether the tournament is public or private.

Priority: High

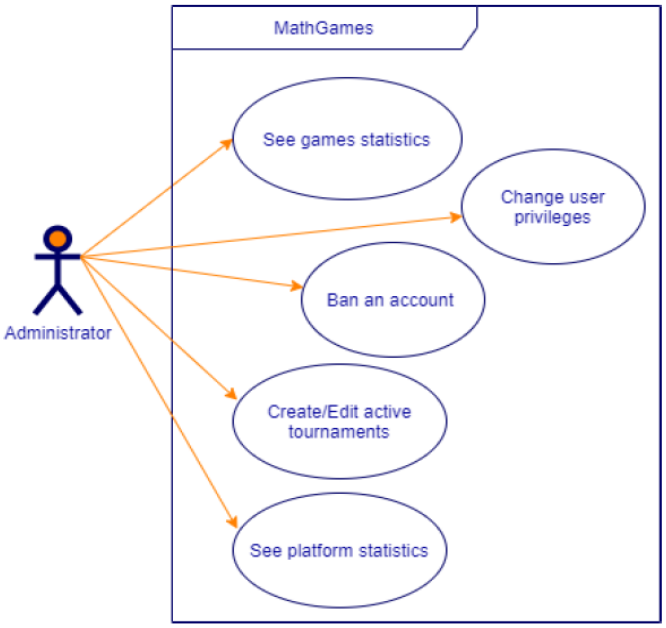


Figure 2 - Admin User Use Cases

- **Invite/ Remove a user from a tournament**

After creating it, Ana wants to remove some players from her tournament since they do not belong to her class. She can accomplish that by choosing the tournaments option on the side menu, she can then select the tournament from the list of personal tournaments. From inside the tournament page, she will have an option to remove current participants. Regarding the invites, Ana can create the tournament with a private password, meaning that if she only shares the password with the students, only the students will join.

Priority: Medium

- **Change Tournament Settings**

In case Ana wants to alter any information regarding the tournament's settings (description, max number of players, …) she can do so by choosing the tournaments option on the side menu, she can then select the tournament from the list of personal tournaments. From inside the tournament page, she can choose to edit the various settings.

Priority: Medium

- **View Statistics About the Platform**

Artur wishes to see the statistics of all the games, to find out what the most played games are. Administrators who want to see statistics for all games should access the statistics page through the side menu.

Priority: Medium

- **Ban a User**

Artur intends to ban an account that has been reported by multiple users of using cheats. Administrators can ban users by accessing the page with the list of all players through the side menu. Then search for the account they want to ban and click on the ban icon.

Priority: High

3.4. Functional Requirements

3.4.1. Playing Games

- Play games with a friend on the same device;
- Play games with a friend in distinct devices;
- Play games against an AI Bot;
- Allow changing AI Bot's difficulty;
- Play games against random people (with or without identical rank);
- Validate each move done in online game modes to prevent invalid moves.

3.4.2. Communication

- Allow communication via chat with pre-defined sentences;
- Report users;
- Add friends;
- Remove friends;
- Invite friends to play.
- Send notifications on fundamental operations (friend request, request deny, game request, tournament round initiation, …)

3.4.3. Customization

- Change personal account settings;
- Change/Customize avatar.

3.4.4. Tournaments

- Create public or private tournaments;
- Administrate tournaments (Change configurations);
- Invite players to a tournament;
- Remove players from the tournament;

- Join tournaments;
- Create and display tournament brackets.

3.4.5. Authentication/Authorization

- Register a new account;
- Log-in using a created account.
- Verify account privileges to perform actions on the server

3.4.6. Admin Privileges

- Upgrade account privileges;
- Ban users;
- Watch full statistics on the admin side regarding all games/users.

3.4.7. User Records

- Access match history;
- View ranking of a specific game;
- View account level.

3.5. Non-Functional Requirements

3.5.1. Performance

- Each page must load within 3 seconds.

3.5.2. Usability

- The application must be open access for different devices;

- The software should be portable. Moving from one OS to another should not create any problem.

3.5.3. Availability

- Application must be available all the time, except when in maintenance service, keeping maintenance time minimal.

3.5.4. Security and Data Integrity

- User's information should be confidential;
- Passwords shall never be viewable at the point of entry or at any other time.

3.5.5. Regulatory

- Application must follow GDPR guidelines.

3.5.6. Capacity

- The app should be able to handle a high number of simultaneous users;
- Should also be able to store the data related to those users.

3.6. System Architecture

3.6.1. Architecture Diagram

Our architecture diagram aims to show the different layers of components in our project and how they are linked. Firstly, we have a database in MySQL since it is a relational database, the type of database that is recommended for most projects, and it fits our needs to persistently save data. The database will be connected to a NodeJS server that contains an API containing the necessary endpoints for establishing all needed communications between the server and the frontend, the business logic to support these endpoints, and the online game engines that validate each move made by a player in an online game. On the client-side, we have a desktop application implemented with Electron[3], React[4], and Phaser. This application will contain the React website and a local NodeJS server containing the local game engines needed to support offline games, the AI agents, and the services to communicate with the API. Lastly, we also have a mobile application implemented in React Native[5] and Expo[6] similar to the Web Application.

[3] Framework used to create native applications using web technologies.
[4] JavaScript framework for web development.
[5] JavaScript framework for mobile development.
[6] Set of React Native tools used to significantly lessen the need for native code.

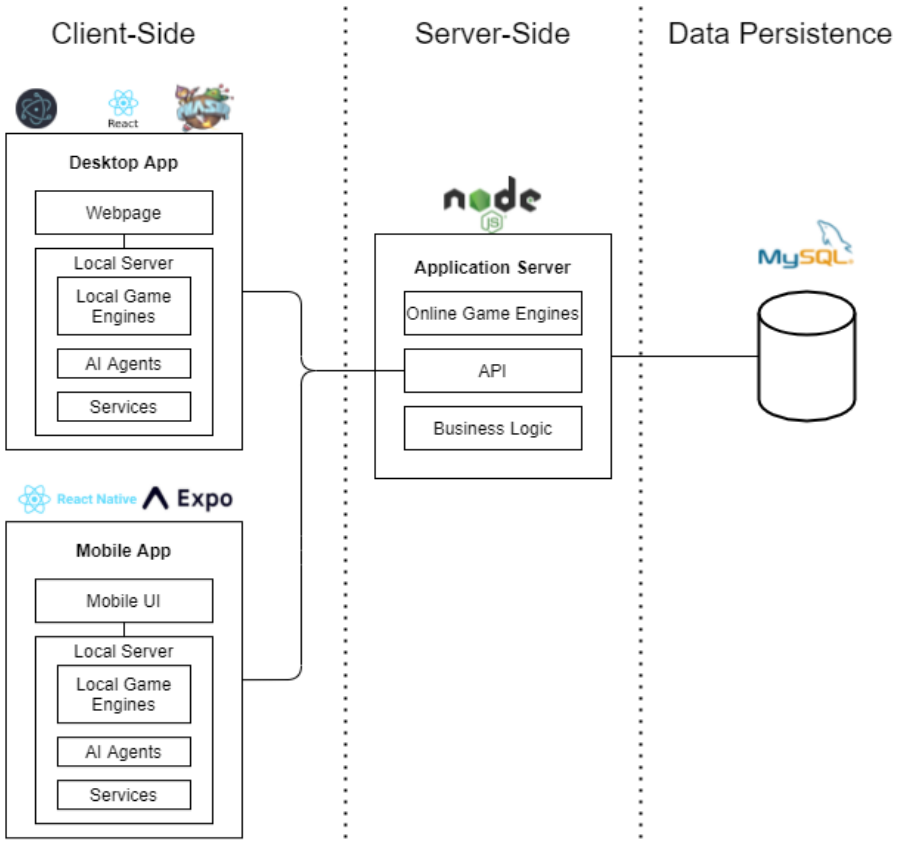


Figure 3 - Architecture Diagram

3.6.2. Domain Model

Our domain diagram (Figure 4) aims to represent the data our system will contain. Therefore, it includes:

- A user who must have an id, a username, an email, a password, an account level, an account type, a code to represent the color, hat, shirt, trousers, and accessory of the avatar, information if the account is banned or not, a rank for each game and N friends;
- A game that must have an id, a name, a description, and a recommended age;
- A game match that must include an id, a game, two users (player1 and player2), a winning player, and a game type;
- A tournament which must have an id, a creator, a name, a description, a game, the max number of participants, information on whether it is private or public, a password, N participating users (with information if it has already

been eliminated), a current round number, the current status of the tournament and a winning user;

- A tournament match which is a game match and must also have a tournament to which it belongs, two users (player1 and player2) that although end up being duplicated significantly reduce the time and complexity of the needed queries for the tournament, the number of the round, information if it is a match that will not be played since the tournament is not full, the ids of the game matches that were played lastly by the two players of the match and the id of the game match that will be played by the winner of the tournament match;
- A banned user who must have the reason why that user was banned;
- A report which must have an id, the id of the sender, the id of the receiver, and the reason it exists;
- A notification that must have an id, the id of the sender, the id of the receiver, its type, and a text;
- An avatar item that must have a name, a level in which it is unlocked, and a category;

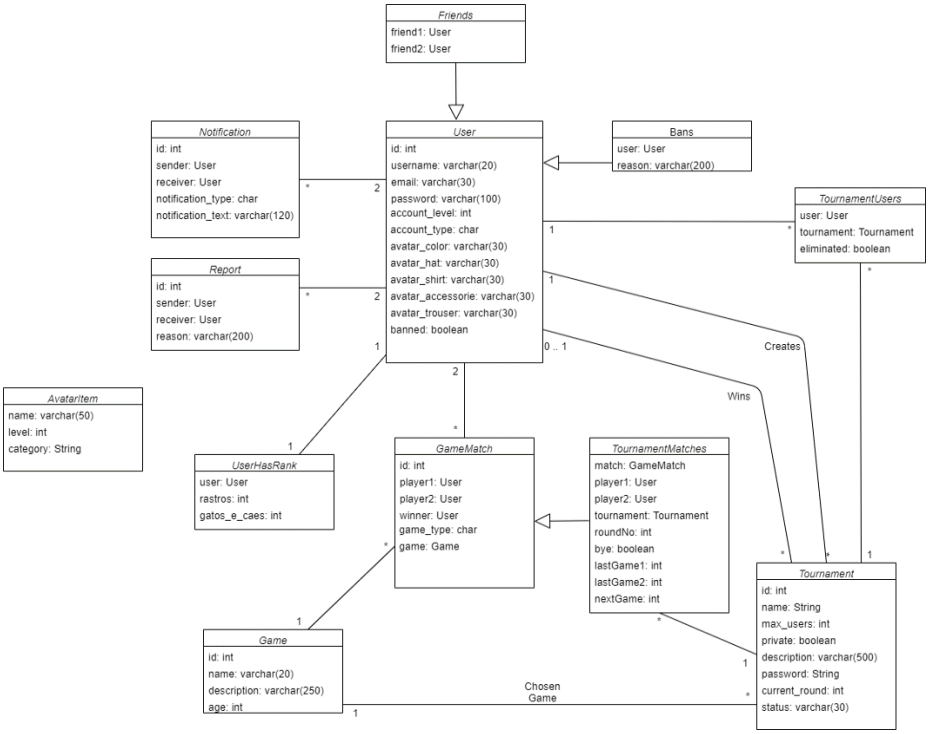


Figure 4 - Domain Model

3.6.3. Deployment Diagram

Our project will contain distinct components, which can all run on different machines. Our deployment diagram (Figure 5) seeks to show how the components are connected and deployed. Essentially, we will have a MySQL relational database (deployed in a MySQL8 Docker container), which will be connected through NodeJS's MySQL Driver to our Application Server (deployed in a node Docker container). Furthermore, we will have a mobile application and a desktop application in Electron (which internally contains a web server, a client browser, and a local database). These applications will be connected to the Application Server API through HTTP.

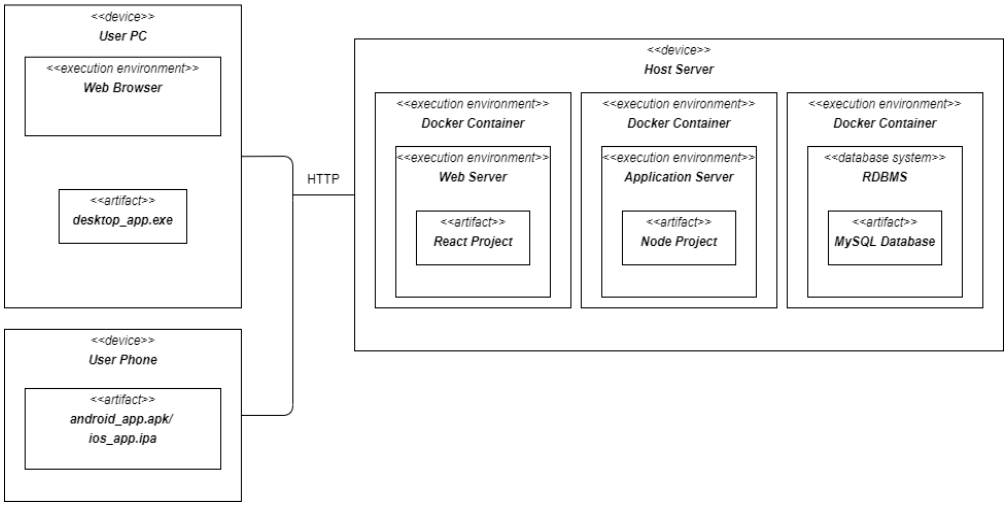


Figure 5 - Deployment Diagram

Chapter 4. Implementation

4.1. Frontend

4.1.1. Web Application

Following the idea of this project, it was a requirement to have an interface on which the user could play games. Upon this, we developed a free web application that allows the users to play without an account, create an account and grind ranks and gain rewards and even participate in tournaments.

To develop the interface of the application, like we previously stated, we decided to use React. React is an open source JavaScript library, based in components and states that aim to create interfaces. Along with React, HTML5 and CSS3 were also used to make the website format and style.

To reach the functionalities we desired, we then split the structure of our Web Application in some key aspects:

- Pages: Components that are used only one time, to display an HTML page with CSS styling, functions and states associated.
- Components: Bits of reusable code that can be used in multiple pages.
- Data: Files that contain static data, that do not contain privacy risks, that are used in components and pages, preventing code duplication.
- Services: Entities that are in charge of establishing connection with API. They handle the data received and deal with possible errors.
- Store: Redux store, that helps with the management of states throughout the application.

In this section we will now get a look at each page that constitutes our web application.

Home Page

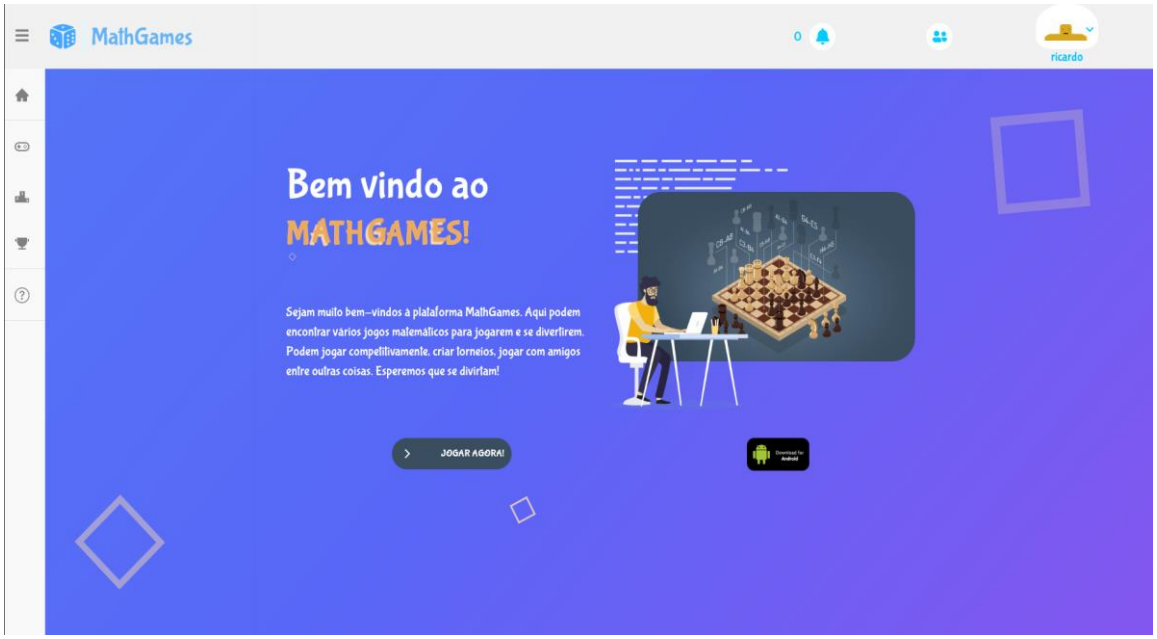


Figure 6 - Web Home Page

In the home page (Figure 6), we included a brief description highlighting some of the principal features of our application. It encourages users to try our application, and play some games.

Login

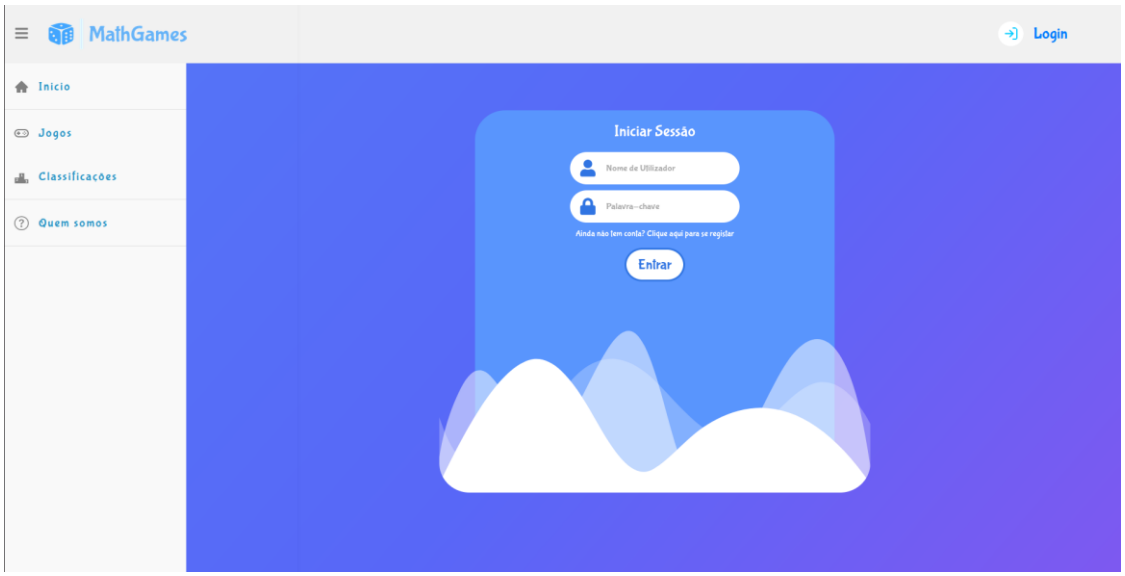


Figure 7 - Web Sign In/ Sign Up Page

In the register page we only ask users for a username (that needs to be unique), an email (also unique) and a password. Only the minimum information since the

website is directed to teenagers and kids. For the login page (Figure 7) the username and password are the necessary credentials, so that the user can log into his account.

Menu

In the navbar, if the user is logged in, he can logout, access his profile, see notifications, and invite friends to play. If he is not logged in, he does not have access to these functionalities.

In the side menu, we have different functionalities whether the user is logged or not, and different functionalities if he is part of staff or admin. The functionalities are similar between logged and not logged accounts, the major difference being the tournaments section being only available to users that are logged. In terms of the admin displayed sections, they change drastically, being displayed the statistics and user's management section.

Games

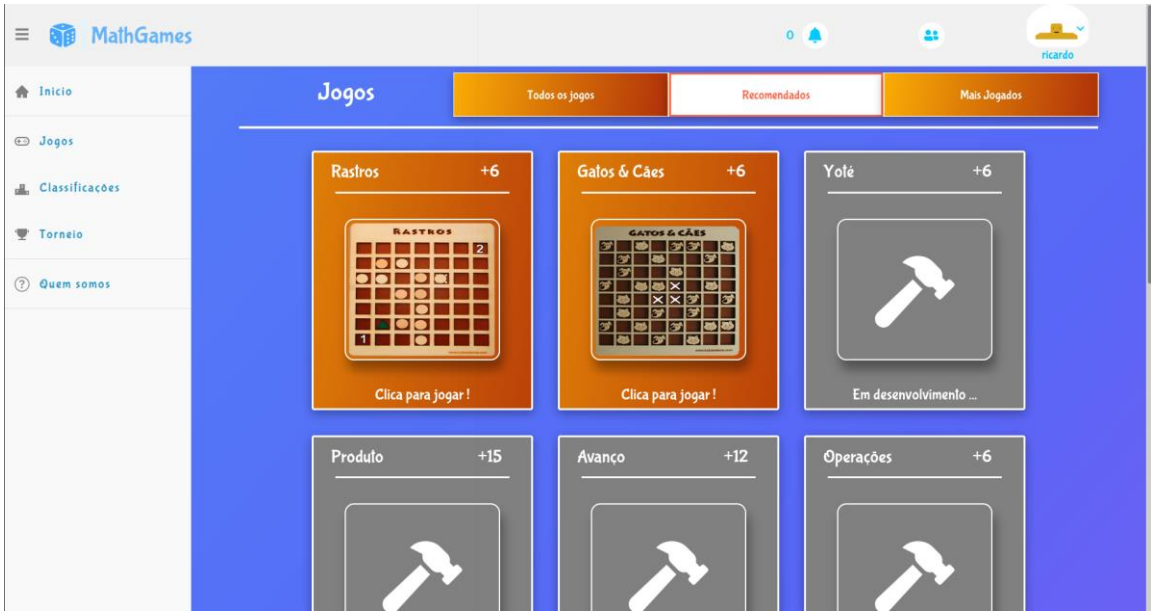


Figure 8 - Web Choose Game Page

In the choose game page (Figure 8), users have a list of games that are available and unavailable. The user can see the minimum recommended age to play the game, the user can also have a brief description of the game if he puts the mouse over the corresponding card, and by clicking on the card of the game that he wants to play, he goes to another page where he can choose the game mode. The games that

are still being developed, appear properly identified, teasing the user to wait for new games.

Besides the games list, there are three filters at the top of the page. “All Games” will display the games ordered by the first games that were implemented, “Recommended” and “Most Played” as the name says the games will be ordered by the most played to the least played game.

Figure 9 - Web Choose Game Mode Page

When the user chooses the game he wants to play, he now has two sections on the screen (Figure 9). The left section provides a brief description of the game, enumerating the rules when clicking on the question mark icon, providing the difficulty levels and the target age.

In the right section, if logged in, the user can see his current rank in that particular game, and finally he can choose to play either offline against a friend on the same computer, offline against an AI agent, online against another random player or online against a friend (through an invite link). If the user is not logged in, he has the same game modes, however when playing competitively he is not going to win rank points and when playing the other modes, he won't be rewarded with experience.

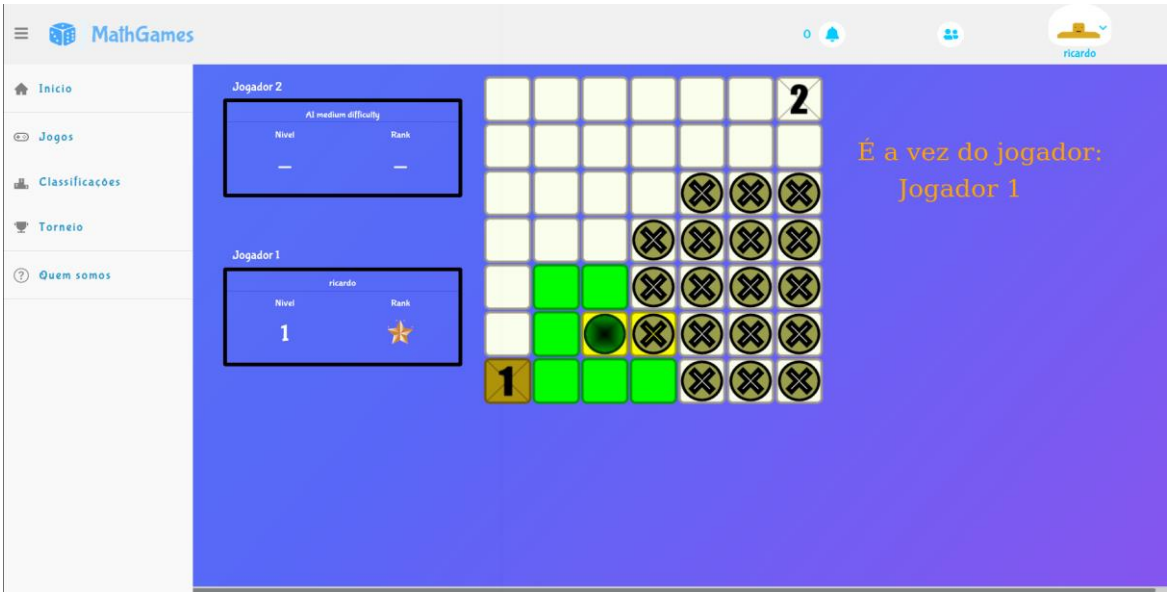


Figure 10 - Web Game Page

When the user clicks on play, he is now presented with a board (Figure 10). In this case, playing the game “Rastros”. Each game has a specific board.

Apart from the board, which is the main thing in the screen, it also displays user’s information. In this case, the mode chosen was against the computer, but on the other game modes the display is very similar. He can see the other player's rank and level, and it is also indicated whose turn it is to play.

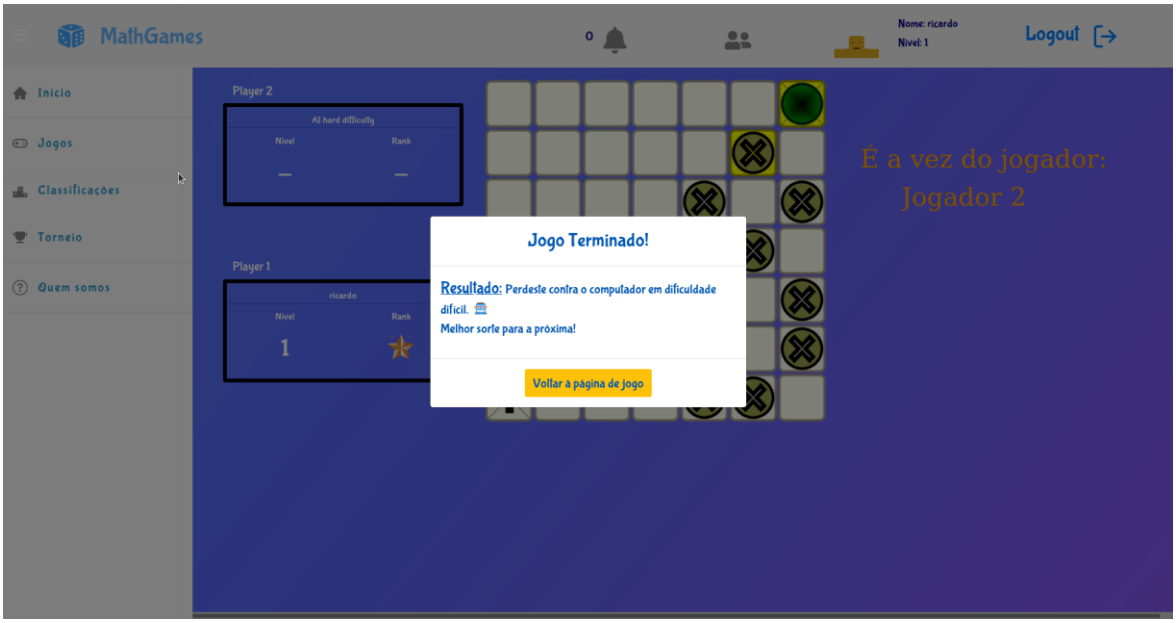


Figure 11 - Web Game Over Modal Against AI

After the game is finished, it a notification pops up (Figure 11) saying whether the user won or lost.

Classifications

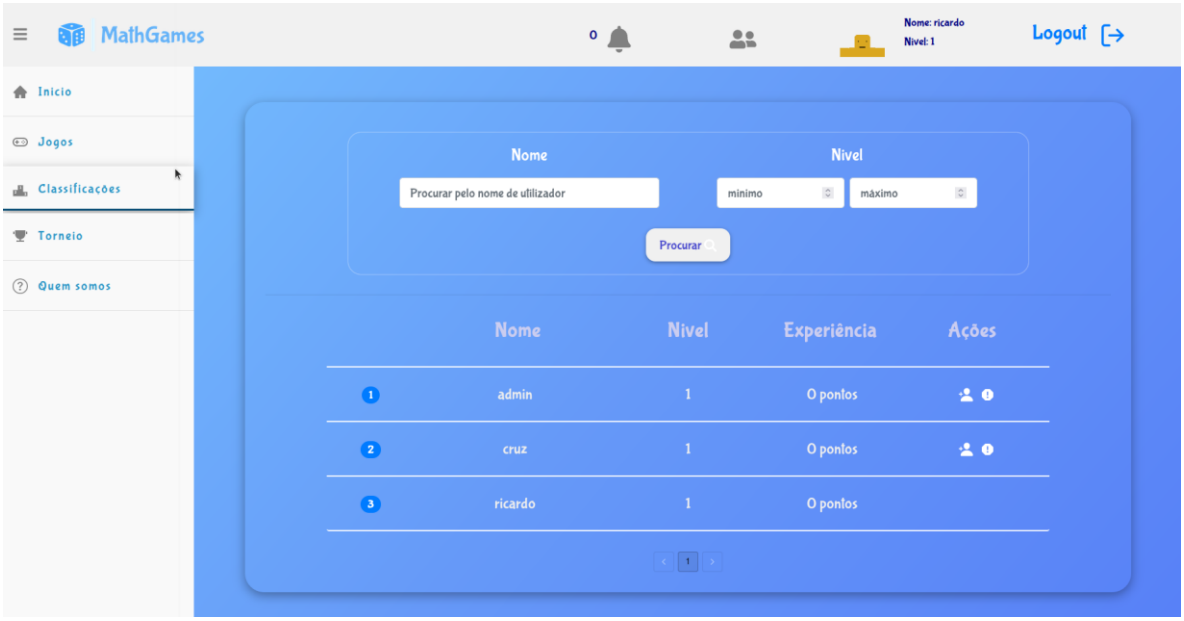


Figure 12 - Web Rankings Page

In the rankings page (Figure 12), the user can search for top tier players, sorted by the highest level. He can also search for players, filtering by name and level. When he finds what he was searching for, he can add them as friends, or even report them, indicating the reason so it can be analyzed.

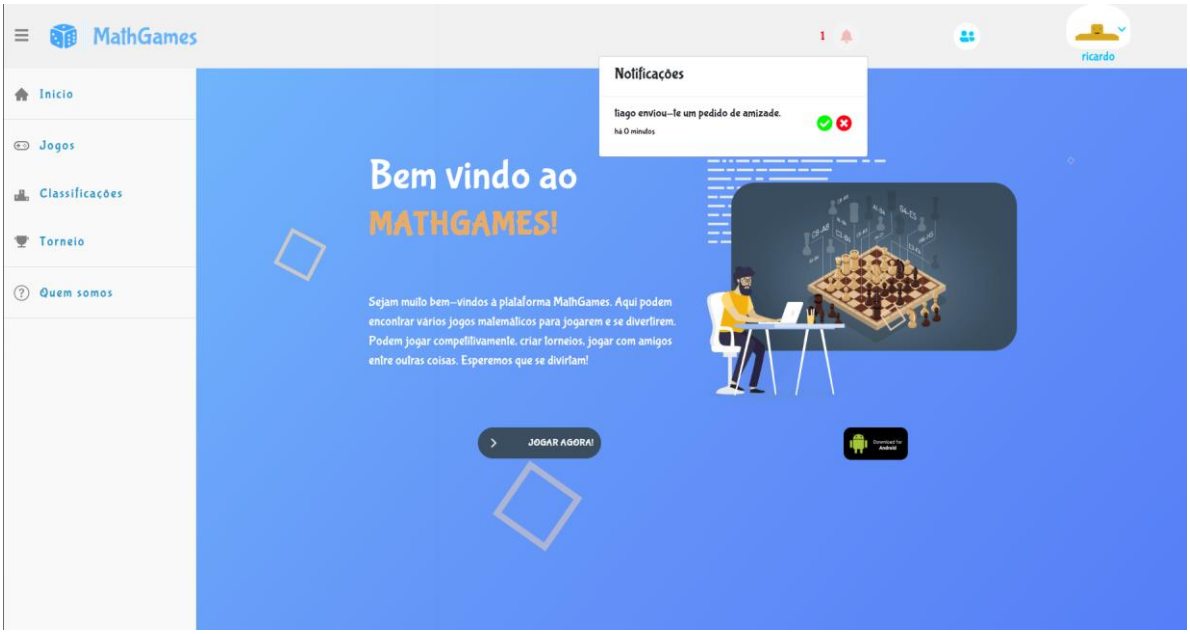


Figure 13 - Web Friend Request Notification

When adding a player, the other player receives a notification that blinks to catch user attention (Figure 13). The user can choose to accept or decline the request, and the other player will be notified as well.

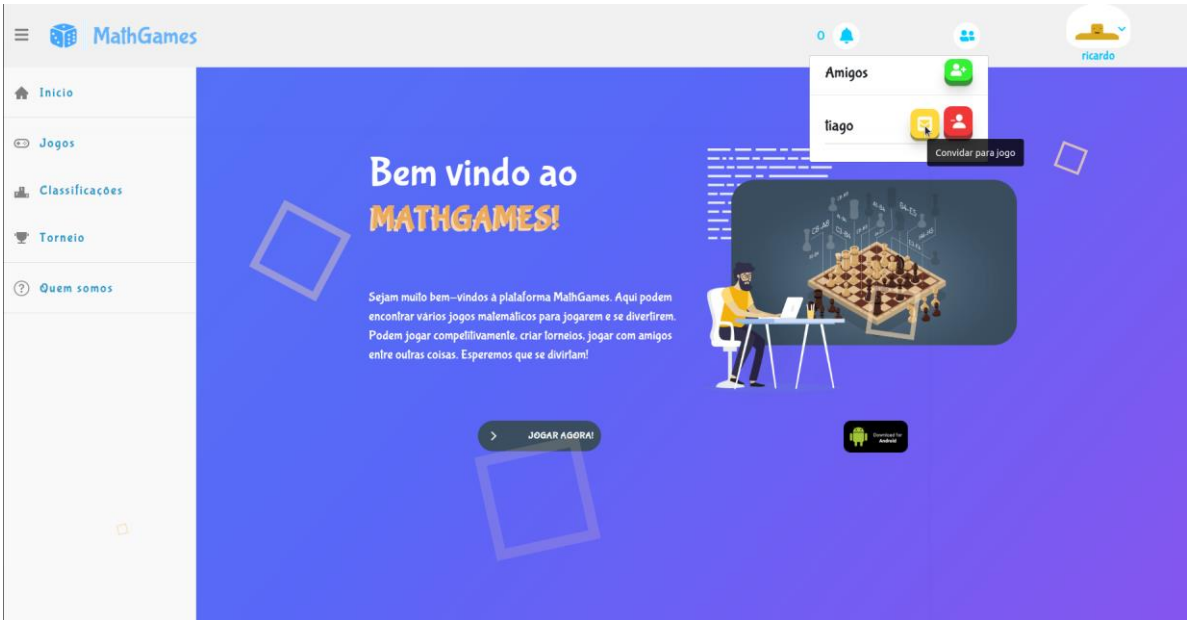


Figure 14 - Web Friends List

If he accepts the invitation, he can see that he now has “tiago” as a friend. He can invite him to play or remove him as a friend.

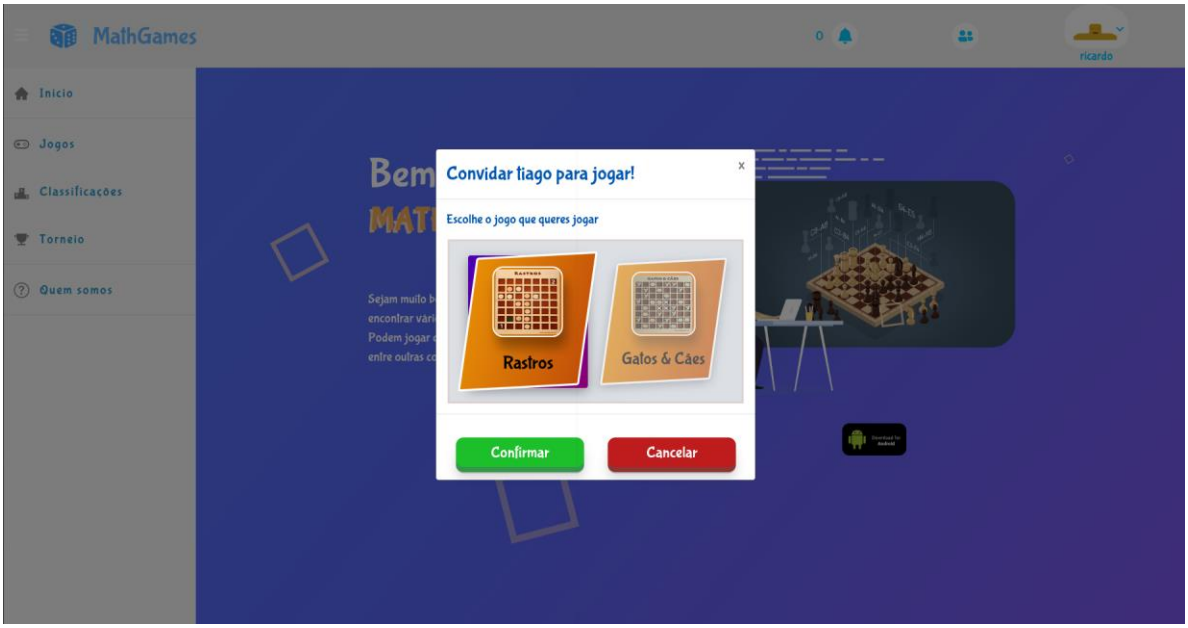


Figure 15 - Web Invite a Friend to a Match

If he wants to invite his friend to play, he has to choose the game (Figure 15), and an invitation will be sent. If the other user accepts, the game will start.

Profile

In the profile area (Figure 16) there is a sidebar where the user can choose the different options.

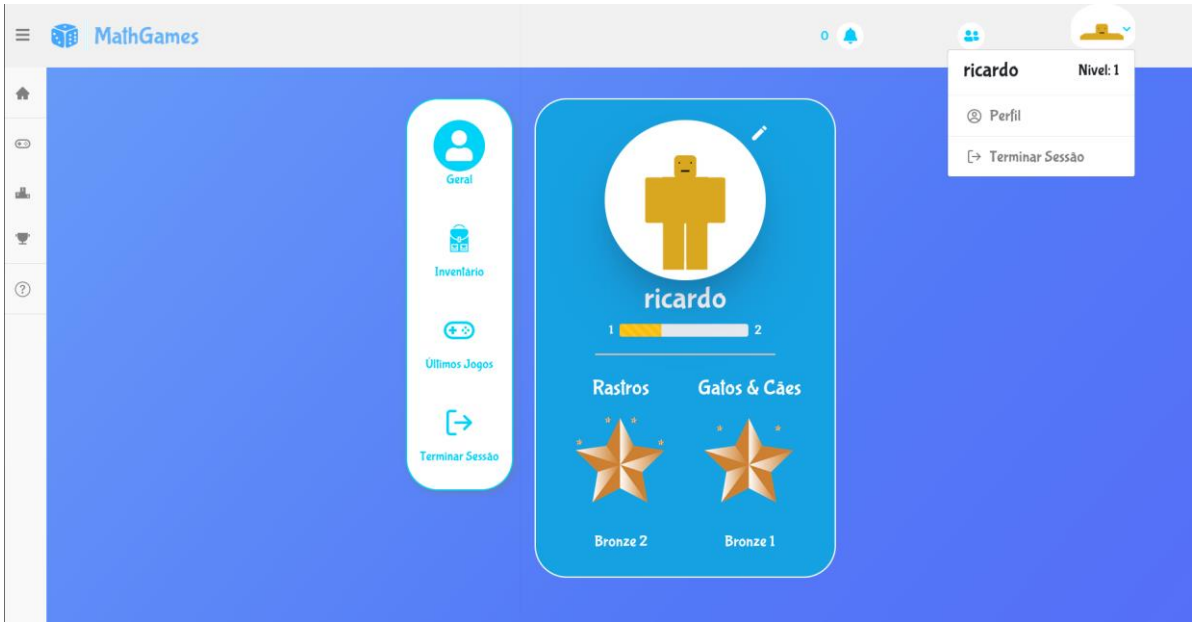


Figure 16 - Web Profile Page

Starting with the main one, the user can see his avatar, level and check his current rank in different games.

On the “Invetário” section (Figure 17), the user can check the items he has available and the items he can unlock. He can edit his avatar by switching in different tabs and choosing between the different items he has unlocked.

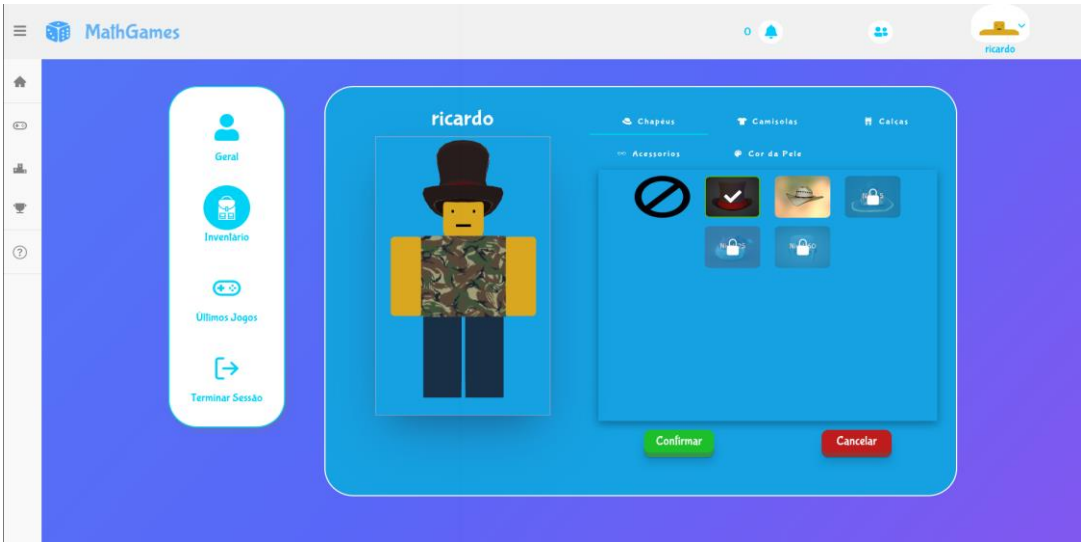


Figure 17 - Web Inventory Page

Last, in profile the user can also check the results of his last games (Figure 18).

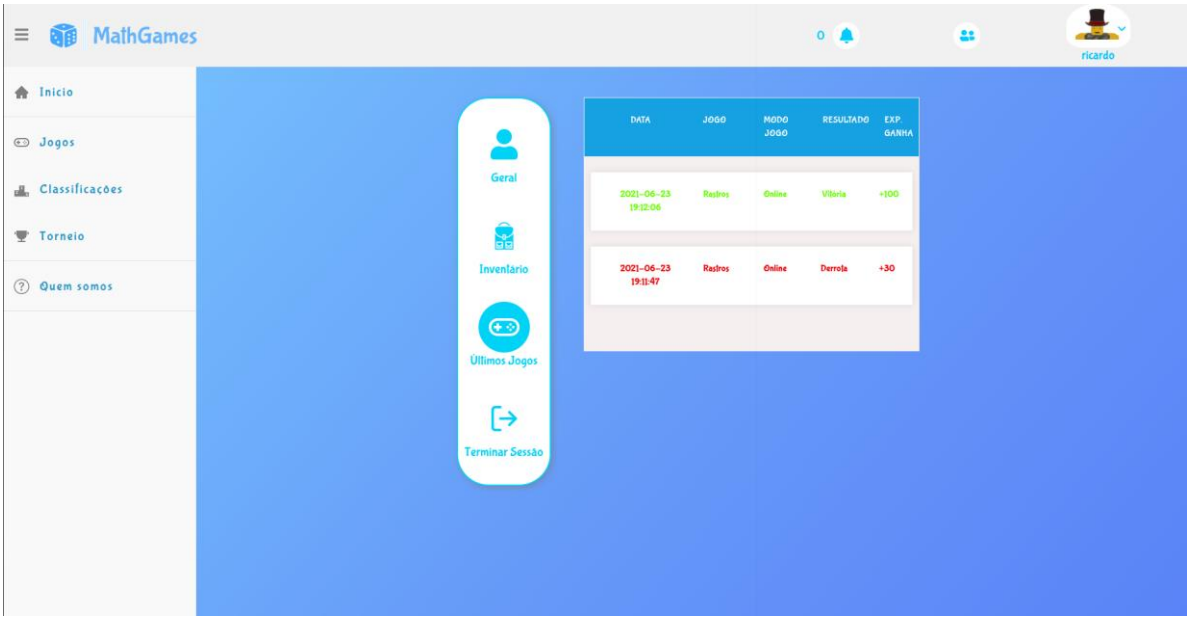


Figure 18 - Web Match History Page

He can see details such as the game he played, the mode and the experience points he gained.

Tournaments

Tournaments are another key functionality of MathGames. For that , we developed a page where all the tournaments available, that can only be created by user with privileges, are displayed and where users can access them. They can filter by game, privacity, name, or even capacity.

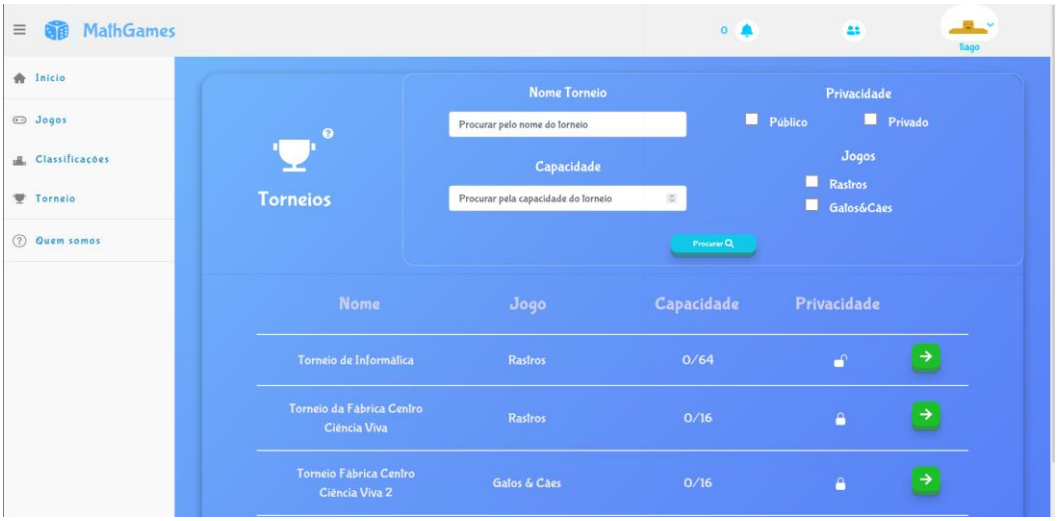


Figure 19 - Web Tournaments Page

When accessing a tournament, if the tournament is private, it is necessary to enter the password (Figure 20).

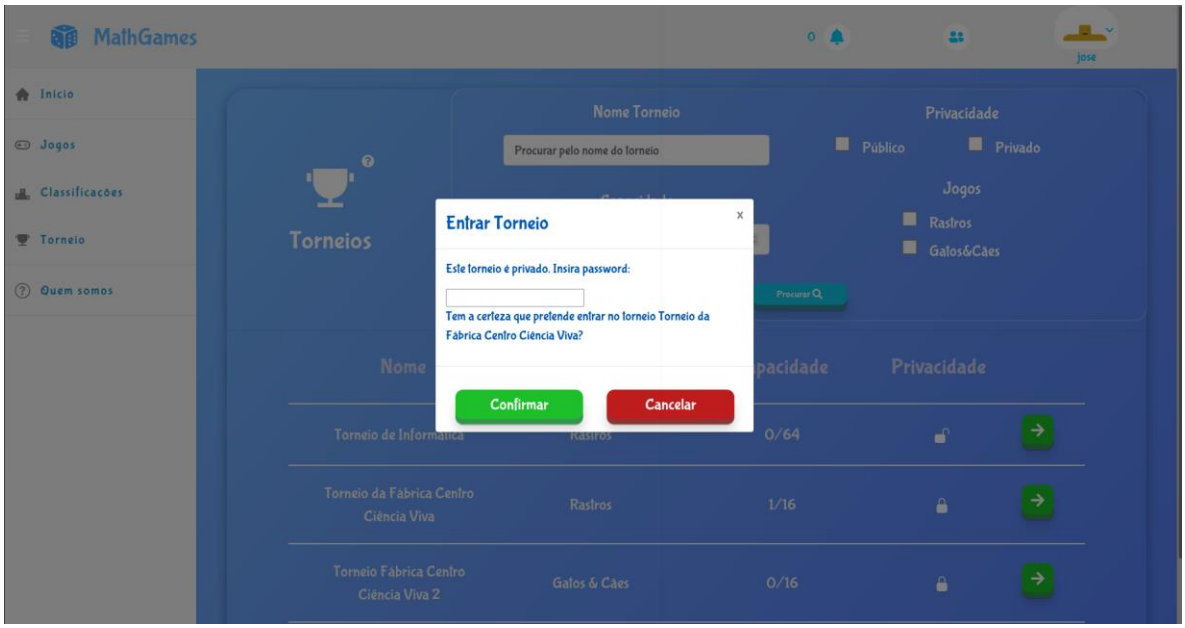


Figure 20 - Web Private Tournament Password Request

When entering the correct password and accessing the tournament page (Figure 21), the user can see tournament details, the other participants, and the current state of the tournament.

Figure 21 - Web Tournament Page

The tournament creator has the possibility to edit the description and the ability to start the tournament and initialize each different stage of the tournament at any

time, these functionalities can be accessed through the tournament's page (Figure 22).

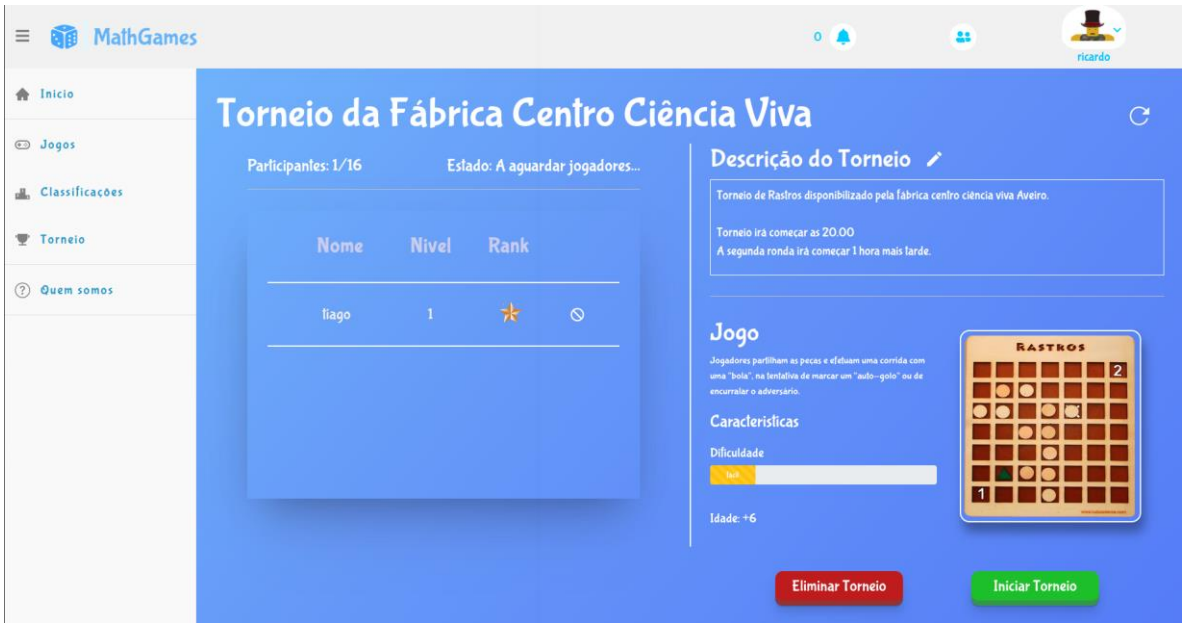


Figure 22 - Web Tournament Page: Creator Point of View

When the creator initializes the tournament, this means that it is not possible to join the tournament anymore. The creator has now the possibility to initiate the next phase, as well as seeing the generated bracket (Figure 23). When the creator initializes it, the participants will receive a notification, and have to do check In. They have a limited time, and if they do not check in within time, they are eliminated from the tournament.

Figure 23 - Web Tournament Page: After tournament start

When the creator initiates the next phase, a notification is sent (Figure 24).

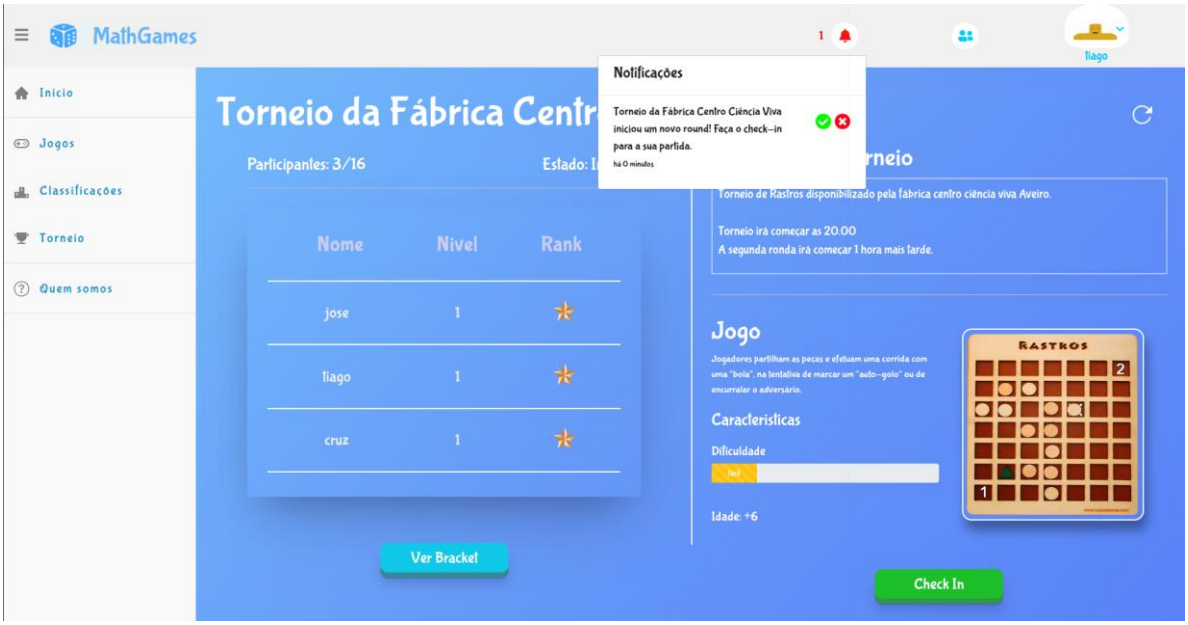


Figure 24 - Web Tournament Start Notification

When the participant checks in, he waits for his opponent to check in as well. When both of them check in, the game is initialized.

This process of checks in and notifications is repeated until the final is played, and the winner is rewarded with a great amount of experience.

To check the current bracket, the players can click on the button “Ver Bracket” and it will display the current bracket (Figure 25).

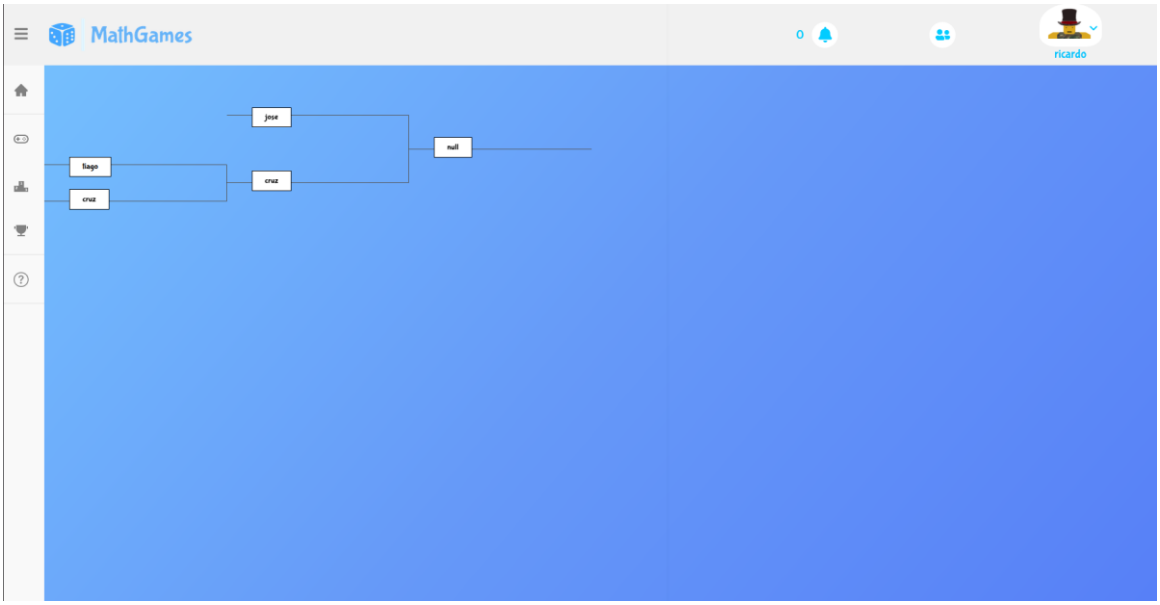


Figure 25 - Web Tournament Bracket

Admin

Apart from all the functionalities described until now, we decided that it would be important to have an admin side, where he could ban players, see statistics about games and players. The sidebar has different functionalities, and from there admins can access players statistics and games statistics

Accessing game statistics (Figure 26 and Figure 27), the admin can see statistics about the most played games, seeing the number of game played of each game.

Figure 26 - Web Admin Statistics Page

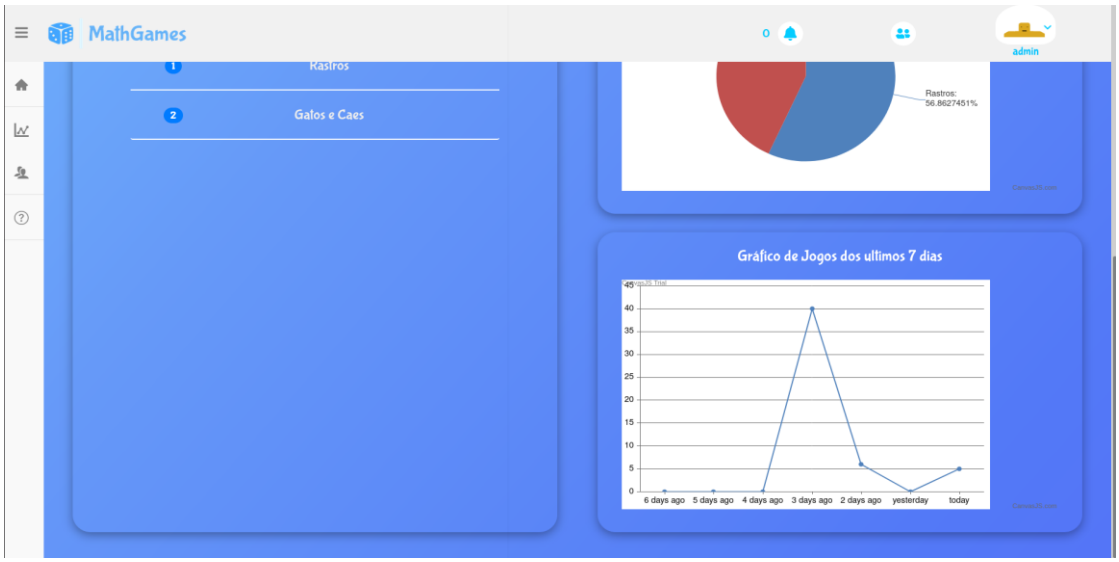


Figure 27 - Web Admin Statistics Page

When accessing a particular game, the admin can see games played and in addition, the distribution of percentages of each rank in that game (Figure 28).

Figure 28 - Web Admin Game Statistics Page

When accessing statistics of players, the admin can see information regarding the new number of players, the number of banned players and a table with the players with more reports (Figure 29).

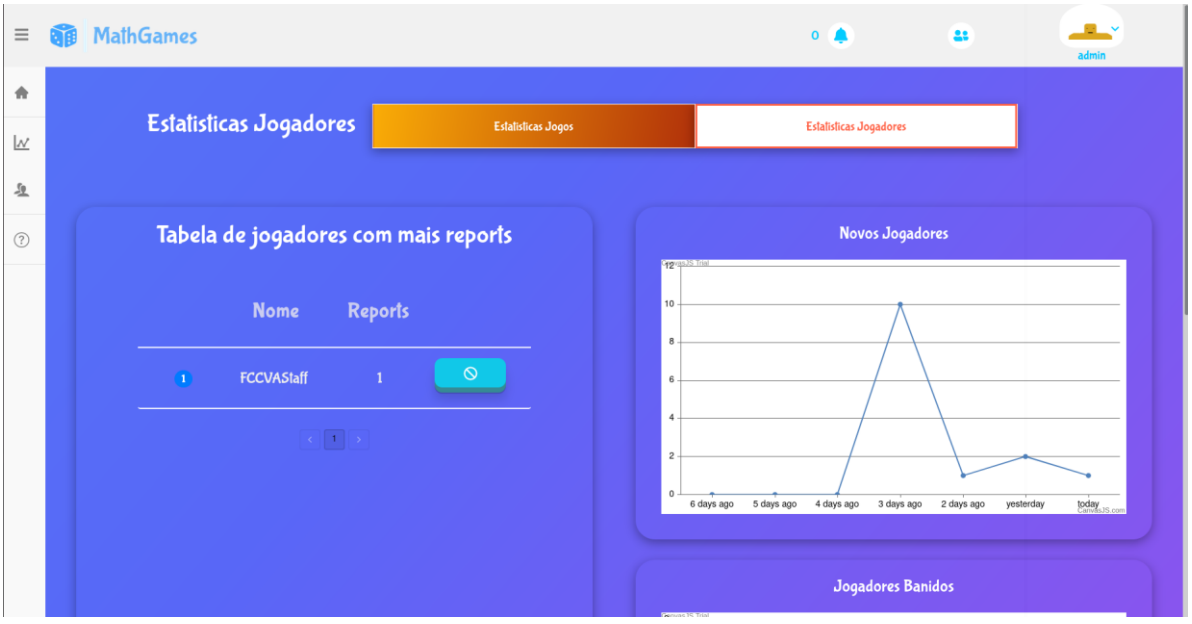


Figure 29 - Web Admin Player Statistics Page

Apart from these statistics, the admin can also check all the players in the application, and search with filters, and from there, he can upgrade his account, downgrade, or ban (Figure 30).

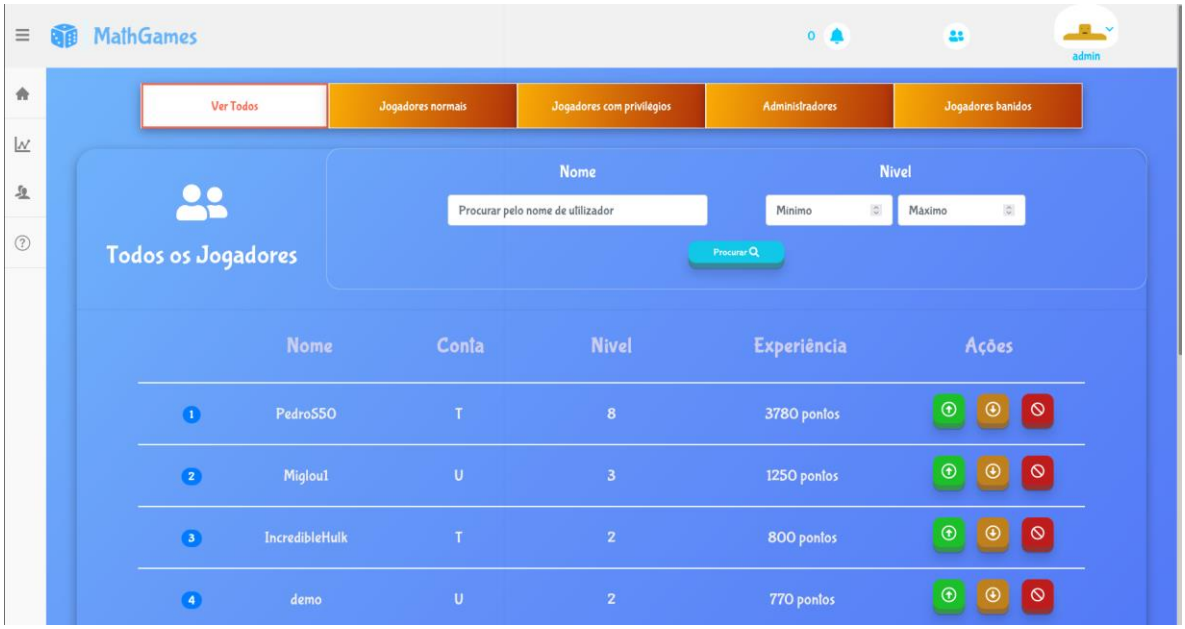


Figure 30 - Web Admin Rankings Page

4.1.2. Mobile Application

Another requirement for this project was the mobile application. This mobile application is destined to users who want to play the games we offer on the platform and should be similar to the web app.

To fulfill this requirement, React Native was used alongside Expo. These technologies are used to develop applications for Android and iOS by enabling developers to use React's framework. The application is working for both operative systems, both applications have the same functionalities and essentially work the same way.

The structure of our Mobile Application is similar to the structure presented in the web app:

- Screens: Components that are used only one time, to display a page with functions and states associated.
- Components: Bits of reusable code that can be used in multiple pages.

- Data: Files that contain static data, that do not contain privacy risks, that are used in components and pages, preventing code duplication.
- Services: Entities that establish the API connection.
- Utilities: Management of the async storage and socket.

Welcome Screen

This is the first page that the user has access to, when he first opens the application (Figure 31).

Figure 31 - Welcome Screen

The user can immediately press the play button and play without an account, he can log in his account or access the lateral menu (Figure 32) where he can find other screens to go to.

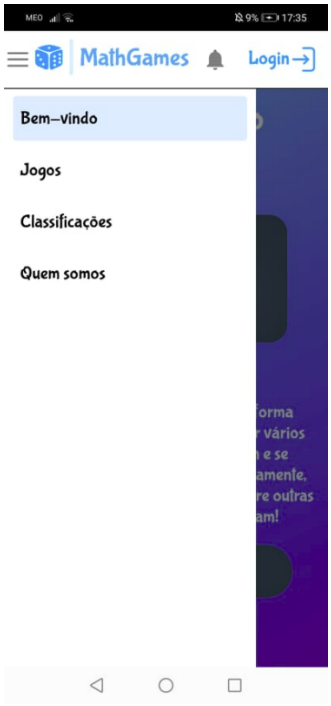


Figure 32 - Side Menu

Login Screen

This screen (Figure 33) is a simple login page, where the user can introduce his credentials to access the account or if he doesn't have an account, create one.

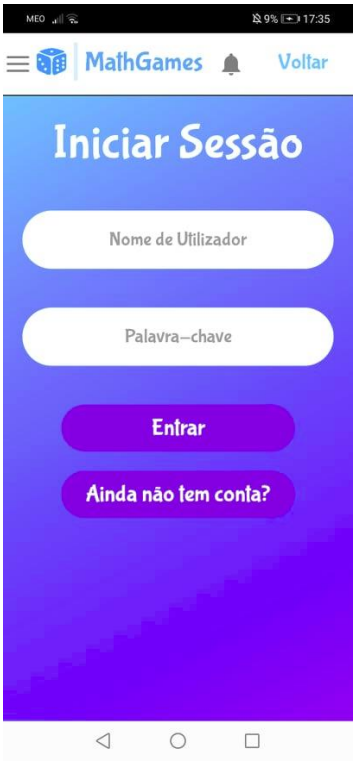


Figure 33 - Login Screen

Choose Games Screen

On the games page (Figure 34), users can see listed the games available and the upcoming games. Every card contains a game and has its name as title, a brief description, an image of the game and the minimum recommended age to play the game. By pressing the card the user will be directed to the game page screen.

Figure 34 - Choose Game Page

Game Page Screen

After the user chooses his game, this screen (Figure 35) will be presented. Here the user can see his rank to that game (only if he's logged into his account) and can choose between the 4 game modes. (Competitive, vs Phone AI and play two persons on the same phone). And the user can press the question mark icon to read the game rules.

Figure 35 - Game Page

Game Screen

Once the user presses the desired game mode, the user is presented with the game board (Figure 36) and in the board the user can see his play possibilities according to the game rules. Along with the board is presented the name of both users, whose turn it is to play, and the same question mark icon as in the previous screen so that the user can check the game rules while playing it.

Figure 36 - Game Board Page

Rankings Screen

In this screen (Figure 37) is presented to the user the players registered in the platform ordered by level. Every user listed will have his name, level and experience at the moment displayed along with two actions if the user is logged in, which are adding or removing a friend, and report player (Figure 37, Figure 38 and Figure 39).

Figure 37 - Ranking Screen

This screen also has the search functionality where any user can search for the ranking of another user by name.

Figure 38 - Friend Request Modal

Figure 39 - Report Player Modal

Friends Screen

For authenticated users, the friends screen (Figure 40) will be available. In this screen the user can see his friends list, add, remove, and invite friends.

There is also a search bar to add new friends by name.

Figure 40 - Friends Page

Profile Screen

The profile page (Figure 41) is only available for users with an account.

In this screen the user has two buttons on top to navigate to other parts of the profile and then is displayed his level with a progress bar and then the user can see for every game his respective rank.

Figure 41 - Profile Page

Inventory Screen

From the profile screen (Figure 42) the user can navigate to the Inventory screen, where is the gamification part of the app. In this screen is where the user can modify his avatar. The user can change the clothes, the hat, the accessories, and the color skin of the avatar . There will be some items that are locked, and the user will need to level up in order to be able to wear them.

After the user is done with the changes, he can store those changes by pressing the save button.

Figure 42 - Inventory Page

Last Games Screen

From the profile screen the user can also navigate to the last games screen (Figure 43). This screen is where the user can go to see the record of his last games.

In that list is presented the date of the game, the game played, and mode played. The lost games have red borders and the games that the user wins have green borders.

Figure 43 - Match History Page

4.1.3. Game Engines

For the development of our game engines, we used Phaser 3, a javascript framework for web game design. On the other hand, we used React Native Game Engine on the mobile app. These frameworks are useful for representing the game itself as well as handling user input on game objects.

At the time of writing this report, our platform provides users with 2 different games to play, Rastros and Gatos & Cães. Both games have similar code implementations, the major differences being in how the inputs are handled and validated. Phaser uses the concepts of games and scenes, being that every game instance can have multiple scenes which can be switched during gameplay, for our implementations, only one scene per game was required. The scene begins by loading all the assets, these are the game's images (board squares, pieces, etc) and the sound played whenever a move is made, and by initializing variables that will be used throughout the game loop, after everything has been loaded, the game canvas is then filled with all necessary objects. The game pieces and squares have associated inputs,

which will trigger when clicked by the player, the input is then handled and, when needed, the result is rendered. When a game finishes, all inputs are disabled.

In React Native Game Engine the game is an array of entities that are rendered on the screen. These entities are react native components so they can be images, texts or even custom components. Then in each game loop, any dispatched events such as asking an AI to make a move or touching a game square to make a move are processed so as to make the necessary changes to the existing entities or add new ones which are then rendered.

For the implementation of Rastros, we also decided that there were 2 features that would be beneficial to users and would improve UX, those features being highlighting the possible moves when a user clicks on the moving piece, and highlighting the last move made so that users, in case they weren't looking at the screen, don't need to figure out what move their opponent played.

4.1.4. Game Over Modal

To improve UX, a simple but effective way to display game over messages was needed, and for that, we developed the GameOverModal component, which renders a modal displaying a custom message. This modal can be triggered by one of two events, the first one occurs when a user is playing a game online, either in competitive mode or against a friend, the match end message received by the WebSocket triggers the modal to appear on screen with the appropriate win or lose message, the second one occurs when a user is playing offline, either in 1v1 offline or against and AI, the finish match method triggers the modal to be rendered on screen.

Due to our games having multiple ways of ending (the piece reached the end goal, the opponent is cornered, an invalid move is played, etc) we needed a way to display different messages based on the different endings, so that our users are always aware of why the match finished. For this to happen, we created a data file which has all endings mapped on a dictionary, which is later accessed by the GameOverModal component.

4.1.5. Timer

To avoid problems associated with never ending matches, like a user being stuck on a match, we decided to introduce timers into our games. These timers are currently set to 5 minutes per user during a match, meaning all moves must be played out in those 5 minutes, if the timer reaches 0 and the game hasn't finished yet, the player loses the match and the opponent wins. When playing online, the timers are created on the server and are never updated by information originated from the client to prevent tampering with the remaining time, so no matter what happens on the client's side, if a player's time reached 0 and no winner has yet been declared, the match will be terminated, and a winner will be declared.

When playing 1v1 offline, the timer won't be stored in the server, but since the players are offline and will gain no rewards, the security aspects aren't relevant. We also decided to not include a timer when playing vs the AI for 2 main reasons, the first one being that it is a mode meant for practice, so the user should take his time learning the game and how to improve without having to worry about the time he's taking, and the second one being that an AI would always play very fast, to fix this we could introduce artificial delays, but the experience would still not feel very realistic.

The timers also introduce a way for players to see who's currently playing. The active player will have it's counter counting down whereas his opponent will have the timer stopped and with its opacity lowered, to make it mode noticeable.

4.1.6. Artificial Intelligence Agents

In order for the user to be able to play our games alone we needed to create an AI that would be able to pose a challenge to the players with various degrees of difficulty. Since our games are turn-based board games with two players in which each player does one move per turn, our teachers advised us to use the Minimax Algorithm with Alpha-Beta pruning. This is a tree search algorithm similar to depth search with limit but to each terminal node an heuristic is assigned. In each of the non terminal nodes, its heuristic is derived from the leaf nodes heuristics being the biggest of them if node depth is even and the smallest if odd. This alternation simulates each player alternately trying to make the best possible move.

Although this logic is the same for each one of the games, the algorithms differ both in the way to calculate the heuristic and the depth of it. In “Rastros” we need to put the piece in a specific position while only being able to move the piece one square in every direction. Given the objective, the distance between the piece and the goal was used as heuristic, having also a big number representing the AIs goal and a small number representing the player goal. On the other hand, in “Gatos&Caes” we must make it so that the adversary has no more moves left while only being able to put pieces where there is not already neighboring opponent piece above, below, left or right of that square. With these rules the heuristic implemented for this game ended up being the number of moves the opponent has left. In terms of depth we decided to use an old smartphone as the reference point so that we could obtain a depth that would not slow down the game and then as the game goes on and less moves are possible it increases.

Finally, to have different difficulty levels we decided to use a probability of good move: 0.2 for easy, 0.5 for medium and 0.8 for hard. Therefore, as the difficulty of the game goes up the AI uses the above algorithm more times. In the other moves the choice is mostly random, preventing only clearly bad plays and assuring a win, if possible, in that turn.

4.1.7. Game Timers

To avoid problems associated with never ending matches, like a user being stuck on a match, we decided to introduce timers into our games. These timers are currently set to 5 minutes per user during a match, meaning all moves must be played out in those 5 minutes, if the timer reaches 0 and the game hasn’t finished yet, the player loses the match and the opponent wins. When playing online, the timers are created on the server and are never updated by information originated from the client to prevent tampering with the remaining time, so no matter what happens on the client’s side, if a player’s time reached 0 and no winner has yet been declared, the match will be terminated, and a winner will be declared.

When playing 1v1 offline, the timer won’t be stored in the server, but since the players are offline and will gain no rewards, the security aspects aren’t relevant. We also decided to not include a timer when playing vs the AI for 2 main reasons, the first

one being that it is a mode meant for practice, so the user should take his time learning the game and how to improve without having to worry about the time he's taking, and the second one being that an AI would always play very fast, to fix this we could introduce artificial delays, but the experience would still not feel very realistic.

The timers also introduce a way for players to see who's currently playing. The active player will have its counter counting down whereas his opponent will have the timer stopped and with its opacity lowered, to make it more noticeable.

4.1.8. Customized Avatar

Our idea to fulfill the functional requirement that says that the users can be able to customize their avatars was implemented in the web app with ThreeJS, react-three/fiber and react-three/drei and in the mobile app with ThreeJs, expo-gl and expo-three.

Figure 44 - Avatar Customization Page

Our avatar scene consists in simple box geometries and rounded box geometries, with a perspective camera pointing directly to the avatar.

Is possible to interact with the avatar making use of orbit controls. The orbit controls allow the user to change the camera position with the mouse only, or with mouse and keyboard combinations. That means that the user can make the camera orbit around the avatar.

Another way to interact with the avatar is by changing the avatar's wear. Is possible to change the avatar's hat, shirt, trousers, and accessories (Figure 44). Those pieces are models for the hats and accessories, and are textures for the clothes.

The models and textures used for the avatar customizations were made by a third party, those third parties are credited in the about us page.

Some of those customizations are locked and in order to unlock them the user needs to play so that his level increases. Every five levels the user is rewarded with a new item.

The avatar is presented in the profile, inventory and in the navbar we have an avatar too, but with a camera positioned closer to the avatar only showing avatar head and shoulders.

4.2. Backend

4.2.1. API Endpoints

At the server we have built a Rest API able to respond to all the requests required to perform the operations required by the users. Our api is divided by eleven files (one per controller). We will list the controllers and the endpoints:

Ban Controller

1. create - Creates a new ban.
2. findAll - Retrieves all bans.
3. statistics - Retrieves statistics regarding how many players were banned per day in the last week.
4. findOne - Retrieve a ban by its id.
5. deleteAll - Removes all bans.
6. delete - Remove a ban by its id.

Friend Controller

1. create - Creates a new friend relation.
2. findAll - Retrieves all friends' relations.
3. findByUserId - Retrieves all friends' relations from a user by its id.
4. delete - Removes a friend's relation by both players ids.
5. deleteAll - Removes all friend's relations

Game Controller

1. create - Creates a new game.
2. findAll - Retrieves all games.
3. findById - Retrieves a game by its id.
4. update - Updates a game by its id.
5. delete - Removes a game by its id.
6. deleteAll - Removes all games

GameMatch Controller

1. create - Creates a new match.

2. findAll - Retrieves information of all matches.
3. statistics - Retrieves statistics regarding games played per day in the last week.
4. statisticsbygame - Retrieves statistics regarding percentage of each game played.
5. findOne - Retrieves information of match by its id
6. update - Updates a match by its id
7. delete - Removes a match by its id.
8. deleteAll - Removes all matches.

Notification Controller

1. create - Creates a new notification.
2. findAll - Retrieves all notifications.
3. findById - Retrieves a notification by its id.
4. delete - Removes a notification by its id.
5. deleteAll - Removes all notifications.

Report Controller

1. create - Creates a new report.
2. findAll - Retrieves all reports.
3. findUsersWithMostReports - Retrieve top 10 users that have more reports.
4. findByReceiverId - Retrieves reports by its receiver id
5. delete - Removes a report by its id.
6. deleteAll - Removes all reports.

Tournament Controller

1. create - Creates a new tournament.
2. join - Player want to join a specific tournament.
3. initialize - Tournament owner initializes the tournament. Instances are created on the database.
4. initializeround - Tournament owner initializes a new round.
5. findOne - Retrieves information of a tournament by its id.
6. findByName - Retrieves information of a tournament by its name.

7. findByCreator - Retrieves information of all tournaments that belong to creator by creator id.
8. update - Updates tournament information by its id
9. delete - Removes a tournament by its id.
10. deleteAll - Removes all tournaments.

Tournament Match Controller

1. create - Creates a new match from a tournament.
2. findAll - Retrieves information from all tournament matches.
3. findMatchesByTournament - Retrieves all matches from a specific tournament by tournament id.
4. update - Updates information from a specific tournament match by tournament_match_id.
5. delete - Removes a tournament match by its id.
6. deleteAll - Removes all tournament matches.

Tournament User Controller

1. findAll - Retrieves information from all tournament users.
2. findUsersByTournament - Retrieves all users of a tournament by tournament id.
3. findTournamentsByUser - Retrieves all tournaments of a user by user id.
4. delete - Removes a tournament user by its id.
5. deleteAll - Removes all tournament users.

User Controller

1. create - Creates a new user account
2. authenticate - Executes login operation
3. findAll - Retrieves information from all users.
4. statistics - Retrieves statistics regarding how many new accounts were created per day in the last week.
5. findAllBanned - Retrieve all Users that have been banned.
6. findAllNormal - Retrieve all Users with normal accounts
7. findAllPrivilege - Retrieve all Users with privilege accounts
8. findAllAdmin - Retrieve all Users with admin permissions

9. findByName - Retrieves a single user by its username
10. findOne - Retrieves a single user by its id
11. update - Updates a user by its id.
12. upgradeAccount - Upgrades user privileges by its id
13. downgradeAccount - Downgrade user privileges by its id
14. delete - Removes a user by its id.
15. deleteAll - Removes all users.

UserRanks Controller

1. findAll - Retrieves ranking from all users.
2. statistics - Retrieve percentage of users per rank by game id.
3. findByUserId - Retrieve ranks of a user by user id.
4. delete - Delete ranks of a user by user id.
5. deleteAll - Delete ranks of all users

4.2.2. Websockets

We needed a mechanism able to ensure fast communication between a client and the server, since we are playing games. If we had created API endpoints and lay on them all the logic needed to play the games, this would not only require a lot of http requests over the time but would also be a poor solution, being the communication too slow.

Considering this, we have chosen to use WebSockets to perform all the communication regarding games and tournaments guarantee fast speed.

In terms of the implementation, we create a socket (Figure 45) in every client as soon as he joins our website:

```
import {urlAPI} from "./data/data";
import io from "socket.io-client";
let socket = io(urlAPI);
export default socket;
```

Figure 45 - Socket Creation

Now, let's take a look at every communication that can be done between the client, and the server. Firstly we are going to start by client's side:

1) User wants to play online game in competitive mode

After selecting competitive mode and clicking the start game button in the game's page, the user emits a message to "enter_matchmaking" event (Figure 46), passing his user_id (which can be the id of the user in the database if the user have an account or the id attributed by us whenever he joins our website) and the id of the game that he wants to play.

Then, the user waits for a server message that will inform him with the match_id and the name of the players that will play the match (Figure 47). With this information, user is able to start the game

```
} else if (gameMode === "online") {
    setSearchingForMatch(true);
    socket.emit("enter_matchmaking", {"user_id": AuthService.getCurrentUserId(), "game_id": game_id})
    create_match_found_listener("online", null);
```

Figure 46 - Online Mode Selection

```
function create_match_found_listener(gameMode, tournament) {
    socket.off("match_found");
    socket.once("match_found", (msg) => {
        setSearchingForMatch(false)
        var match = { match_id: msg['match_id'], player1: msg['player1'], player2: msg['player2'] };
        dispatch( addMatch(match) );
        history.push({
            pathname: "/game/?g="+game_id,
            state: {
                game_id: game_id,
                game_mode: gameMode,
                ai_diff: AIdiff,
                match: match,
                tournament: tournament
            }
        })
    })
}
```

Figure 47 - Match Found Listener

2) User wants to play with a friend through invite by link game

When the user wants to play through a link, after selecting this game mode and clicking the play button, he will emit a message to the event "generate_link" passing his user_id.

After it, two listeners are created (Figure 48). The first listener on the event “invite_link” is used to get the link from the server, the link that the user should share with his friend.

A second listener is also created, on the event “match_found”. This listener is used to inform the user that his friend has already joined the match, providing the match_id and both players names. The user is now able to start the game.

```
if (gameMode === "amigo") {
    socket.emit("generate_link", {"user_id": AuthService.getCurrentUserId(), "game_id": game_id})

    socket.once("invite_link", (msg) => {
        let new_match_id = msg['match_id'];

        if ( new_match_id === null ) {
            alert("Criaste um link recentemente, espera mais um pouco até criares um novo.")
            return;
        }

        friend_match.current = new_match_id;
        setGerarLinkMode(true);
    })

    create_match_found_listener("amigo", null);
```

Figure 48 - Against a Friend Mode

3) User joins a link of an invite by link game

Whenever a user opens the link provided by his friend that started an invite by link game, the user will create a listener on the event “match_found” in a similar way as his friend, which we showed previously.

After it, a message is sent to the event “entered_link” with the user_id, match_id and game_id (Figure 49). This message is used to notify the server that the invited player has joined the match, being now both players ready to start playing.

```
} else if ( new_match_id !== null ) {
    //Código executado pelo jogador que entra por link no jogo
    create_match_found_listener("amigo", null);

    socket.emit("entered_link", {"user_id": AuthService.getCurrentUserId(), "match_id": new_match_id, "game_id": game_id})
}
```

Figure 49 - Friend Joined Section

4) User invites a friend in the platform to play

Users are also able to invite friends on the platform to play, directly through the friend list. When it comes to the communication done by websockets (Figure 50), firstly the user sends a message to the “generate_invite” event where he passes his user_id, outro_id (other player id) and game_id. This event will generate a match_id for this match.

After it, at the listener “invite_link” the user will receive the match_id.

To finish, the listener on the event “match_found” is also created to provide the match_id and both players name information. With this, the game is created and both players are able to start playing.

```
if (localStorage.getItem("jogoporinvite")) {
    //Código executado pelo jogador que convida outro por amizade para jogar
    localStorage.removeItem("jogoporinvite")
    let id_outro_jogador = parseInt(localStorage.getItem("outrojogador"))
    localStorage.removeItem("outrojogador")

    socket.once("invite_link", async (msg) => {
        let new_match_id = msg['match_id'];

        if ( new_match_id === null ) {
            alert("Envias-te um convite recentemente. Espera mais um pouco para puderes enviar um novo.")
            return;
        }
                                    (method) AuthService.getCurrentUser(): any
        var current_user = AuthService.getCurrentUser()

        var notification_text = current_user.username + " convidou-te para jogares."

        //await userService.send_notification_request(current_user.id, id_outro_jogador, "P", notification_text);
        socket.emit("new_notification", {"sender": current_user.id, "receiver": id_outro_jogador, "notification_type": "P", "notification_text": notification_text})

        friend_match.current = new_match_id;
        setInviteFriendMode(true);
    })
    create_match_found_listener("amigo", null);

    socket.emit("generate_invite", {"user_id": AuthService.getCurrentUserId(), "outro_id": id_outro_jogador, "game_id": game_id})
```

Figure 50 - Game By Invite Section

5) User enters a game through an invite by a friend on the platform

On the other side, the user that accepts the invite also communicates with the server through sockets (Figure 51). The communication done by this specific user can be divided in two phases.

Firstly, when he accepts the game at the notification dropdown, he will send a message to the server to the listener “get_match_id” where he provides his user_id, and the user_id of the player who invited him (outro_id). The response sent by the server reaches the user at the “match_link” listener. Inside this listener, the user receives the match_id and game_id values, which are used to create the url that will guide the user to the match.

Secondly, already inside the game's page, the user creates a listener in the “match_found” event, that as we have seen before, is responsible for receiving match_id and both players name information (Figure 52).

```
async function accept_game(notification, index) {
    deleteNotification(index);
    await UserService.delete(notification.id);
    var id_outro_jogador = notification.sender_user.sender_id

    socket.once("match_link", (msg) => {
        if (msg["match_id"]) {
            let new_match_id = msg['match_id'];
            let game_id = msg['game_id']

            localStorage.setItem("entreijogoporinvite", true)
            localStorage.setItem("outrojogador", id_outro_jogador)
            var url = "/gamePage?id="+game_id+"&mid=" + new_match_id
            history.push(url)
            //window.location.assign(urlWeb+url)
        } else if (msg["error"]) {
            setConfirmModalShow(true)
        }
    })

    socket.emit("get_match_id", {"user_id": AuthService.getCurrentUserId(), "outro_id": id_outro_jogador})
}
```

Figure 51 - Accept Game Method

```
    //Código executado pelo jogador que aceita o convite de jogo por amizade
    localStorage.removeItem("entreijogoporinvite")

    let id_outro_jogador = parseInt(localStorage.getItem("outrojogador"))
    localStorage.removeItem("outrojogador")

    create_match_found_listener("amigo", null);

    socket.emit("entered_invite", {"user_id": AuthService.getCurrentUserId(), "outro_id": id_outro_jogador, "match_id": new_match_id, "game_id": game_id})
```

Figure 52 - Friend Joined Method

6) Tournament owner initializes a new round

Another feature of our project that involves sockets is the logic behind the tournament's execution.

In our implementation we gave to the tournament owner the freedom to initiate the next round of his tournament whenever he decided, with some conditions of course (Figure 53).

Therefore, whenever a tournament owner requires a new round to initiate, after a first verification that all the conditions are fulfilled, the owner emits a message to “tournament_newround” event passing his user_id and tournament_id. Server will

then run a block of code responsible for verifying the conditions again, to ensure that everything is right and, in the end, updates his internal state and information regarding this tournament.

To conclude this operation, the owner also creates a listener to receive the feedback of the operation.

```
socket.emit("tournament_newround", {"user_id": AuthService.getCurrentUserId(), "tournament_id": tournament_id})

socket.once("round_start", (msg) => {
    let erro = msg['erro'];
    if ( erro ) {
        elemento = document.getElementById("erroInitializeRound")
        if (elemento !== undefined && elemento !== null)
            elemento.style.display = "flex"
    } else {
        elemento = document.getElementById("initializeRoundSuccess")
        if (elemento !== undefined && elemento !== null)
            elemento.style.display = "flex"
    }
})
```

Figure 53 - Tournament New Round Method

7) User perform check in for his next game in the tournament

Another task regarding tournaments that incorporate the use of sockets is the check in for the next game operation (Figure 54).

To perform this, the user sends his user_id and the tournament_id of the tournament that he is checking in. On the server some conditions are verified as we will see in the future.

Then in the listener “check_in”, the user receives the necessary feedback regarding his operation and operates accordingly, being in case of success supplied with the match_id of his next match.

```
socket.emit("tournament_checkin", {"user_id": AuthService.getCurrentUserId(), "tournament_id": tournament_id})

socket.once("check_in", (msg) => {
    let erro = msg['erro'];
    if ( erro ) {
        let message = msg["message"]
        if (message === "Game has already started")
            setErroCheckIn("O teu jogo já iniciou. Check In já nao é permitido. Caso tenhas vencido a partida, espera que o próximo round seja iniciado.")
        if (message === "You are not participating in this tournament")
            setErroCheckIn("Voçe já foi eliminado ou não pertence a este torneio.")
        if (message === "Tournament is not active")
            setErroCheckIn("Este torneio não se encontra ativo.")
    } else {
        if (msg["message"]) {
            setErroCheckIn("Voçe já esta qualificado para round seguinte. Espere que o próximo round seja iniciado.")
        } else {
            let match_id = msg['match_id']
            history.push("/gamePage?id="+tournament.game_id+"&tid="+tournament.id+"&mid="+match_id)
        }
    }
})
```

Figure 54 - Tournament Check In Method

8) User joins a tournament match

Following the previous operation, after checking in for the next match, as we said in case of success the user receives the match_id of the next game (Figure 55). Thus, he goes to the game's page where he creates a listener in "match_found" event, that as we have seen already multiple times, is responsible for receiving match_id and players information.

Finally, he sends a message to the "tournament_enteredmatch" event, where he provides his user_id, match_id and tournamnet_id. This message informs the server that this specific user is ready to play his match.

```
} else if (tournament_id !== null) {
    //Tournament section
    create_match_found_listener("online", tournament_id);

    socket.emit("tournament_enteredmatch", {"user_id": AuthService.getCurrentUserId(), "match_id": new_match_id, "tournament_id": tournament_id})
```

Figure 55 - User Joins Tournament Method

9) Notifications

Most of the operations performed in our application that require feedback to the users are associated with a notification. These notifications are implemented with sockets. The communication with the server is equal to all the notifications, being a message sent to the event "new_notification". For example, when someone denies an invite for a match sends a notification to the friend who sent the invite (Figure 56).

```
function deny_game(notification) {
    var receiver = notification.sender_user.sender_id
    var notification_text = current_user.username + " recusou o seu convite."
    socket.emit("new_notification", {"sender": current_user.id, "receiver": receiver, "notification_type": "D", "notification_text": notification_text})
}
```

Figure 56 - Deny Game Method

10) Inside match room (game's engine) communication

To conclude the communication done on the client side using sockets, the final operations are incorporated inside the game engine. Inside the game engine, we create two listeners (Figure 57). The "match_end" listener is responsible for receiving a message from the server that indicates that the game has finished. The "move_piece" listener is used to receive the moves executed by our opponent through the server.

According to the previous listener, we also need to send our moves to the server, using the “move” event passing our id, match_id and the information needed to verify the move (Figure 58).

```
socket.on("move_piece", (new_pos) => {
    console.log("Received move: ", new_pos);
    this.move(this.squares_group.getChildren()[new_pos]);
});

socket.on("match_end", (msg) => {
    console.log("Game Over Received")
    if (this.game_over === false)
        this.finish_game(msg)
})
```

Figure 57 - Socket Listeners

```
if ( this.valid_squares.has( parseInt(clicked_square.name) ) ) {
    this.move(clicked_square);
    if ( game_mode === "online" || game_mode === "amigo" )
        socket.emit("move", clicked_square.name, AuthService.getCurrentUserId(), current_match['match_id']);
}
```

Figure 58 - Sending Move Between Players

We have finished the client’s side. Although we have already revealed some aspects on what is done by the server to respond to each type of communication that we have seen, we are now going to watch in a deeper detail the server side:

11) Game by friend invite section

- **“generate_invite”** - This listener is activated by the player who invites a friend to play, directly on the platform. It is responsible for verifying that the user does not have any active invites, to prevent multiple invites in a short time. After it, generates a match_id and updates active_friend_invites dictionary, associating a dictionary with the other players id, the match_id and the game_id to the current user_id key. To finish, it creates a timeout of 2 minutes. After those 2 minutes, the invite will expire.
- **“get_match_id”** - This event is launched when the invited player accepts the invite. It is responsible for verifying that the user that accepts the invite and the player who invited him are correct and have a current open match (if the timeout expires this verification will fail). If everything is right, the server will send a message to the user with the match_id (id generated in the previous listener).

- **“entered_invite”** - Listener activated also by the player who was invited. After receiving the match_id as said in the previous listener, the player will reach the game’s page and will notify the server. This listener verifies that everything is correct (both players ids are correct) and creates the game instance in the server.

```
socket.on("generate_invite", (msg) => {
  let user_id = msg["user_id"]
  let outro_id = msg["outro_id"]
  let game_id = parseInt(msg["game_id"])

  // If the user still has an active link, return null match_id
  if ( Object.keys(active_friend_invites).includes(user_id) ) {
    io.to(socket.id).emit("invite_link", {"match_id": null});
    return;
  }

  users_info[user_id] = socket.id;

  let new_match_id = uuid.v4();
  active_friend_invites[user_id] = {"outro_id": outro_id, "match_id": new_match_id, "game_id": game_id};

  io.to(socket.id).emit("invite_link", {"match_id": new_match_id});
  // After 2 minutes, the invites expires
  setTimeout(() => { delete active_friend_invites[user_id]; }, 120000);
})
```

Figure 59 - Generate Invite Listener

```
// user that accepts friend invite needs match_id.
socket.on("get_match_id", (msg) => {
  if (msg["user_id"] !== null) {
    var user_id = msg["user_id"]
    var outro_id = msg["outro_id"]

    users_info[user_id] = socket.id

    if ( Object.keys(active_friend_invites).includes(String(outro_id)) && active_friend_invites[outro_id]["outro_id"] === String(user_id)) {
      io.to( users_info[String(user_id)] ).emit("match_link", {"match_id": active_friend_invites[outro_id]["match_id"], "game_id": active_friend_invites
    } else {
      io.to( users_info[String(user_id)] ).emit("match_link", {"error": "invite expired"})
    }
  }
})
```

Figure 60 - Get Match Id Listener

```
socket.on("entered_invite", (msg) => {
  console.log("User conected through link.")
  if (msg["user_id"] !== null) {
    var match_id = msg["match_id"]
    var user_id = msg["user_id"]
    var outro_id = msg["outro_id"]
    var game_id = parseInt(msg["game_id"])
    users_info[user_id] = socket.id

    if ( Object.keys(active_friend_invites).includes(String(outro_id)) && active_friend_invites[outro_id]["outro_id"] === String(user_id)) {
      console.log("Vou criar e enviar!")
      create_game(match_id, game_id, user_id, outro_id, "amigo", null)
      //io.to( users_info[outro_id] ).emit("friend_joined", {"match_id": match_id, "player1": user_id, "player2": outro_id})
    }
  }
})
```

Figure 61 - Entered Invite Listener

12) Game by generated link section

- **“generate_link”** - This listener (Figure 62) is activated by the player who generates a link to share with a friend in order to play a game with him. The listener starts by verifying that the user does not have any active links preventing the user from generating multiple links in a shorter time window. After it, the server generates a match_id and sends the match_id to the user, making this match_id a core component of the link generated. Similar to the “generate_invite” listener, this listener creates a timeout of 2 minutes. After 2 minutes, the invite will expire.
- **“entered_link”** - This event (Figure 62) is triggered when the friend enters the link. Server checks that the match_id provided is valid and has not expired yet. If everything is fine, a game instance will be created on the server and the user who has created the link and is waiting for the friend is notified that the game is starting.

```
//User sends user_id and wants to play with a friend through a link
socket.on("generate_link", (msg) => {
  let user_id = msg["user_id"]

  // If the user still has an active link, return null match_id
  if ( Object.values(active_friend_link).includes(user_id) ) {
    io.to(socket.id).emit("invite_link", {"match_id": null});
    return;
  }

  users_info[user_id] = socket.id;

  let new_match_id = uuid.v4();
  active_friend_link[new_match_id] = user_id;

  io.to(socket.id).emit("invite_link", {"match_id": new_match_id});
  // After 2 minutes, the links expires
  setTimeout(() => { delete active_friend_link[new_match_id]; }, 120000);
  console.log("Active friend link:", active_friend_link)
})

socket.on("entered_link", (msg) => {
  console.log("User conected through link.")

  if (msg["user_id"] !== null) {
    var match_id = msg["match_id"]
    var user_id = msg["user_id"]
    var game_id = parseInt(msg["game_id"])
    users_info[user_id] = socket.id

    if ( Object.keys(active_friend_link).includes(match_id) ) {
      console.log("Vou criar e enviar!")
      create_game(match_id, game_id, user_id, active_friend_link[match_id], "amigo", null)
      io.to( users_info[active_friend_link[match_id]] ).emit("friend_joined", {"match_id": match_id, "player1": user_id, "player2": active_friend_link[match_id]})
    }
  }
})
```

Figure 62 - Generate Invite and Entered Link Listeners

13) Online games section

This section is composed by 2 different listeners. It is relevant to say that some of these listeners are used in all game modes that involve online games (competitive, invite by link, invite by friend list):

- **“enter_matchmaking”** - This listener (Figure 63) is activated whenever a player wants to play an online competitive game. The server puts the user in a match_queue. For every different game, there is a different queue. After it, the server verifies if the queue already has two or more players. If so, the server will take the first two entries, verify that they are not the same and create the game. If both entries correspond to the same player, the server puts the player back in the front of the queue.
- **“leave_matchmaking”** - This listener is activated whenever a player expresses that he wants to leave the matchmaking queue. The server verifies if the user is in a queue and, if so, removes the user from the queue.
- **“forfeit_match”** - This listener is used by a player that expresses his willingness to forfeit a match. The server verifies that the user belongs to the match and, if so, ends the match and notifies both players.
- **“move”** - This listener (Figure 64) is used by every online game mode and is triggered every time that a player performs a move in the game. The server first verifies that the match_id is valid. After it, the server checks that the player is valid for the passed match_id. If both steps aboard before are successfully passed, the move is verified by a function called “valid_move”. This function will evaluate the move using the current game state saved by the server. If the move is invalid, the game will finish and the current player will lose it. On the other hand, if the move is valid, it will be passed to the other user to make the move displayed in his screen, the game state will be updated in the server and the timers will switch. If this move is a final one, the match will also finish and both players will be notified.

```
//User says that he wants to play Online and put himself in matchqueue list
socket.on("user_id", (msg) => {
  console.log(msg)
  var user_id = msg["user_id"];
  var game_id = parseInt(msg["game_id"]);
  users_info[user_id] = socket.id;
  match_queue[game_id].push(user_id)

  if ( match_queue[game_id].length >= 2 ) {
    console.log("Match found.");
    var player1 = String( match_queue[game_id].shift() );
    var player2 = String( match_queue[game_id].shift() );
    if (player1 !== undefined && player2 !== undefined ) {
      if (player1 !== player2)
        create_game(null, game_id, player1, player2, "online", null);
      else
        match_queue[game_id].unshift(player1)
    } else {
      if (player1 !== undefined) match_queue[game_id].unshift(player1)
      if (player2 !== undefined) match_queue[game_id].unshift(player2)
    }
  }
});
```

Figure 63 - On User Id Listener

```
//User sends match id, userid and new pos when he wants to make a move in the game
socket.on("move", (new_pos, user_id, match_id) => {
  user_id = String(user_id);
  match_id = String(match_id);

  if ( Object.keys(current_games).includes(match_id) )
    if ( Object.keys(current_games[match_id]['users'] ).includes(user_id))
      if ( valid_move(user_id, match_id, new_pos) ) {
        let opponent = current_games[match_id]['users'][user_id][0]
        io.to( users_info[opponent] ).emit("move_piece", new_pos);

        // Pause user's timer and restart opponent's
        current_games[match_id]['timers'][user_id].pause();
        current_games[match_id]['timers'][opponent].start();

        if ( current_games[match_id]['state']['isFinished'] ) {
          current_games[match_id]['timers'][user_id].pause();
          current_games[match_id]['timers'][opponent].pause();

          let endMode = current_games[match_id]['state']['extra'];

          finish_game(match_id, endMode)

        }

      } else {
        //Move is not valid. Match will end and oponnent will win.
        let opponent = current_games[match_id]['users'][user_id][0]

        current_games[match_id]['state']['isFinished'] = true
        current_games[match_id]['state']['winner'] = (user_id === current_games[match_id]['state']['player1']) ? "2" : "1"
        current_games[match_id]['timers'][user_id].pause();
        current_games[match_id]['timers'][opponent].pause();

        finish_game(match_id, "invalid_move")
      }
})
```

Figure 64 - On Move Listener

14) Tournament operations section

- **“tournament_newround”** - This listener is activated by the owner of a tournament when he notifies the server that wants to start a new round of the tournament. First, we verify that the user is actually the tournament owner. Then we select all matches of the next round that are already created in the database and with this information we update the internal server

active_tournaments dictionary. After this, we select all players that belong to this tournament and have not been eliminated yet to notify every one of them that a new round is starting. Finally, we update the current_round of the tournament in the database.

- **“tournament_checkin”** - This listener (Figure 65) is launched whenever a player wants to perform his check in for the next match in a specified tournament. Firstly, the server verifies that the tournament passed is active. Then, the server checks that the player is a participant of the passed tournament. In addition, the server gets the match_id of the match that this player should play in the passed tournament in the current round of the tournament and verifies if the match has already started or not. In the case that the match has already started, this means that the player should wait for the next round of the tournament because he has already won the match corresponding to the current round of the tournament. If the match hasn't started yet, the server will include the player in the players_in (list of checked in players) and will notify the user that the operation had success providing the match_id.
- **“tournament_enteredmatch”** - This listener (Figure 66) is triggered whenever a player joins a tournament match, providing to the server the match_id, user_d and tournament_id. Firstly, the server verifies that the tournament passed is active. Then, the server checks that the player is a participant of the passed tournament. After it, it verifies that this user has already checked in. Finally, the server checks if both users have already entered the match or if it is the first user joining in. If both players have already entered the match, the game will be created on the server, notifying both players that the game is starting.

```
//Tournament Players checkin for their games
socket.on("tournament_checkin", async (msg) => {
  var user_id = msg["user_id"];
  var tournament_id = msg["tournament_id"];

  users_info[user_id] = socket.id;

  if ( Object.keys(active_tournaments).includes(tournament_id) ) {
    if ( Object.keys(active_tournaments[tournament_id]["players"]).includes(String(user_id)) ) {
      var match_id = active_tournaments[tournament_id]["players"][String(user_id)]
      if (active_tournaments[tournament_id]["rooms"][match_id]["started"] === false) {
        if (!active_tournaments[tournament_id]["rooms"][match_id]["players_in"].includes(user_id) ) {
          active_tournaments[tournament_id]["rooms"][match_id]["players_in"].push(user_id)
        }
        io.to( socket.id ).emit("check_in", {"erro": false, "match_id": match_id});
      }
      io.to( socket.id ).emit("check_in", {"erro": true, "message": "Game has already started"});
    } else {
      io.to( socket.id ).emit("check_in", {"erro": true, "message": "You are not participating in this tournament"});
    }
  } else {
    io.to( socket.id ).emit("check_in", {"erro": true, "message": "Tournament is not active"});
  }

});
```

Figure 65 - Tournament Check In Listener

```
//Tournament Players Enters Game
socket.on("tournament_enteredmatch", (msg) => {

  if (msg["user_id"] !== null) {
    var match_id = msg["match_id"]
    var user_id = msg["user_id"]
    var tournament_id = msg["tournament_id"]

    users_info[user_id] = socket.id

    //Verify tournament exists
    if ( Object.keys(active_tournaments).includes(tournament_id) ) {
      //Verify this user is on the tournament
      if ( Object.keys(active_tournaments[tournament_id]["players"]).includes(String(user_id)) ) {
        //Verify this user is already checked-in
        if ( active_tournaments[tournament_id]["rooms"][match_id]["players_in"].includes(user_id) ) {
          //Verify both users have already entered game
          if ( Object.keys(active_tournaments[tournament_id]["rooms"][match_id]["players_in"]).length === 2 ) {
            var player1 = active_tournaments[tournament_id]["rooms"][match_id]["player1"]
            var player2 = active_tournaments[tournament_id]["rooms"][match_id]["player2"]
            active_tournaments[tournament_id]["rooms"][match_id]["started"] = true
            create_game(match_id, active_tournaments[tournament_id]["torneio_info"]["game_id"], player1, player2, "online", tournament_id)
          }
        } else {
          active_tournaments[tournament_id]["rooms"][match_id]["players_in"].push(user_id)
        }
      } else {
        io.to( socket.id ).emit("match_found", {"erro": true})
      }
    } else {
      io.to( socket.id ).emit("match_found", {"erro": true})
    }
  }
})
});
```

Figure 66 - Tournament End Match Listener

15) Notification section

- **“new_notification”** - This listener (Figure 67) is launched whenever a user sends a new notification to the server. The server creates the new notification, saves the notification in the database, and notifies the receiver.

```
socket.on("new_notification", (msg) => {
  let sender = String(msg["sender"])
  let receiver = String(msg["receiver"])
  let notification_type = msg["notification_type"]
  let notification_text = msg["notification_text"]

  create_notification(sender, receiver, notification_type, notification_text)
})
```

Figure 67 - New Notification Listener

4.2.3. Database Connection

As we have described during our architecture chapter, we have used a mysql persistence database. The connection between our node server and the database itself was performed using the configuration file displayed on (Figure 68).

In this file we passed the host, port, user, password and the name of the database, to allow the node server to connect to our mysql database that was running on its own container. The connection is established through the pool specified, which prevents the connection from crashing whenever mysql throws an error.

```
1  module.exports = {
2      HOST: '127.0.0.1',
3      PORT: "3306",
4      USER: "demo",
5      PASSWORD: "password",
6      DB: "demo",
7      TIMEZONE: "+00:00",
8      dialect: "mysql",
9      pool: {
10         max: 5,
11         min: 0,
12         acquire: 30000,
13         idle: 10000
14     }
15 };
```

Figure 68 - Module Exports

We have also used an ORM (Object Relational Mapper) tool, Sequelize, that keeps the management of the database easier. Sequelize maps an object syntax onto our database schemas. ORM also helped us to set the initial state of our database in a dynamic way. Besides the table creation, we also create some registers that are

stored on our database from the beginning , for example, games information. Apart from that, we also created an admin account. All these steps are done conditionally (if the database already has these registers nothing is done) meaning that we do not have to be worried every time we want to reset our database, in the development environment, to create all these registers manually.

4.2.4. Authentication & Authorization

The registration process in our application is very simple. We tried to keep it as simple as possible, requiring the minimum of personal information from our users, since our main target is younger people that might be willing to provide too much personal information without knowing the risks associated with it. Therefore, we only request our users to give a username, a password, and an email. During the process we guarantee that both the email and the username are unique, and the password is verified and must pass some conditions that ensure a minimum level of security

Considering that our application works with users being these users able to create personal accounts we needed a mechanism to authenticate and get the list of authorizations of each person. Having this in mind, we have chosen to use Json Web Tokens Authentication that uses cryptography to ensure that everyone is who they claim to be.

We create a new token (Figure 69) in every login and to create these tokens we have a secret key and an expiration time of 24h. We also stored the user's personal id as well as the type of account that the user has.

```
if (!response.dataValues.password ||
 !await response.validPassword(password.toString(),
  response.dataValues.password)) {
   resolve(false);
} else {
 const token = jwt.sign({ id: response.id, account_type: response.account_type }, config.secret, {expiresIn: 86400});
 const { password, ...userWithoutPassword } = response.dataValues;
```

Figure 69 - Authentication Token Generation

Whenever an operation is performed, if the operation requests the interaction with our REST Api we send the user personal token. Then we have some levels of security:

1. **Public Addresses** - Does not need any verification. These routes are open to everyone. Exemple: Login route (Figure 70);

```
// Login
router.post("/login", users.authenticate);
```

Figure 70 - Login Endpoint

2. **Normal Account Addresses** - Need the verification that a user is logged in and the user is who he claims to be. Example: Update user route (Figure 71);

```
// Update a User with id
router.put("/:id", authJwt.verifyToken, users.update);
```

Figure 71 - Update User Endpoint

3. **Tournament Account Addresses** - Need the verification that a user is logged in, the user is who he claims to be, and the type of account is at least a tournament privilege account. Example: Create a tournament route (Figure 72);

```
// Create a new tournament
router.post("/", [authJwt.verifyToken, authJwt.isTournamentManager], tournaments.create);
```

Figure 72 - Create New Tournament Endpoint

4. **Admin Account Addresses** - Need the verification that a user is logged in, the user is who he claims to be, and the type of account is an admin account. Example: Delete user route (Figure 73).

```
// Delete all Users
router.delete("/", [authJwt.verifyToken, authJwt.isAdmin], users.deleteAll);
```

Figure 73 – Delete Users Endpoint

These verifications are performed by the functions that are displayed in the respective figures. Looking into those functions with more detail:

1. **verifyToken** - Function that takes the token provided by the user and verifies its integrity, assuring that the token was not modified or manipulated. If the token is valid, this function also decodes the token to retrieve the user id and the account type that were stored in the token in its creation phase.

2. **isTournamentManager** - Function that takes the user's personal id that was decoded in the previous function to get the user from the database and check if the account is of tournament type. This process is done to prevent accounts that have seen their privileges downgraded after the user has already logged in (token still has the old privileges) to perform tasks that require those permissions.
3. **isAdmin** - Function that takes the user's personal id that was decoded in the verifyToken function to get the user from the database and check if the account is of admin type. This process is done to prevent accounts that have seen their privileges downgraded after the user has already logged in (token still has the old privileges) to perform tasks that require those permissions.

This mechanism allowed us to have a strong request's authorization control. It is also a simple mechanism where we are able to change the permissions of a specific route quickly.

Chapter 5. Results and Discussion

5.1. Planned and Implemented Features

5.1.1. Game Modes

Play game with a friend on the same device

The first feature that we are going to talk about is the offline game mode with a friend, where we allow our users to play with a friend on the same device. Since the game runs in an offline mode, users are not required to have internet access. They are not also required to have two technological devices being only one required.

Play game against the computer

The second feature is also an offline game mode. We give the possibility to play games against an artificial intelligence bot. We also created three different AI Bot difficulties in order to allow users to select the appropriate level to their current skill. This feature might be useful to anyone who wants to practice to get better at the games and to people that don't have friends to play with. Similar to the previous feature, this one also does not require internet access.

Play game online in competitive mode

Having in mind that people might not always have friends to play with and bots are nice to practice with but users usually like to play against each other, we decided to implement an online competitive game mode. Here, users can put themselves on a waiting list and try to find matches with someone else across the internet.

Generate a link to play with a friend

As we have already stated in this report, users are allowed to play games even when they do not have an account. Considering this, we decided to create a game mode where two friends that do not have an account on the platform are able to play against each other through the internet. This can be achieved with the generation of a

url link that can be shared amongst users. When accessing the link, both friends are redirected to a match room where they can play against each other.

Play against a friend on the platform

Besides allowing game invites through a link, users may also directly challenge someone from their account's friends list to a match. This requires the users to have previously established a relationship by one of them adding the other as a friend on the platform.

5.1.2. Authentication

Register an account

We have developed a simple registration mechanism that allows users to create personal accounts. We tried to keep the process as simple as possible since we do not want to force our, mostly young, users to provide too much personal information. Some rules over passwords were put in place to provide some additional security over our user's accounts.

Login

After creating an account, users are able to login into their accounts and access new features. In the login process users must fill in their username and password chosen during account creation.

5.1.3. Rankings

Account Level

In our application we try to force players to get better individually and develop their skills. In order to give them the capacity to follow their own progress we have created an account level. The level of each individual account is upgraded every time the user plays a game. If the user wins the game, he will get higher experience points than if he were to lose.

Games Ranking

Even though the account level gives feedback to each user regarding his individual skills, this feedback is limited since the level is a general metric, and can only increase over time. Having this in mind, we decided to implement rankings per game, therefore, each player has a different rank in each game. The ranking is updated on every game played in the competitive mode, if the player wins the match, his ranking points will increase, however, if he loses, they will decrease.

Classification Page

We have created a classification page where all the players are listed ordered by their account level. In this page we provide a filter mechanism by username and level to make searching for a specific player easier.

5.1.4. Friends

Add and Remove Friends

As we said before, we have created the possibility to invite friends on the platform directly to play. Therefore, we had to implement a friend relationship between the players in our platform. Every player is able to invite any other user to be his friend. Players can only send one invite to the same player to avoid flooding the other players notification box. Players are able to accept or refuse the invite and, if they accept the invite, they can remove the player from their friend list anytime.

5.1.5. Bans and Reports

Report a player

In every game platform there exists the issue of players using cheats, abusing bugs and having bad behaviour. To combat this problem we determined that every player should be able to report other users in our application. Each player is only able to report the same user once. Then, the administrator is able to check the list of users with most reports.

Ban/Unban a player

Completing the previous feature, administrators are able to ban players permanently from our platform. In the admin dashboard, there exists a section where admins are able to check the list of users with most reports and are able to decide if they want to ban them or not. The login of banned players is blocked.

We also keep track of all banned players to solve situations where a ban is wrongly applied. In this situation, administrators are also able to remove the bans.

5.1.6. Account Privileges

Upgrade/Downgrade player privileges

As we have already said in the previous sections of the report, we established three levels of privileges in our platform which, in an ascendent order, are the following:

a. Normal account
b. Tournament privileges account
c. Admin account

In order to manage the privileges of each account we have created inside the admin dashboard a mechanism where the admin is able to upgrade and downgrade the privileges of each account. With this system, users are not able to execute operations that require a higher level of privilege than what they have in the hierarchy.

5.1.7. Statistics

Watch Player Statistics

In order to understand the progress of our application and the reception by the audience, we have developed inside the admin dashboard a section of statistics related to the players of our platform. Inside this section we are able to look at different tables and graphs that display information such as the number of new accounts in the system, the number of accounts banned and players with most reports.

Watch Games Statistics

To serve as a complement relatively to the previous feature, we have also implemented a section inside the admin dashboard with statistics regarding the different games that we offer to our users. Inside this section, some statistics like the number of matches played, the distribution of those matches between the different games and the number of players in each rank per game are presented.

5.1.8. Tournaments

Create and Delete Tournaments

The last group of features that were implemented in our application is related to the existence of tournaments inside our platform. Tournaments add more competitiveness between players. Institutions, schools and teachers are welcomed to use our application to host tournaments between their students and visitors. Tournaments can be public or private and can have different numbers of maximum users. The tournament creators are also able to delete their own tournaments if they haven't already started.

Search for tournaments

Similar to the functionality of searching for users, we have created a page with all active tournaments where users can search for a tournament to play. The page includes filters by tournament name, maximum capacity, privacy and tournament game. This page, as many others, have pagination implemented to keep the information easier to see and understand by our users.

Tournament and round initialization

In our tournament's execution, we decided to provide a higher level of management to the tournament creator, giving to him the responsibility of starting the tournament as well as each round of it. The tournament can start if at least 3 participants have already joined, not necessarily the full capacity. Regarding the rounds, they can only start when all the matches of the previous rounds have finished. All these verifications are done inside the server.

Kick Players from tournaments

Following the same line of thinking that tournament creators should have greater ability to manage them, we give them the possibility to kick players from they're tournaments being able to have control in the list of participants. To provide a higher level of control, we have also added the opportunity to create tournaments with passwords, which we call, private tournaments. The password is indicated by the creator, and he can share only with the players he desires.

Change tournament description

The last feature we have added to the range of tournament creator management options is the ability to change tournaments description. Inside the description, the owner of the tournament is able to write everything that the participants should know about the tournament. Things like who is the tournament organizer, how to contact him, which are the tournament rules and what is the predict for the start of each round of the tournament might be added to the description by the tournament owner.

Play tournament matches

We cannot have tournaments without matches. Considering this, we also implemented a way to play tournament matches. These matches can be accessed directly by the tournament page. Each participant has a button to perform his check in for the next game. When both players execute their check-ins they are redirected to the game page. After the game, all information regarding the tournament is updated automatically.

View tournament bracket

To finish, we have created a separate page in each tournament where the players can access to get a full view of the tournament's bracket. With this mechanism, players are able to know who are their possible future opponents as well as detect the stage the tournament is in.

5.2. Planned but Not Implemented Features

Chat with default messages

In the beginning, during our requirement analysis phase, we decided to include in our possible features the existence of an ingame chat with default messages to avoid toxicity between players, especially in this age. Although this feature was not implemented due to the lack of time, we pretend to include it in the future work section as we will see next.

Bigger number of games

At the start, one of our goals was to implement the highest number of games in our application. However, after some time, also with the advice of our teachers we realised that it was more important to develop all other functionalities on the application, since now, with everything done, adding new games to the application only require the development of their engines.

5.3. Added Features

Customized Avatar

MathGames is always trying to find ways to please their users. In our opinion, this is how we can retain users and make them return to our platform. Considering this, we developed a customized avatar. The rotating 3D avatar can be customized by each user selecting different parts to equip it. Some of those pieces are available since the level 1 and others are unlocked accordingly with the progress of the user in our application. A new item is unlocked at every 5 levels of experience.

Notifications Box

We have implemented a notification system where the users are notified regarding some operations on our application. The notification box shows every active notification that the logged user has. Then, the user is able to remove the notifications and respond to them (when they require an answer).

For now, the list of notifications is the following:

1. A user has invited you to be his friend
2. A friend has invited you to play
3. A new round from the tournament (which you are participating in) has started

1.1 Test Results

Given that our application is intended to be used by children and adolescents doing usability tests plays a big part in the evaluation of the quality of our system. Therefore, we shared our application with a sizeable number of people. The users were then asked to try all the existing features with minimal guidelines in order to ascertain the intuitiveness and complexity of our system. After trying our system the users were then given a form containing the standard System usability scale questions. The results of this form can be observed below ().

Figure 74 - User Usage Frequency Graph

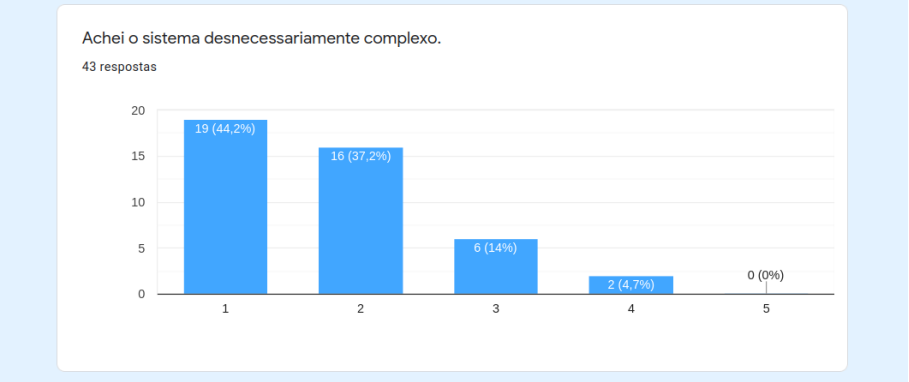


Figure 75 - Unnecessary Complexity Graph

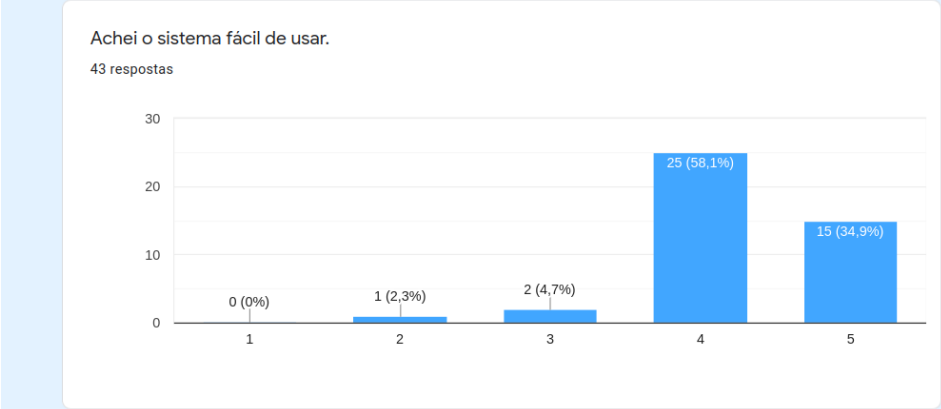


Figure 76 - How Easy is the system to Use

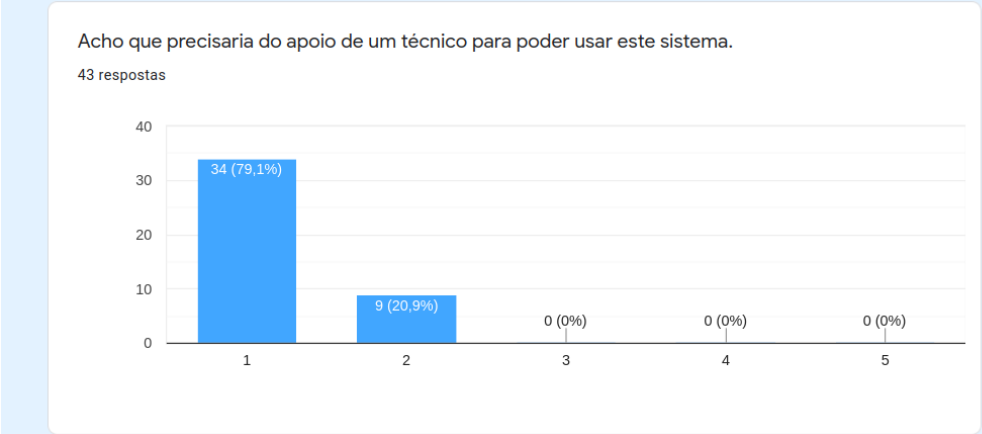


Figure 77 - Need for Technical Help Graph

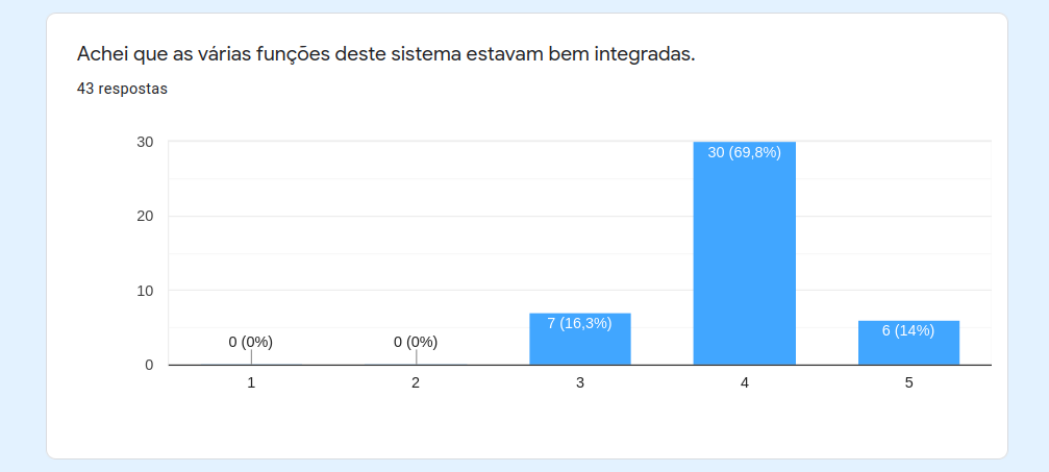


Figure 78 - Well Integrated System Methods Graph

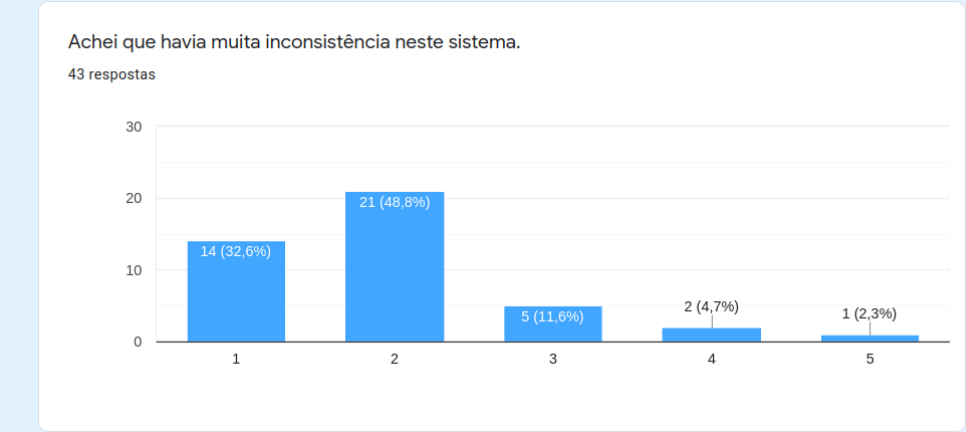


Figure 79 - System Inconsistency Graph

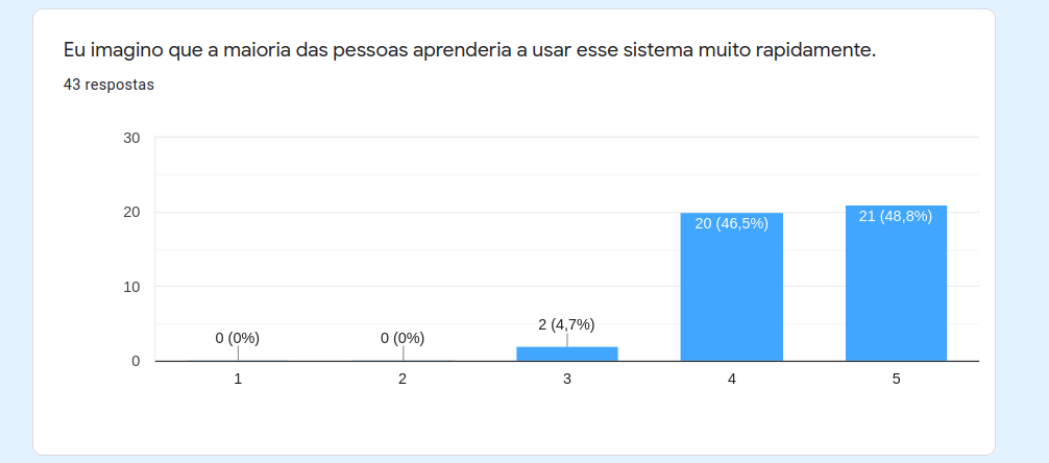


Figure 80 - Easeability to Learn Graph

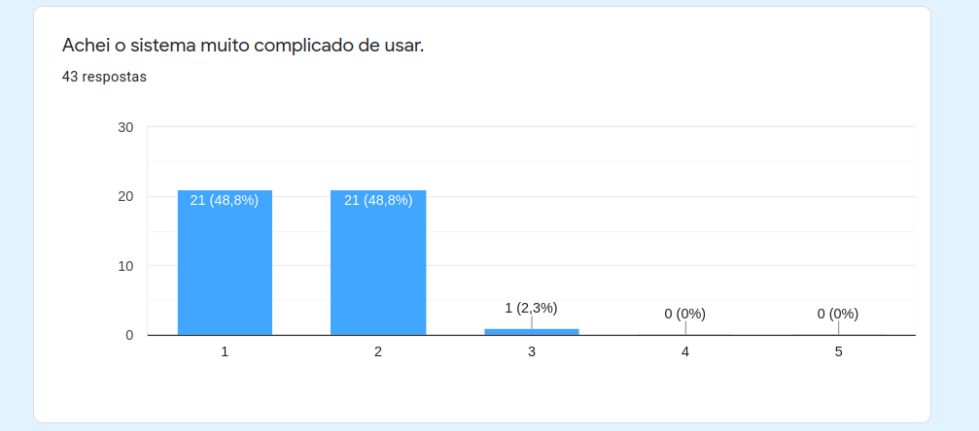


Figure 81 - System too Complicated Graph

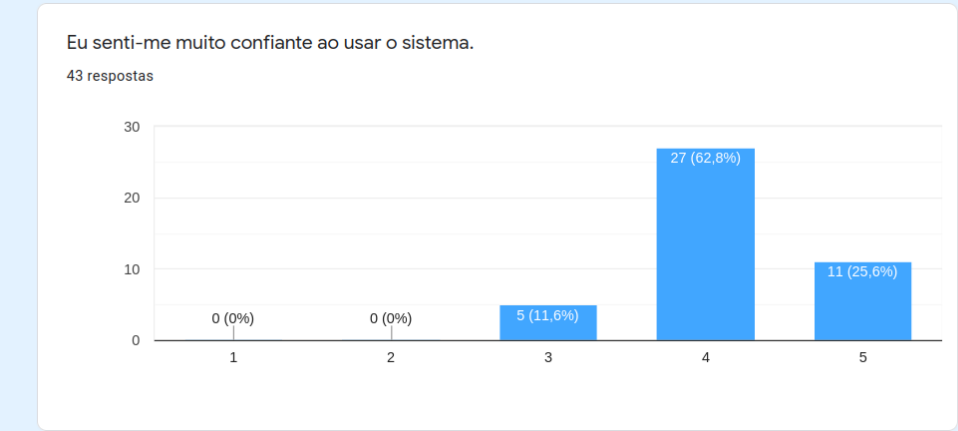


Figure 82 - User Confidence in using the System

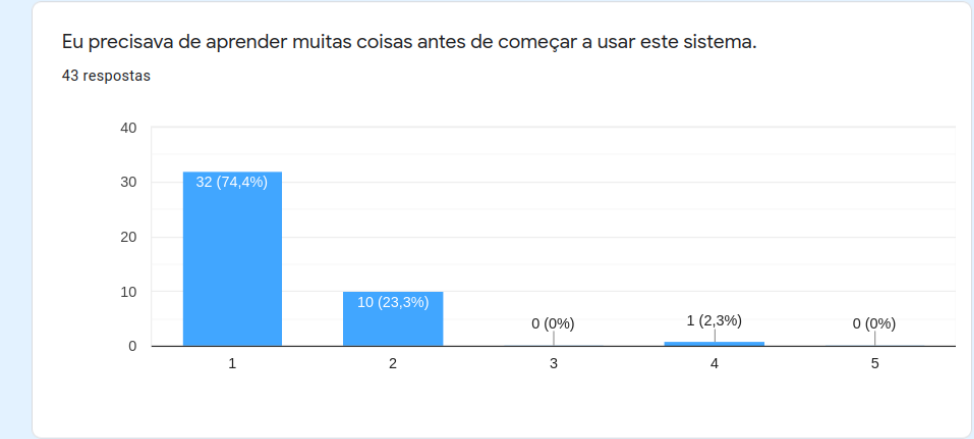


Figure 83 - User would need to learn a lot of aspects before using the System

As we can see above the results were mostly positive given that in all questions the best and second best options were the most picked of the five. In order to fully use the System usability scale, its score was calculated resulting in 69,6 which is above the recommended 68 score.

Chapter 6. Backlog Management and CI/CD Pipelines

To manage our backlog, the tool Jira was used. The backlog was divided into sprints and in each sprint a set of user stories and tasks was selected to be implemented. These items were initially placed in the “TO DO” column and then in the “IN PROGRESS” column once someone started working on it. Then once it was finished it was either placed in “TESTING” while it was being tested or directly in the “DONE” column if no tests were required.

To organize the repository the gitflow workflow was adopted having a master branch with a stable release, a release branch where the result of each sprint could be tested before going to the master, develop where we had an in-development version and feature branches where each member could take care of a feature, bug fixes and documentation independently. Therefore, we have also used a merge request policy to merge the branches. In these merge requests, we introduce a description, assign workers and reviewers and add a label to the merge. Then, the reviewers check the code and accept the merge if everything is fine.

In addition, a Continuous Integration and Delivery pipeline was implemented using Gitlab CI. This pipeline runs on Gitlab runners hosted in a Ubuntu VM on DigitalOcean. In terms of CI, the web application was compiled failing if there was any bug or warning. Then, in terms of CD, if it was running on the branch develop it will proceed to build a docker container image for the server and a docker container image for the web application pushing them to the Gitlab Container Registry. Then these images were deployed on the DigitalOcean VM alongside a MySQL docker image.

Chapter 7. Documentation

In this project, some documentation was made in mkdocs mainly to inform future developers on how to run this project in development mode. We also integrated swagger in our node server providing a rich documentation for each route (Figure 84).

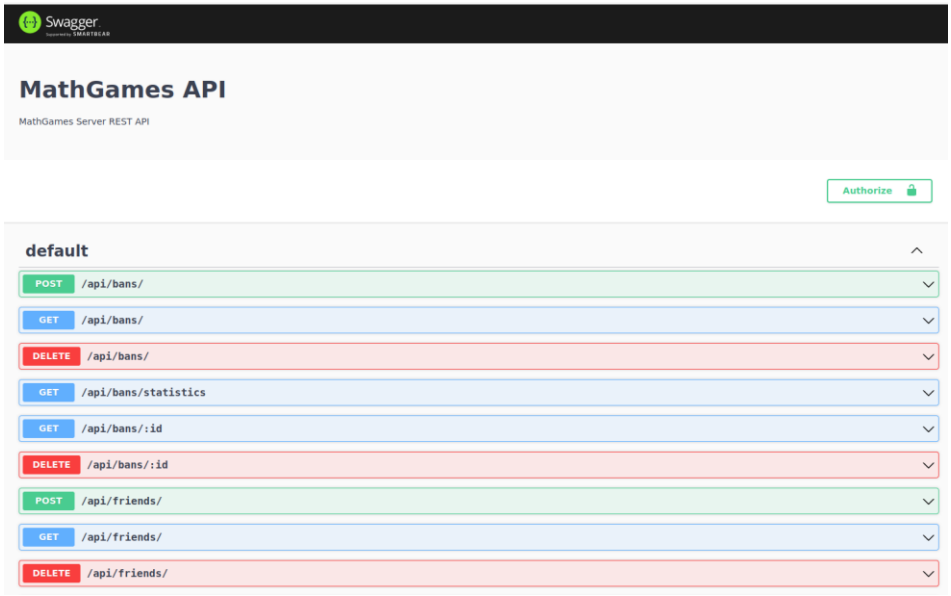


Figure 84 - API Endpoints

Each route is associated with a comment section that describes some route aspects like the http method, a description, the path and the possible response codes (Figure 85). With this practice we were able to maintain a complex api well documented.

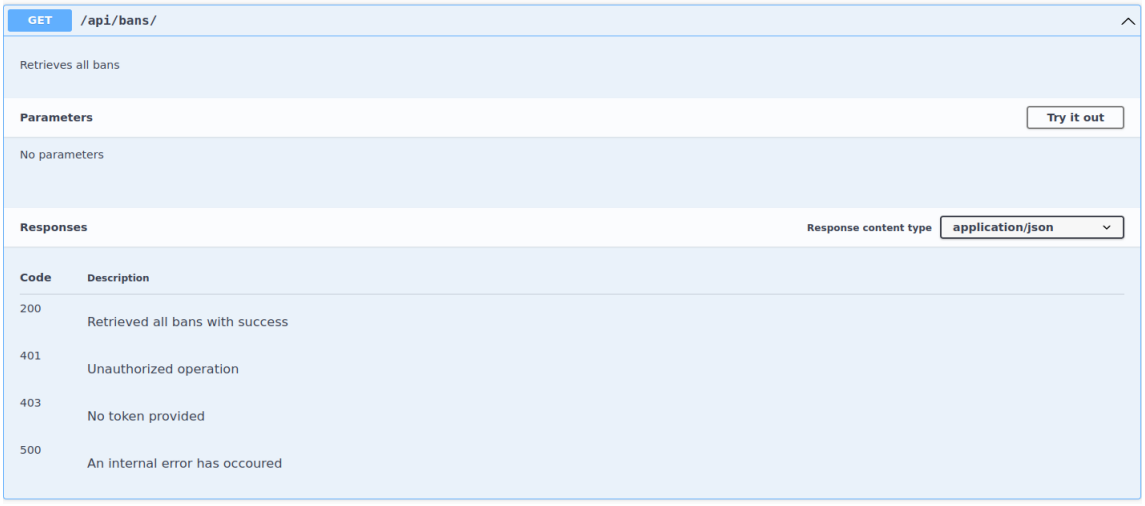


Figure 85 - Endpoint Description

Finally, we have also added swagger-ui dependency and configured it to provide us an interface where we can test our routes performance.

Note: To get the full documentation of our API, we just launch the server and access /api-docs endpoint. We can also check the documentation and update it by accessing our code and checking every comment section that is above each route.

Chapter 8. Future Work

To conclude our work, we have enumerated a list of useful upgrades to our application, which if our team would keep working on this project in the future we would implement. Some of these ideias emerged during the reunions that we have made during the project development and have not been implemented due to the lack of time. Follows the list:

- Implement the rest of the math board games to increase the range of games that we have to offer to our users
- Implement a chat with default messages in the match room to allow players to communicate between themselves with restrictions.
- Implement different formats of tournaments, for example, double elimination bracket tournaments. This would give more freedom to tournament creators in the management of their tournament formats.
- Implement the possibility for the players to choose if they want to play with or without a timer and, in case of playing with a timer, change the timer duration. For example, have games with 5, 10 or even 15 minutes.
- Change the matchmaking system to incorporate the rankings of the players. Currently, the opponent rank is only taken into consideration when we calculate the ranking points won and lost by players. In addition, the players should play with opponents with similar ranking. This feature would avoid huge skill gaps between opponents in the game.
- Upgrade the tournament winner rewards. Apart from the extra level points earned, some pieces from the avatar could be unlocked only with tournament wins. This would give an extra competitive component to the tournaments forcing players to get better.
- Implement an spectator mode. Users would be allowed to join friends' matches to watch them play.

Chapter 9. Conclusion

We are very glad that we managed to develop a project with this dimension and we are pleased with the final result. We consider that we have produced a nice work where we were able to implement all the requirements that were presented. Although one of the planned features was not implemented and the number of different games developed could also be higher, as we have mentioned before, the reasons behind it are plausible. We were also able to incorporate some additional functionalities to our work that were not planned in the early stages of the project, giving a positive answer to the changes on our customer requirements.

We have also developed a detailed report with all the information required to better understand the efforts that were applied in the project elaboration. To conclude, another important component to maintain and evolve the software was produced, the documentation where we tried to describe every piece of code in the project.

Regarding our individual evolution, the project was a great challenge for us, as the number of technologies we used with which we had not had any contact before was high. Therefore, learn all those technologies, reinforce the knowledge related to Continuous Integration, Continuous Delivery and backlog management, work with game engines, use our artificial intelligence experience, develop a mobile application for the first time and work with 3D dimensional avatares in a project with considerable dimension have increased our individual skills and knowledge.

References