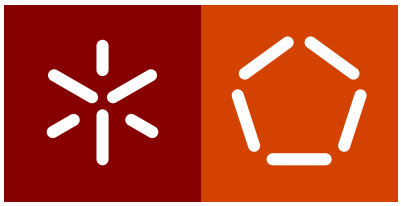


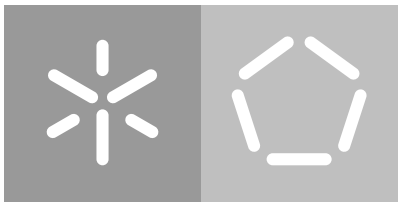
Universidade do Minho
Escola de Engenharia
Departamento de Informática

João da Cunha e Costa

Hypatiamat: A funcionalidade de multijogador em jogos online

Dissertação de Mestrado

Novembro 2022



Universidade do Minho
Escola de Engenharia
Departamento de Informática

João da Cunha e Costa

Hypatiamat: A funcionalidade de multijogador em jogos online

Dissertação de Mestrado

Dissertação de Mestrado
Mestrado Integrado em Engenharia Informática

Dissertação realizada sob a orientação de
José Carlos Leite Ramalho
Ricardo Manuel Neves Pinto

Novembro 2022

DECLARAÇÃO DE INTEGRIDADE

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração. Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

João da Cunha e Costa

RESUMO

O desempenho escolar na área da Matemática é uma preocupação crescente junto da comunidade educativa, tendo em conta o elevado insucesso escolar e correspondente abandono escolar precoce. Os professores tentam perceber qual a melhor forma para captar a atenção dos alunos ou incutir um maior interesse nestes relativamente a esta disciplina. Para além das técnicas utilizadas no ensino da matéria, os professores começam a aderir cada vez mais às novas tecnologias que auxiliam o ensino, uma vez que, estas permitem captar uma maior atenção por parte das crianças e jovens que, desde cedo, estão familiarizadas com as tecnologias atuais.

O *Hypatiamat* é um projeto, orientado mais em específico para alunos do 1º ao 9º ano, cujo principal objetivo passa por despertar, junto dos alunos, o gosto pela Matemática e consequentemente aumentar o aproveitamento escolar nesta disciplina.

Para este efeito, esta plataforma fornece diversos conteúdos hipermédia como explicações, resumos, aplicações, jogos, etc. Para além disso, esta plataforma realiza periodicamente diversos campeonatos com alguns dos seus jogos para milhares de alunos. No entanto, estes pecam por não possuírem a opção de multijogador online, o que dificulta a sua realização e impossibilita que os utilizadores possam jogar entre si os diversos jogos da plataforma de forma remota.

Deste modo, esta dissertação centra-se na implementação deste modo multijogador online num dos jogos do Hypatiamat, para futuramente servir como modelo de um guião a ser implementado nos restantes jogos que a plataforma Hypatiamat possui.

Palavras-Chave: Hypatiamat, Matemática, Jogos, Multijogador Online.

ABSTRACT

School performance in Mathematics is a growing concern within the educational community: everyone is worried about the school failure and the corresponding early school leaving. Teachers are concerned with questions about what is the best way to capture the students' attention in this subject. Moreover, in addition to the techniques they already use, teachers are increasingly embracing new technologies that aid in teaching, as they allow them to capture more attention from children and young people who are familiar with technology from an early age.

Hypatiamat is a project that aims to awaken in students a taste for Mathematics and consequently increase school success in this subject, being oriented more specifically to students from 1st to 9th grade. For this purpose, this platform provides several hypermedia contents such as explanations, summaries, applications, games, etc. In addition, this platform periodically holds several championships with some of its games for thousands of students. However, these do not currently support any online multiplayer functionalities, which makes it difficult to hold these championships and makes it impossible for users to play the platform's various games remotely.

Having said this, this dissertation focuses on the implementation of this online multiplayer mode in one of the *Hypatiamat* games, with the purpose of serving as a model of a guide to be followed in the future in the implementation of these same functionalities in the other games that the *Hypatiamat* platform has.

Keywords: Hypatiamat, Math, Games, Online Multiplayer.

CONTEÚDO

LISTA DE FIGURAS

LISTA DE EXEMPLOS

ACRÓNIMOS E GLOSSÁRIO

A

API Application Programming Interface - Interface ou protocolo de comunicação entre um cliente e um servidor.

B

BACKEND Referente ao servidor e base de dados de uma aplicação.

C

CRUD Create, Read, Update, Delete.

F

FRONTEND Referente à interface de um *software* ou *website*.

H

HTML HyperText Markup Language.

I

ID Identificador.

J

JSON JavaScript Object Notation.

M

MYSQL Sistema de gestão de base de dados relacional, que utiliza a linguagem SQL como base.

T

TCP Transmission Control Protocol - Protocolo de comunicação que garante a entrega da totalidade dos dados na sua ordem correta.

1

INTRODUÇÃO

No presente capítulo será apresentado o contexto em que se insere esta dissertação, bem como a motivação por detrás da mesma. Numa fase final, serão apresentados os objetivos da presente dissertação, a metodologia utilizada e um resumo da estrutura do documento.

1.1 CONTEXTO

É notório, nos dias de hoje, os desafios que a sociedade enfrenta tendo em conta a constante globalização e o desenvolvimento tecnológico em aceleração. Torna-se assim necessário formar os jovens adequadamente nas mais variadas valências, nomeadamente no contexto do ensino da matemática, fundamental para o seu desenvolvimento.

Neste sentido, a escola assume um papel fundamental, na medida em que é um local privilegiado para aliar a matemática ao desenvolvimento tecnológico que presenciamos diariamente, preparando assim os alunos para o futuro. preparando assim os alunos para o futuro.

No entanto, a Matemática é uma das disciplinas onde os alunos apresentam maiores dificuldades, sendo esta uma preocupação maior por parte da comunidade educativa.

Aliado à intensificação da informatização do setor educativo e à crescente procura por ferramentas educativas *online*, estes foram alguns dos fatores que levaram à criação do Hypatiamat.

Esta Plataforma surgiu com o objetivo de ajudar os professores de Matemática do ensino básico a incutir nos seus alunos o gosto por matemática, através de jogos e de aplicações de conteúdo, que a própria plataforma disponibiliza. Deste modo, os alunos poder-se-ão sentir mais motivados na aquisição e prática dos conhecimentos envolvidos nas matérias lecionadas ao longo do ano letivo, uma vez que nasceram na era da tecnologia e veem a utilização desta com bons olhos e até com prazer.

Em suma, a Plataforma Hypatiamat pretende não só cativar os alunos e flexibilizar o estudo das matérias lecionadas ao ritmo de cada um, como também contribuir para que os estes sejam mais autónomos, tendo como finalidade aumentar sucesso escolar nesta disciplina.

1.2 MOTIVAÇÃO

A principal finalidade da Plataforma **Hypatiamat** é a promoção do sucesso na disciplina de matemática, combatendo o elevado insucesso escolar e correspondente abandono escolar precoce. Para isso, a plataforma do Hypatiamat fornece diversos conteúdos hipermédia como explicações, resumos, aplicações, jogos de cálculo mental, etc. Além disso, esta plataforma realiza periodicamente diversos campeonatos com alguns dos seus jogos para milhares de alunos. No entanto, existe uma condicionante nos jogos utilizados, uma vez que para um jogador poder jogar contra outro é necessário estarem a jogar no mesmo dispositivo, porque os jogos não têm a funcionalidade de multijogador *online*. Implementar esta funcionalidade facilitaria a realização destes campeonatos e permitiria aos alunos jogarem entre si remotamente, levando a um aumento do número de horas passadas a jogar, propiciando ainda mais a prática do cálculo mental. A crescente necessidade de ver o multijogador implementado foi o principal motivo que originou esta dissertação.

1.3 OBJETIVOS

O objetivo desta dissertação passou pela reformulação do **SAMduelJr** [5], um dos jogos do **Hypatiamat**[4], de forma a implementar as diversas funcionalidades necessárias ao multijogador.

Futuramente, esta dissertação tem a finalidade de servir como modelo de um guião a ser utilizado para implementar estas mesmas funcionalidades nos diversos jogos que a plataforma Hypatiamat possui.

Para alcançar os objetivos acima mencionados, é necessário:

- Desenvolver um sistema de autenticação dos utilizadores da plataforma dentro do **SAMduelJr**;
- Desenvolver um servidor para gerir a comunicação entre o jogo e os seus jogadores, assim como de alguma da lógica do jogo;
- Desenvolver a funcionalidade de multijogador com dois modos de jogo, um modo de fila de espera e um sistema de convite;
- Elaborar um sistema de ranking;
- Desenvolver uma *API* para o tratamento dos dados que estão associados ao sistema de ranking;
- Desenhar novas interfaces para suportar o paradigma de multijogador;

- Desenvolver um quadro com os cem melhores jogadores do ano letivo atual e desde sempre.

1.4 METODOLOGIA

A metodologia desta dissertação baseou-se em reuniões semanais com o orientador e co-orientador da dissertação, fundamentais para a sua realização, na medida em que são validados os requisitos pedidos pelo co-orientador e também conselhos sobre quais as ferramentas a utilizar. Numa segunda fase, a nova arquitetura aplicacional será colocada em ambiente de produção, paralelamente à existente. Desta forma, esta plataforma será testada e validada pelos utilizadores finais que irão fornecer input nas versões seguintes.

A metodologia de trabalho utilizada nesta dissertação baseou-se nos seguintes passos:

- Estudo sobre a estrutura e implementação da plataforma;
- Levantamento e análise de requisitos;
- Estudo das tecnologias necessárias à implementação do software;
- Implementação do software de acordo com os requisitos;
- Análise do desempenho e testes sobre o software desenvolvido;
- Reuniões periódicas com o orientador e coordenador do Hypatiamat.

1.5 ESTRUTURA DO DOCUMENTO

Neste capítulo introdutório, foi apresentado o contexto e motivação associados à origem deste trabalho, bem como o tema que este aborda e os objetivos a alcançar. O segundo capítulo começa pelo estudo de alguns casos de aplicações semelhantes ao Hypatiamat em contexto real, estudo do jogo SAMduelJr e respetiva análise das tecnologias necessárias ao desenvolvimento do jogo. De seguida, é apresentada a problemática da comunicação em tempo real, bem como algum do seu contexto teórico. Posteriormente, são apresentadas as funcionalidades desenvolvidas, juntamente com as interfaces implementadas. No quinto capítulo é apresentada a componente de *deployment*, onde se explica o processo de integração das funcionalidades desenvolvidas na plataforma Hypatiamt. Por fim, serão apresentadas algumas conclusões sobre o trabalho realizado.

2

ESTADO DA ARTE

Neste capitulo serão apresentadas, de forma sucinta, algumas plataformas que recorrem à tecnologia para promoção do ensino e aprendizagem da matemática no Ensino Básico. Adicionalmente, será explicado em que consiste o jogo **SAMDuelJr**, elaborando também a sua estrutura inicial. Por fim, será realizado o enquadramento das várias tecnologias e materiais utilizados.

2.1 PLATAFORMAS DE APOIO AO ENSINO DA MATEMÁTICA

A plataforma **Hypatiamat** disponibiliza uma grande variedade de conteúdos, tais como: aplicações interativas hipermédia (aplicações e jogos); fichas e recursos em pdf para utilização em contexto de sala de aula; guiões de exploração e leitura por parte do aluno, através do Mat-histórias; propostas de planificação e critérios para professores; propostas de utilização da Plataforma para auxiliar professores, alunos e encarregados de educação; um número extenso de questões, utilizadas para praticar vários conteúdos da disciplina; e outros materiais para estudantes e professores do Ensino básico, sendo o objetivo principal da plataforma encorajar a excelência e o crescimento do sucesso no tópico da matemática (do 1º ao 9º ano).

Esta plataforma é diretamente dirigida aos estudantes, fazendo uso da sua afinidade por ambientes eletrónicos. Para tal, oferece aos alunos, professores e encarregados de educação recursos que lhes permitem incorporar, nas suas práticas diárias na sala de aula ou em casa, técnicas que utilizam este tipo de ambientes.

Por sua vez, todo o progresso realizado por alunos e professores é constantemente monitorizado pela equipa Hypatiamat através de um *backoffice*. Este *backoffice* permite guardar toda a atividade desenvolvida na plataforma, permitindo aos professores acompanhar o desempenho dos seus alunos em tempo real.

O Hypatiamat organiza também diversos campeonatos de cálculo mental, recorrendo à competição para fomentar o desenvolvimento das competências a nível do cálculo mental. Estes campeonatos têm imenso sucesso, sendo que já se realizaram mais de uma dezena de eventos, com a participação de milhares de alunos de norte a sul do país.

Para além do Hypatiamat, existem outros projetos educativos no panorama nacional português. Uma das mais utilizadas é a **Escola Virtual**, sendo uma plataforma educativa digital que se foca desde o ensino Pré-escolar ao 12.º ano, oferecendo recursos e ferramentas de apoio ao estudo e aprendizagem, para os alunos e para os professores, que disponibiliza ferramentas para o planeamento das aulas, como testes interativos, relatórios de desempenho dos alunos, criação de turmas/grupos personalizados e sugestões de conteúdos.

2.2 SAMDUELJR

Como explicado anteriormente, o **Hypatiamat** fornece diversos jogos online de cálculo mental. No entanto, estes apenas permitem jogar em modo offline ou se dois jogadores estiverem no mesmo dispositivo fisicamente. Neste sentido, surgiu a necessidade de criação de um multijogador online implementado nestes jogos.

Como ponto de partida, surgiu esta dissertação, cujo objetivo foi implementar esta funciona- lidade, bem como todas as funcionalidades necessárias à implementação do mesmo num dos jogos da plataforma, que no caso será o jogo **SAMDuelJr**.

Com o iniciar desta dissertação, foi realizada uma análise à estrutura e funcionamento do jogo SAMduelJr. Verificou-se que este foi desenvolvido em **Adobe Animate**, contendo uma camada em *HTML* e outra em *JavaScript*. Através da figura abaixo é possivel visualizar a arquitetura inicial do jogo.

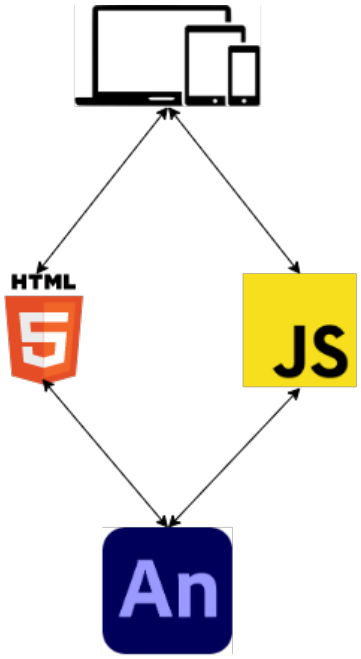


Figura 1: Arquitetura inicial do SAMDuelJr

O jogo do **SAMDuelJr** é um jogo de cálculo mental, focado nas operações aritméticas. No início de cada jogo, é apresentado um quadro a cada jogador com números inteiros de um a dez, um símbolo a representar a operação aritmética a utilizar e um resultado. Cada jogador tem de selecionar dois números que, aplicando a operação em questão, resulte no número apresentado na variável resultado. Se acertar nos números, estes desaparecem sendo apresentada uma nova operação e um novo resultado. Este processo termina quando não existirem mais números no quadro, sendo que o primeiro jogador a acabar com os números ganha. De referir também que, antes de cada jogo se iniciar, é possível escolher quantas rondas vão existir, sendo que é possível jogar à melhor de 3,5,7,9 ou sem limite de rondas. Na figura abaixo é possível visualizar a interface do jogo.

Figura 2: Interface do jogo contra computador

O SAMDuelJr já tinha uma série de funcionalidades desenvolvidas, nomeadamente, uma secção de explicação das instruções do jogo, um modo de jogo usado para treinar e um para jogar contra o computador.

Na secção das instruções é explicado em que consiste o jogo. É apresentada a forma de como se deve jogar e de que forma é possível vencer uma partida. Além disso, é apresentada uma animação com um exemplo do tabuleiro do jogo, para facilitar a compreensão de como se deve jogar. A figura seguinte representa a interface desta secção.

Figura 3: Interface do menu de instruções.

O modo de jogo de treino existe para permitir aos jogadores aperfeiçoarem o seu nível de jogo e para facilitar a compreensão do funcionamento do mesmo. Neste modo, é possível escolher quais operações aritméticas se pretende treinar. Adicionalmente, no eventual caso em que um jogador não saiba que jogada fazer, existe uma opção que fornece pistas para ajudar. Na figura seguinte podemos ver a interface relativa a este modo de jogo.

Figura 4: Interface do modo de jogo de treino.

Por fim, existe o modo de jogo onde um jogador joga contra o computador. O jogador tem a opção de escolher quais operações aritméticas serão geradas e também pode selecionar quantas rondas serão jogadas, podendo ser a partida à melhor de 3,5,7,9 ou até mesmo sem fim.

Figura 5: Interface do modo de jogo contra o computador.

2.3 ESTUDO DAS TECNOLOGIAS

Nesta secção será explicado em que consiste o jogo **SAMDuelJr** e a sua estrutura inicial. Adicionalmente, será realizado o enquadramento das tecnologias utilizadas de forma a alcançar o objetivo proposto nesta dissertação, comparando-as com as suas alternativas mais diretas.

2.3.1 *Node.Js*

Node.js [10] é um ambiente de *runtime* open-source multiplataforma de **Javascript**. Este tem uma natureza assíncrona baseada em eventos, apresentando uma arquitetura *I/O single-threaded* não bloqueante, o que o torna eficiente e adequado para aplicações em tempo real. Tem como objetivo ser utilizado para criar aplicações *server-sided* rápidas e escaláveis, conseguindo responder simultaneamente a diversos pedidos.

O Node.Js usa uma arquitetura *"Single Threaded Event Loop"* para gerir vários clientes em simultâneo. É necessário compreender como os clientes *Multi-Threaded* concorrentes são tratados em linguagens como *Java*, de modo a entender a diferença entre outros *runtimes*. Numa arquitetura de resposta *Multi-Threaded*, múltiplos clientes enviam pedidos e o servidor processa cada um deles individualmente antes de enviar a resposta de volta. No entanto, para tratar pedidos simultaneamente, é necessário usar diversas *Threads*. Sempre que um pedido é recebido, uma das *Threads* presente no conjunto das *Threads* disponíveis é responsável por tratar desse pedido.

O Node.Js funciona de forma diferente. Sempre que chega um pedido, ele coloca-o numa fila de espera, onde de seguida existe um ciclo *Single-Threaded* sempre à espera de pedidos, quando chega um, o ciclo retira-o da fila e verifica se o pedido tem operações bloqueantes ou não, se não tiver, este é processado e é enviada a resposta de volta. Se o pedido tiver operações bloqueantes, o ciclo designa uma *Thread* de uma *"Thread Pool"* auxiliar limitada(*"Worker Thread Pool"*) para o processar. Assim que a operação bloqueante estiver processada, o ciclo volta a inserir este pedido na fila de espera, mantendo assim uma natureza não bloqueante em toda a sua execução.

Na imagem seguinte podemos ver uma ilustração deste processo:

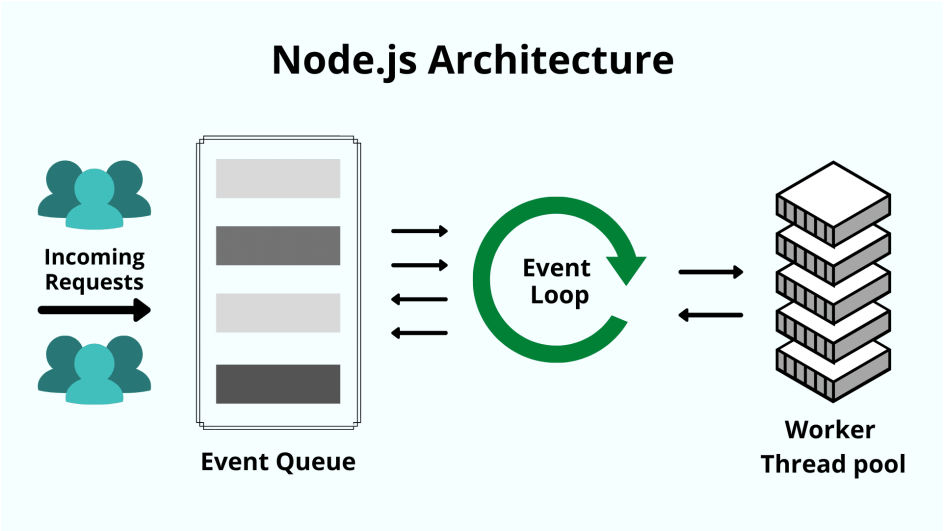


Figura 6: Arquitetura do Node.Js

O Node.Js tem vindo a ganhar cada vez mais seguidores graças a algumas das suas características:

- **Escalabilidade**: Sendo *Single-Threaded*, é capaz de lidar com um enorme número de pedidos simultâneos com elevado rendimento;
- **Velocidade**: Uma arquitetura não bloqueante torna a execução mais rápida e eficiente;
- **Multiplataforma** : O facto de ser multiplataforma permite criar *Websites*, aplicações desktop, e mesmo aplicações móveis, tudo utilizando o Node.js;
- **Fácil manutenção**: É uma escolha fácil para os desenvolvedores, uma vez que tanto o *frontend* como o *backend* podem ser geridos com *JavaScript*;
- **Fácil aprendizagem**: É fácil começar a usar Node.Js uma vez que é suportado por uma grande comunidade que fornece muito material de apoio.

Uma vez que o Node.js utiliza menos recursos/memória, verifica-se uma execução mais rápida das tarefas. Quando é necessário processar tarefas de dados intensas faz mais sentido a utilização de linguagens *Multi-Threaded* como *Java* mas, para aplicações em tempo real, o Node.js é a escolha a seguir.

Dessa forma, dada a necessidade que existe de uma comunicação em tempo real para a jogabilidade do SAMDuelJr, foi esta a ferramenta proposta para desenvolver o servidor de gestão da comunicação do jogo.

2.3.2 *Express*

O **Express.Js** [2] é uma framework grátis e *open-source* de Node.Js. É utilizada para desenvolver aplicações *web* e API's de forma fácil e eficaz. Uma vez que o Express.js requer apenas *Javascript*, torna-se mais fácil construir aplicações *web* e API's sem muito esforço. Adicionalmente, esta *framework* fornece um grande leque de funções e *middleware* que agilizam a construção das aplicações e API's, sem esta, é necessário, por exemplo, escrever o código da componente de encaminhamento dos pedidos *HTTP*, que se torna uma tarefa demorada e desnecessária. Dito isto, o Express.Js oferece simplicidade, flexibilidade, eficiência, minimalismo e escalabilidade aos programadores, tendo também a vantagem de um desempenho poderoso, uma vez que é uma *framework* do Node.js.

A elevada performance e facilidade de aprendizagem do Node.js e a facilidade com que o Express.Js permite criar uma API, são os principais motivos que levaram à escolha destas duas tecnologias para desenvolver a API que suporta o sistema de ranking do **SAMDuelJr**.

2.3.3 *Socket.IO*

O **Socket.IO** [13] é uma biblioteca *JavaScript* construída em cima do protocolo **WebSocket** 3.4, com o objetivo de estabelecer um canal de comunicação bidirecional em tempo real entre um cliente e um servidor. O Socket.IO é constituído por duas partes, o lado do cliente que corre no *browser* e o lado do servidor que corre em Node.Js.

De seguida, são enumeradas algumas características do Socket.IO comparativamente com simples WebSockets, nomeadamente:

- **HTTP long-polling fallback**: Por omissão, Socket.IO utiliza o protocolo WebSocket se este for suportado pelo navegador. No caso dos navegadores mais antigos, como o *IE9*, não suportam WebSockets. Nesse caso, o Socket.IO optará por utilizar outras tecnologias, tais como **HTTP long-polling**. Esta técnica consiste no envio periódico de pedidos ao servidor. Primeiramente, é enviado um pedido ao servidor e estabelece-se uma ligação. Esta conexão permanece aberta até que o servidor tenha informação para enviar de volta para o cliente. Após enviar a informação, a ligação é fechada e o cliente faz outro pedido, repetindo o processo.

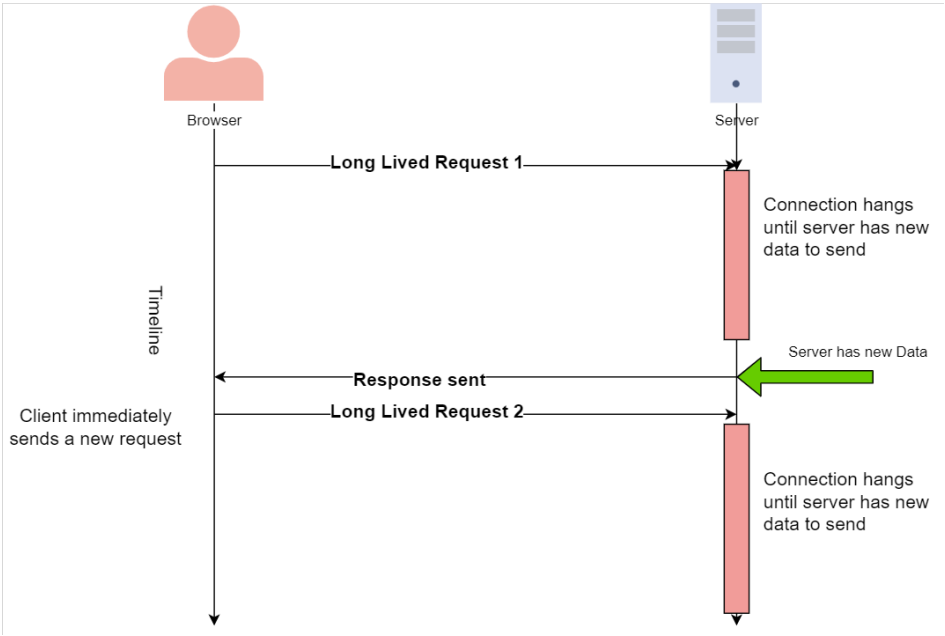


Figura 7: HTTP Polling

- **Broadcasting**: Ao contrário de WebSockets, o Socket.IO permite transmitir uma mensagem a todos os utilizadores conectados. Por exemplo, se estiver a escrever uma aplicação de chat e quiser notificar todos os utilizadores ligados que um novo utilizador aderiu ao chat, facilmente se pode transmitir essa mensagem de uma só vez. Utilizando WebSockets simples, é necessário guardar uma lista de todos os utilizadores ligados e depois enviar a mensagem diretamente, um a um;

- **Reconexão automática**: Se por algum motivo a conexão entre cliente e servidor for interrompida, o Socket.IO tem mecanismos que periodicamente verificam o estado da conexão, restabelecendo-a caso necessário;

- **Proxies e balanceadores de carga**: A utilização destas tecnologias tornam os WebSockets difíceis de implementar e de escalar. O Socket.IO suporta estas tecnologias, facilitando o seu uso;

- **Rooms**: O Socket.IO disponibiliza uma funcionalidade denominada de *Rooms*, a qual permite criar canais arbitrários, nos quais vários *sockets*/clientes podem entrar ou sair. Adicionalmente, é possível aliar a funcionalidade de *Broadcasting* acima descrita, com estas salas. É possível, portanto, transmitir a uma sala específica, ou todas, conseguindo comunicar assim com todos os *sockets*/clientes que a elas se juntaram. Isto é especialmente útil para notificar grupos de utilizadores específicos.

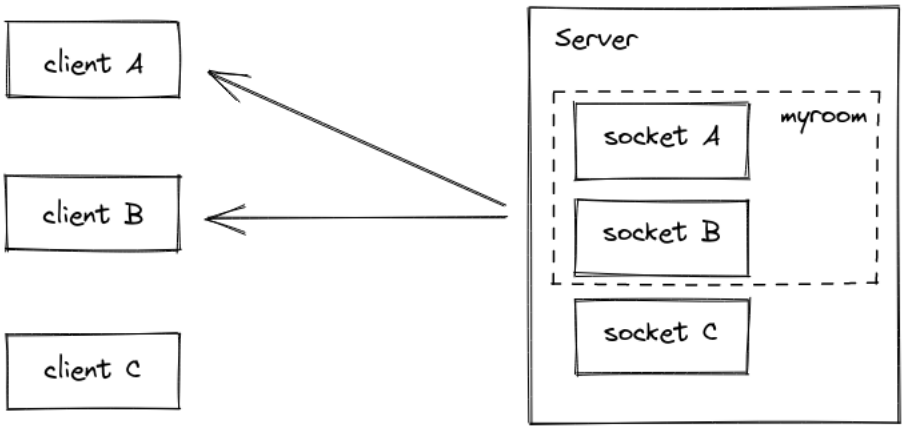


Figura 8: Conceito de Rooms

A figura seguinte permite visualizar,em resumo, as principais diferenças entre WebSockets e o Socket.IO.

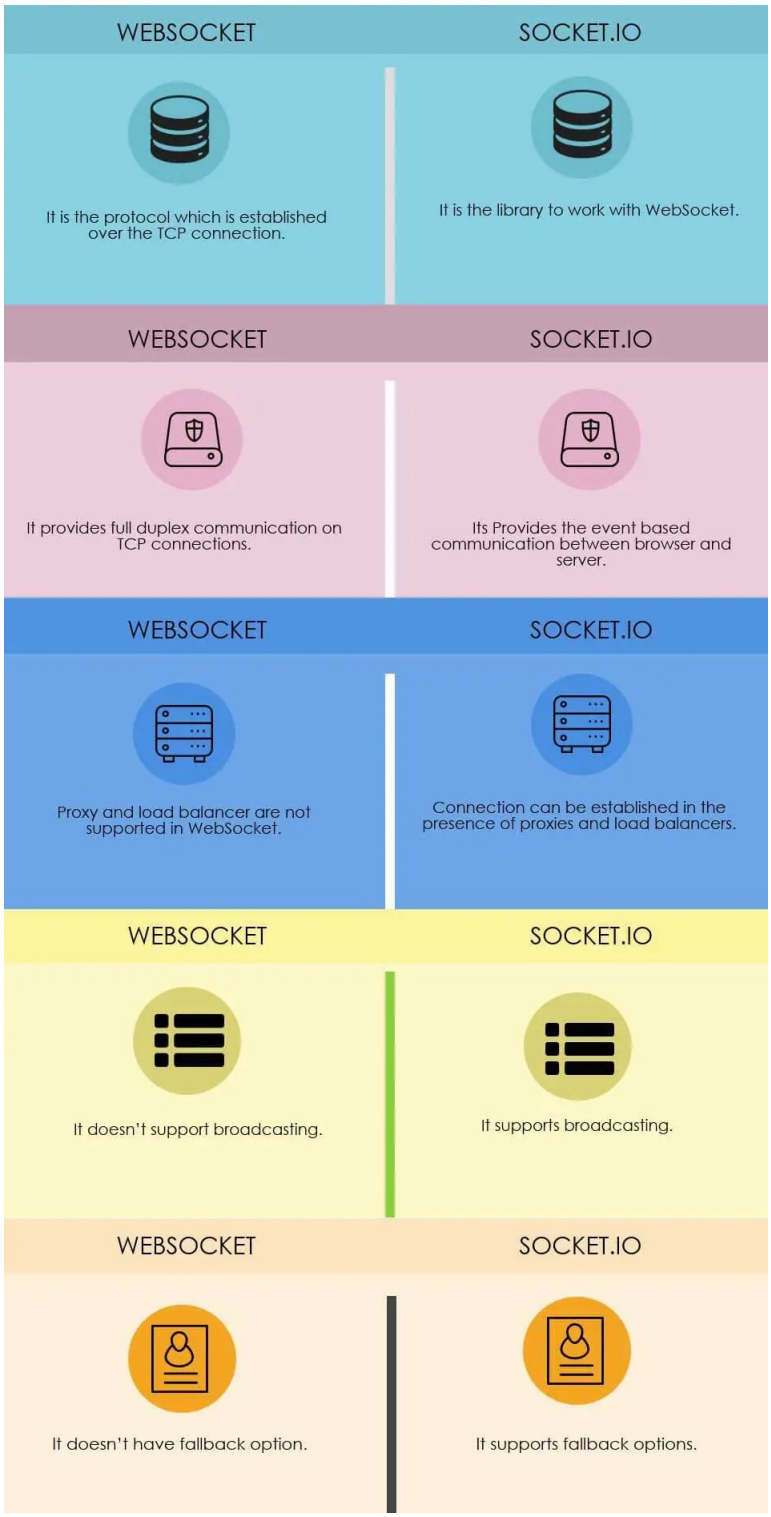


Figura 9: WebSockets Vs Socket.IO

2.3.4 *Identificadores Únicos Universais*

UUID's ou **Universally Unique Identifiers** são números de 128 *bits* usados para identificar informação dentro de um sistema informático.

Os UUID's são usados para identificar informação que necessita de ser única dentro do sistema. Estes identificadores são úteis, por exemplo, para chaves associativas em base de dados ou como identificadores de dispositivos físicos, devido à sua unicidade e baixa probabilidade de serem repetidos. Os UUID's têm a vantagem de não ser necessário serem emitidos por uma autoridade central e, em vez disso, estes podem ser gerados independentemente e consequentemente utilizados em todo o sistema sem levantar preocupações de que um identificador duplicado tenha sido gerado.

Empresas como a Apple, Microsoft, Samsung, entre outras, utilizam UUID's para identificar e monitorizar *hardware*, podendo estes serem definidos pela especificação da IETF [6] ou por uma variante privada.

De seguida, foi feita uma análise de alguns módulos de geração deste tipo de identificadores, sendo que os mais populares e utilizados, para **Node.Js**, são o **uuid** e o **NanoId**. Ao comparar mais detalhadamente ambos os módulos podemos verificar algumas diferenças entre eles, tais como:

- **Tamanho** - Ambos não têm dependências, mas ao contrário do uuid, o NanoId é cerca de 4 vezes mais compacto;
- **Performance** - NanoId é cerca de duas vezes mais rápido;
- **Segurança** - Em vez de usar o inseguro método *Math.random()*, NanoId usa os módulos *crypto* e *Web Crypto API*, sendo que estes usam geradores aleatórios e imprevisíveis com base em *hardware*. Adicionalmente, é usado um algoritmo melhorado de forma a garantir uma maior uniformidade na escolha dos símbolos para geração dos identificadores;
- **Alfabeto** - NanoId usa um alfabeto maior, o que resulta num identificador mais curto, com menos símbolos, otimizando os 36 símbolos do uuid para 21 símbolos. Além disso, o NanoId adiciona uma camada de parametrização, permitindo alterar o tamanho do identificador e também os carateres que pertencem ao alfabeto.

```
const { customAlphabet } = require('nanoid');
const alphabet = 'ABCDEF1234567890';
const nanoid = customAlphabet(alphabet, 10);
nanoid() //=> "7A2A589A0E"
```

Exemplo 2.1: Exemplo de como definir um alfabeto

No exemplo acima é definido um alfabeto com os carateres 'ABCDEF1234567890' e um Id com tamanho 10.

3

COMUNICAÇÃO EM TEMPO REAL

Com vista à implementação do multijogador, são necessárias algumas funcionalidades, tais como: um sistema de convite, em que um jogador convida outro para jogar e um sistema que permita emparelhar dois jogadores para estes poderem jogar entre si. Executar isto levanta um grande desafio: como estabelecer uma comunicação em tempo real entre os jogadores?

Em suma, é necessária uma forma de partilhar os dados globais do jogo entre todos os jogadores que estão ligados ao jogo e atualizar continuamente cada jogador sobre todos os outros. Existem diferentes técnicas que são normalmente utilizadas para o alcançar mas as duas abordagens mais comuns são a **peer-to-peer** e **client-server**.

3.1 ARQUITETURA PEER-TO-PEER

Uma forma simples de interligar os jogadores é através da arquitetura **peer-to-peer**. Embora o nome possa sugerir que apenas dois pares ("nós") estão envolvidos, por definição um sistema de rede peer-to-peer é um sistema em que dois ou mais “nós” estão ligados diretamente um ao outro sem um sistema centralizado a gerir a ligação ou a troca de informações. No caso de um jogo peer-to-peer, podemos ilustrar esta arquitetura com um simples jogo do Galo.

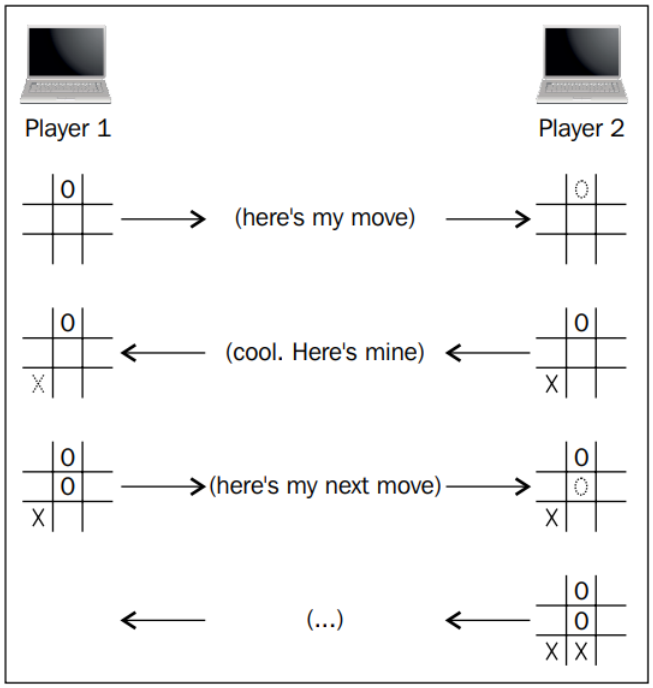


Figura 10: Jogo do Galo

Quando ambos os jogadores estabelecerem uma ligação entre si, quem iniciar o jogo faz uma jogada marcando um símbolo no tabuleiro de jogo. De seguida, a informação é transmitida para o outro jogador (*peer*), que agora está ciente da decisão tomada pelo seu oponente, podendo atualizar o seu próprio jogo. Assim que o segundo jogador receber o último estado de jogo resultante do último movimento do primeiro jogador, o segundo jogador é capaz de fazer a sua jogada. Esta informação é então copiada para o primeiro jogador que pode atualizar o seu próprio jogo e continuar o processo, fazendo a sua próxima jogada. O processo continua até que um dos "nós"se desligue ou o jogo termine.

Alguns dos benefícios de jogos com arquitetura peer-to-peer são:

- **Velocidade na transmissão dos dados**: Aqui, a informação vai diretamente para o nó pretendido. Noutras arquiteturas, a informação pode ir primeiro para algum nó central (ou o "servidor"), que depois repassa a informação para o nó pretendido;
- **Fiabilidade**: Neste caso, um nó que fique offline normalmente não afeta nenhum dos outros nós. No entanto, no caso simples de um jogo para dois jogadores, se um desses jogadores não puder continuar, o jogo provavelmente deixará de ser jogável. Imaginemos, no entanto, que o jogo em questão tem dezenas ou centenas de nós ligados. Se uma parte deles repentinamente perder a sua ligação à Internet, os outros podem continuar a jogar. No entanto, se houver um servidor que esteja a gerir a ligação de todos os nós e o servidor for abaixo, depois nenhum dos outros os jogadores saberão como comunicar entre si.

Por outro lado, esta arquitetura também possui algumas desvantagens:

- **Não se pode confiar nos dados recebidos**: Não se sabe ao certo se o remetente modificou os dados ou não. Os dados que são introduzidos num servidor de jogo também sofrerão do mesmo problema mas, assim que os dados forem validados e transmitidos a todos os outros pares, pode-se estar mais confiante de que os dados recebido por cada nó do servidor terão sido, pelo menos, tratados e verificados, sendo a informação mais credível;
- **Reduzida tolerância a falhas**: O argumento oposto foi feito na secção anterior: se jogadores suficientes partilharem o mundo do jogo, uma ou mais falhas não tornarão o jogo "injogável"para o resto dos "nós". Agora, se considerarmos os muitos casos em que qualquer um dos jogadores que saia repentinamente do jogo afete de forma negativa o resto dos jogadores, podemos constatar como um servidor poderia facilmente recuperar o jogo destas falhas. [12]

3.2 ARQUITETURA CLIENT-SERVER

A diferença mais óbvia entre esta arquitetura e a anterior é que esta, em vez de cada nó ser igual aos outros, um dos nós é especial. Ou seja, em vez de cada nó se ligar a todos os outros, neste caso cada nó (cliente) liga-se a um nó principal centralizado chamado servidor.

De forma resumida, o servidor está encarregue de fornecer dados e serviços a um ou mais clientes. No contexto do desenvolvimento de jogos, o cenário mais comum é quando dois ou mais clientes se ligam ao mesmo servidor; o servidor irá monitorizar o jogo assim como os jogadores associados. Assim, se dois jogadores tiverem de trocar informações que são apenas pertinentes aos dois, a comunicação passará do primeiro jogador para e através do servidor e terminará no segundo jogador.

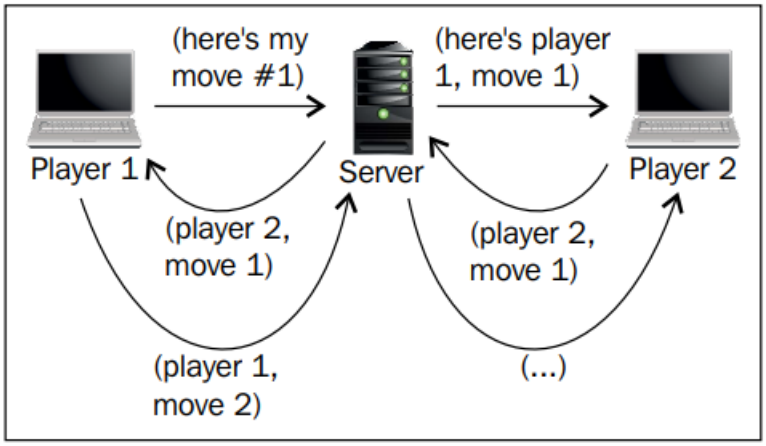


Figura 11: Interação client-server

Seguindo o exemplo do jogo do galo, que vimos na secção sobre a arquitetura peer-to-peer, podemos ver o quão semelhante é o fluxo de eventos num modelo cliente-servidor. Mais uma vez, a principal diferença é que os jogadores não têm conhecimento entre eles e só sabem o que o servidor lhes diz.

Alguns dos benefícios dos jogos com arquitetura client-server são os seguintes:

- **Separação de conceitos**: Para todos os que estão minimamente por dentro de desenvolvimento de software, sabe-se que produzir software organizado e estruturado é sempre um objetivo. A escrita de software com componentes individuais especializados permite-nos realizar uma tarefa de cada vez, tornando a nossa aplicação mais fácil de desenhar, codificar, testar e manter;
- **Centralização**: Ter um lugar central através do qual toda a comunicação deve fluir faz com que seja mais fácil gerir a mesma, aplicando quaisquer regras necessárias, controlando o acesso, e assim por diante;
- **Menos trabalho para o cliente**: Em vez de ter um cliente encarregue de receber input do utilizador, bem como de outros pares, validando toda a informação, partilhando dados entre outros pares, renderizando o jogo, e assim por diante, o cliente pode concentrar-se em fazer apenas algumas destas tarefas, permitindo que o servidor alivie algum deste trabalho.

Alguns inconvenientes desta arquitetura são:

- **A comunicação leva mais tempo a propagar**: No melhor cenário possível, em comparação com uma ligação peer-to-peer, todas as mensagens enviadas do primeiro para o segundo jogador demoraria o dobro do tempo a ser entregue. Ou seja, a mensagem seria enviada primeiro para o servidor e posteriormente do servidor para o segundo jogador;
- **Maior complexidade**: Enquanto grande parte do código pode ser reutilizado entre a componente do cliente e do servidor, no final do dia é necessário gerir um nível de complexidade maior.

3.3 PROTOCOLO HTTP

De forma a percebemos a necessidade de *WebSockets* no desenvolvimento de jogos, iremos abordar um protocolo bastante popular: **Hypertext Transfer Protocol (HTTP)**. Este é o protocolo presente na camada de aplicação que os navegadores *web* utilizam para ir buscar os seus dados a um Servidor *Web*. Apesar de o HTTP ser um protocolo fiável para recuperar documentos a partir de servidores *Web*, este não foi concebido para ser utilizado em jogos em tempo real; por conseguinte, não é ideal para este objetivo.

A forma como o HTTP funciona é muito simples: um cliente envia um pedido a um servidor, que depois devolve uma resposta ao cliente. A resposta inclui um código de conclusão, indicando ao cliente que o pedido ou está em processo, ou precisa de ser encaminhado para outro endereço, ou terminou com sucesso ou insucesso. Há várias questões a registar sobre o protocolo HTTP que deixarão claro que é necessário um melhor protocolo para a comunicação em tempo real entre o cliente e o servidor.

Em primeiro lugar, após a receção de cada resposta, a ligação é fechada. Assim, antes de fazer cada pedido, deve ser estabelecida uma nova ligação com o servidor. Na maioria das vezes, um pedido HTTP será enviado através de *TCP*, o que pode ser relativamente lento.

Em segundo lugar, HTTP é, por conceção, um protocolo *stateless*. Isto significa que, de cada vez que se solicita um recurso a um servidor, o servidor não tem ideia de quem somos e qual o contexto do pedido. Finalmente, o principal fator que torna o HTTP impróprio para a programação de um jogo multijogador é que a comunicação é unilateral ou seja, apenas o cliente se pode ligar ao servidor, e o servidor responde de volta através da mesma ligação.

A resposta a esta falta de funcionalidades do HTTP é bastante simples: os **Websockets**. Estes fornecem uma ponte de ligação que permite a comunicação bidirecional entre o cliente e o servidor. No capítulo seguinte segue-se uma melhor explicação desta tecnologia.

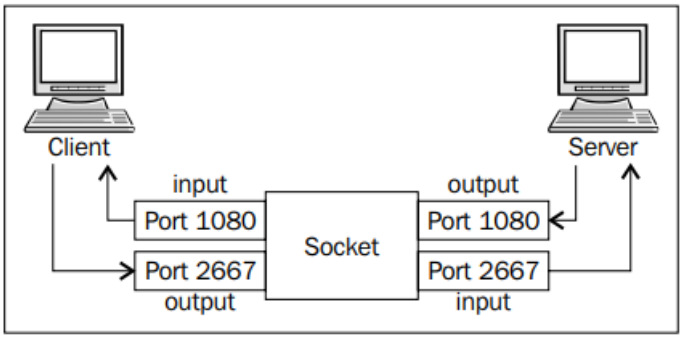


Figura 12: Rede socket

3.4 PROTOCOLO WEBSOCKET

Concluindo a discussão dos tópicos anteriormente mencionados, foquemo-nos na ferramenta que derivou do que foi falado anteriormente e que nos permite programar jogos multijogador *web* robustos: os **WebSockets**. WebSocket é um protocolo construído em cima do TCP, que permite às aplicações baseadas na Web ter comunicação bidirecional com um servidor. O objetivo desta tecnologia é fornecer a estas aplicações um mecanismo de comunicação bidirecional com os servidores que não dependa da abertura de múltiplas ligações HTTP (por exemplo, utilizando *XMLHttpRequest* ou *<iframe>s* e *long polling*).

Inicialmente, existe um 'aperto de mão' para estabelecer a ligação entre o cliente e o servidor (*browser* e servidor) que, uma vez estabelecida, permanece aberta e a comunicação é efetuada até que a ligação seja terminada do lado do cliente ou do servidor. A informação é transferida de forma bidirecional, não sendo necessário haver um pedido para existir esta troca de informação, sendo esta é a grande vantagem do protocolo WebSocket.

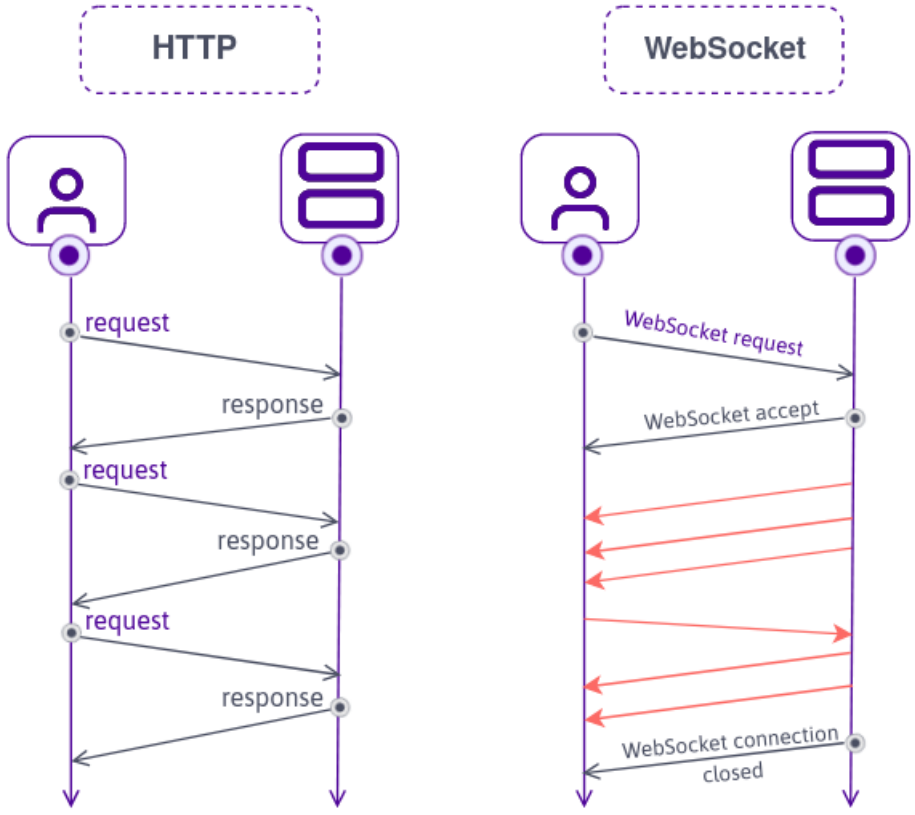


Figura 13: HTTP Vs WebSocket

Os WebSockets são maioritariamente usados para desenvolver aplicações *web* em tempo real. Estas aplicações permitem interagir com os clientes de forma quase instante instantânea,

daí que é essencial manter um canal de comunicação constantemente aberto com os clientes, com o intuito de os manter sempre atualizados. Como este protocolo permite estabelecer conexões entre os clientes e servidor, possibilitando uma troca de dados de forma bidirecional e de forma constante, este adequa-se ao desenvolvimento destas aplicações.

Alguns exemplos de aplicações que usam WebSockets são:

- **Redes sociais e serviços de mensagens**: Estas plataformas necessitam de estabelecer múltiplas conexões com os seus utilizadores, para transmitir e trocar mensagens em tempo real;
- **Jogos**: Em jogos multijogador de tempo real, o cliente e o servidor têm de enviar pacotes de informação em simultâneo. Isto só pode ser alcançado utilizando WebSockets, uma vez que o protocolo HTTP implica que é o cliente que tem de enviar um pedido primeiro;
- **Outras aplicações**: Outros exemplos de aplicações em tempo real que utilizam WebSockets são aplicações relacionadas com a bolsa de valores, criptomoedas e apostas desportivas.

Comparação HTTP vs WebSocket

WebSocket	**HTTP**
Protocolo Bidirecional. Tanto o cliente como o servidor podem trocar informação entre eles, sem necessidade de qualquer pedido prévio.	Protocolo unidirecional. O cliente inicia a ligação apenas para efetuar um pedido.
A ligação pode permanecer aberta por tempo indefinido.	A ligação não está aberta por muito tempo.
A ligação continua até ao momento em que o cliente/servidor decide terminá-la.	A ligação é terminada automaticamente assim que uma resposta é enviada.
Os pedidos frequentes não terão impacto na performance.	Pedidos frequentes deterioram o desempenho.
Maioritariamente utilizado por aplicações web em tempo real onde é necessária uma comunicação constante com o cliente.	Mais adequado para dados armazenados em cache, para tratar de dados estáticos ou para resolver cenários de erro.

Enquanto que o protocolo HTTP é mais adequado para tratar de dados que vão ser armazenados em cache e para recolher maiores quantidades de dados estáticos, este, devido à sua comunicação unidirecional, é menos adequado para aplicações em tempo real que necessitem de uma constante troca de informação entre o servidor e cliente. Dito isto, os WebSockets, com a sua natureza bidirecional, tornam-se a melhor opção para o desenvolvimento do multijogador do **SAMDuelJr**.

Em suma, neste capítulo abordou-se o contexto teórico da problemática da comunicação em tempo real, sendo apresentadas as arquiteturas e tecnologias associadas a esta. No capítulo seguinte, apresenta-se a implementação das funcionalidades necessárias ao multijogador online.

4

DESENVOLVIMENTO

Realizada a análise à estrutura e funcionamento da aplicação e às tecnologias necessárias ao desenvolvimento do jogo SAMduelJr, com vista a alcançar o objetivo final, que é a implementação do multijogador, concluiu-se que seria necessário desenvolver algumas funcionalidades auxiliares, tais como: um sistema de autenticação dos utilizadores dentro do próprio jogo, um servidor de gestão da comunicação entre os jogadores, sendo este também responsável por albergar alguma da lógica do jogo, adicionalmente seria necessário desenvolver uma API de suporte ao sistema de ranking.

Primeiramente, será explicado em que consiste o sistema de ranking e de seguida será abordada a construção da API, bem como da base de dados associada à mesma. Só posteriormente é que se entrará em detalhe na componente de autenticação, do servidor de gestão, do novo sistema de jogo e de novas funcionalidades.

4.1 SISTEMA DE RANKING

Um dos requisitos desta dissertação era criar um modo competitivo dentro do jogo **SAMDuelJr**, de forma a incentivar os alunos a jogar mais tempo entre eles e praticar o cálculo mental.

A partir desta ideia surgiu o sistema de ranking, que serve para classificar cada jogador consoante o seu nível de habilidade. Este sistema é constituído por 5 níveis/*ranks*, que vão do nível 1 ao nível 5, sendo mais baixo o nível 1 e o mais alto o nível 5.

Cada jogador tem associado a si um número de pontos que representa o seu nível de habilidade, também chamado de *elo*. Consoante o intervalo de valores em que o elo do jogador se situa, é-lhe atribuído um *rank*.

Todos os utilizadores, à exceção dos visitantes, possuem um *rank*, sendo que na primeira vez que entram no jogo é-lhes atribuído o *rank* mais baixo(1), de maneira a garantir que todos começam do mesmo ponto de partida.

Todas as partidas de e contra jogadores visitantes não contam para *rank*. No caso do utilizador não querer jogar a contar para a sua classificação, este pode jogar contra o computador ou usar o modo de treino que o SAMDuelJr já possuía 2.2. Além destes modos,

existe outro que não influencia o nível de habilidade, que são as partidas iniciadas através do sistema de convite por identificador. Este modo será abordado em mais detalhe à frente.

Relativamente aos modos de jogo competitivo, após conclusão de cada jogo é adicionado um determinado número de pontos ao *elo* do jogador vencedor e subtraído ao do perdedor, sendo que estes pontos são calculados da seguinte forma: o vencedor do jogo ganha os pontos equivalentes ao *rank* do seu oponente(de 1 a 5), o perdedor perde o módulo da diferença dos *ranks* mais um. Exemplo: um jogador de nível 5 joga contra um de nível 2, se o jogador de nível 5 ganhar, ganha 2 pontos, se perder perde |5-2|+1, ou seja, perde 4 pontos. Por outro lado, se o jogador de nível 2 ganhar, este ganha 5 pontos e, se perder, perde |2-5|+1 pontos, ou seja, também perde 4 pontos. Este sistema segue uma ideia de alto risco/alta recompensa. No caso de um jogador de *rank* mais inferior que tente enfrentar oponentes de *rank* superior, quanto maior for a diferença entre eles, mais pontos pode ganhar. No entanto, existe um risco maior pois, se este perder, também perde mais pontos. No caso de um jogador de mais alto nível, que tente só jogar com jogadores de menor *rank*, para mais facilmente ganhar pontos, quanto maior for a diferença entre eles, menos pontos ganhará e em caso de derrota este perderá mais pontos.

Figura 14: Ranks

4.2 API E BASE DE DADOS

No que diz respeito ao desenvolvimento do sistema de ranking anteriormente explicado, levou à necessidade de se verem implementadas novas funcionalidades, nomeadamente uma API de dados que fizesse de ponte para aceder e tratar os dados necessários para o pleno funcionamento do sistema de *rank*, os quais seriam guardados numa base de dados.

Nesta secção, será abordado primeiramente o processo de desenvolvimento da base de dados, nomeadamente da sua estrutura e modelos. De seguida, falar-se-á tanto da estrutura da API, como das rotas implementadas.

4.2.1 *Base de dados*

Uma base de dados é uma coleção de dados ou informação relacionado com determinado assunto, organizados de determinada estrutura, de forma a permitir a consulta, atualização e outros tipos de operações sobre estes mesmos dados [1].

Tendo em conta que toda a estrutura ligada às bases de dados da plataforma Hypatiamat estava montada com *MySQL*, para facilitar a integração deste projeto com a restante estrutura, decidiu-se utilizar também MySQL para a base de dados.

MODELO DE DADOS

Com base no sistema de ranking explicado anteriormente, verificou-se que seria necessário criar uma estrutura de dados(tabela) com a informação de cada jogador e do seu nível de habilidade(*elo*), adicionalmente de forma a tornar cada entrada da tabela única para não haver repetição de informação, foi adicionado um campo que serve como identificador.

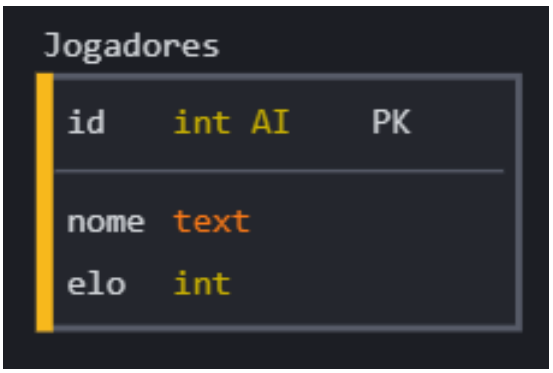


Figura 15: Tabela Jogadores

Como podemos ver acima, criou-se a tabela *Jogadores* com três campos de informação, nomeadamente:

- *id*- Inteiro que representa o identificador único;
- *nome* - Representa o nome de utilizador de cada jogador;
- *elo* - Inteiro que representa o *elo* do jogador.

4.2.2 *API de dados*

No que diz respeito à construção da API de dados, esta foi desenvolvida com **Node.js** 2.3.1, recorrendo à framework do **Express** 2.3.2. Os motivos que levaram à escolha destas ferramentas já foram discutidos anteriormente mas deve-se, principalmente, ao facto destas facilitarem e agilizarem bastante o processo de desenvolvimento de uma API.

ESTRUTURA

Relativamente à API, esta apresenta a seguinte estrutura:

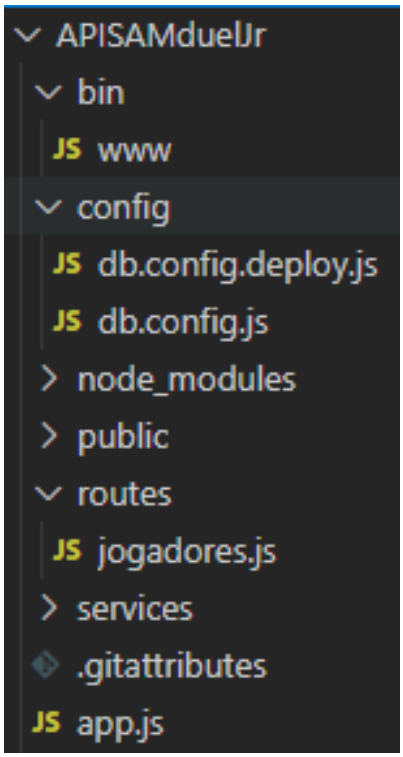


Figura 16: Estrutura da API

De seguida, é explicado cada componente desta estrutura:

- **bin**: É onde se localiza o ficheiro *www*, onde é possível escrever alguns *scrpits* de inicialização da aplicação. Neste caso, é onde se define em que porta o servidor vai estar à escuta;
- **config**: A pasta *config* serve para guardar as credenciais de acesso à base de dados, tendo dois ficheiros, um com as credenciais da base de dados para se usar em estado de desenvolvimento, e outro com as credenciais da base de dados para quando a API está em produção. Isto facilita o desenvolvimento, visto que para alterar as credenciais entre estado de desenvolvimento e estado de produção é só escolher um destes ficheiros. Caso seja preciso mudar as credenciais algum dia, é só alterar os valores nestes ficheiros;

```
module.exports = {
    HOST: "localhost",
    USER: "root",
    PASSWORD: "",
    DB: "hypati67"
};
```

Exemplo 4.1: Exemplo de um dos ficheiros de config.

- **node modules**: Aqui são guardados todos os módulos que o Node.js necessita para este projeto;
- **public**: Nesta pasta, é possível colocar imagens e qualquer tipo de ficheiros, podendo estes serem acedidos de forma pública;
- **routes**: Na *routes* encontram-se os ficheiros *Javascript* onde estão implementadas todas as rotas desta API, também denominados de *routers*;

```
var express = require('express');
var router = express.Router();
const connection = require("../services/db.js");
router.get('/', function(req, res) {
  connection.query('SELECT nome,elo from jogadores', (err, rows) =>{
    if(!err){
      res.status(200).send(rows)
    } else{
      res.status(500).send(err)
      console.log(err)
    }
  })
});
```

Exemplo 4.2: Inicialização do ficheiro da rotas de jogadores e exemplo de uma rota.

- **services**: Nesta componente está definido o código responsável por estabelecer uma conexão com a base de dados, abrindo uma conexão MySQL com a mesma. Para isto, são utilizadas as credenciais dos ficheiros de *config* para aceder à base de dados;

```
const mysql = require("mysql");
const dbConfig = require("../config/db.config.deploy.js");
// Criar conexão à base de dados
const connection = mysql.createConnection({
  host: dbConfig.HOST,
  user: dbConfig.USER,
  password: dbConfig.PASSWORD,
  database: dbConfig.DB
});
// Abrir a conexão MySQL
connection.connect(error => {
  if (error) {
    console.error('Erro a conectar à base de dados: ' + error);
    return;
  }
  console.log("Conexão à base de dados efetuada com sucesso!.");
});
module.exports = connection;
```

Exemplo 4.3: Inicialização do ficheiro de conexão à base de dados.

- **app.js**: O app.js tem a função de importar os ficheiros da *routes* e definir um determinado prefixo para cada uma das rotas associadas a cada um dos respetivos ficheiros da *routes*. Adicionalmente, é responsável por definir algumas opções a serem usadas em toda a aplicação.

```
var express = require('express');
const cors = require("cors");
var corsOptions = {
  origin: "*"
};
var jogadoresRouter = require('./routes/jogadores');
var app = express();
app.use(cors(corsOptions));
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
app.use('/jogadores', jogadoresRouter);
module.exports = app;
```

Exemplo 4.4: Ficheiro app.js.

Deste modo, o fluxo desta aplicação é o seguinte: inicialmente a aplicação é iniciada na porta definida no ficheiro *www* e, de seguida, a componente do *services* utiliza as credenciais presentes nos ficheiros de *config* e estabelece uma conexão à base de dados. Os *routers*, pegam nesta conexão e utilizam a mesma para definir as rotas da API. Os prefixos destas rotas encontram-se definidos no ficheiro *app.js*.

ROTAS IMPLEMENTADAS

Com os modelos da base de dados todos definidos, a base de dados estruturada e com a criação da estrutura base da API, faltava apenas o desenvolvimento das rotas de acesso.

De notar que cada API pode ter diversos prefixos principais, estando estes geralmente associados a determinado modelo de dados. Como neste caso temos somente a coleção dos *jogadores*, decidiu-se usar apenas um prefixo principal que no caso é "/jogadores". Seguidamente, serão apresentadas as diferentes rotas implementadas

JOGADORES

- **GET**
  - "/" - Devolve a lista com a informação de todos os jogadores.

```
router.get('/', function(req, res) {
  connection.query('SELECT nome,elo from jogadores', (err, rows) =>{
    if(!err){
      res.status(200).send(rows)
    } else{
      res.status(500).send(err)
      console.log(err)
    }
  })
});
```

Exemplo 4.5: GET todos os jogadores.

  - "/:username" - Devolve a informação de um determinado jogador, com base no seu nome de utilizador(*username*), o qual é passado como parâmetro.

```
router.get('/:username', function(req, res) {connection.query('SELECT *
    from jogadores WHERE nome = ?', [req.params.username], (err, rows)
    =>{
    if(!err){
      if(rows.length != 0){
        res.status(200).send(rows)
      }else{
```

```
          res.status(404).send("Jogador não existe")
        }
      } else{
        res.status(500).send(err)
        console.log(err)
      }
    })
});
```

Exemplo 4.6: GET de um jogador.

– "/elo/:username" - Devolve o *elo* de um determinado jogador, com base no seu nome de utilizador(*username*), o qual é passado como parâmetro.

```
router.get('/elo/:username', function(req, res) {
  connection.query('SELECT elo from jogadores WHERE nome = ?', [req.
   params.username], (err,  rows) =>{
    if(!err){
      if(rows.length != 0){
        res.status(200).send(rows[0])
      }else{
        res.status(404).send("Jogadores não existem")
      }
    } else{
      res.status(500).send(err)
      console.log(err)
    }
  })
});
```

Exemplo 4.7: GET do elo de um jogador.

– "/top100sempre" - Devolve uma lista com a informação dos cem melhores jogadores do jogo desde sempre, com base no seu *elo*.

```
router.get('/top100sempre', function(req, res) {
  connection.query(`SELECT
      alunos.nome,
      jogadores.elo,
      alunos.escola,
      alunos.turma
    FROM
      jogadores
    LEFT JOIN alunos ON jogadores.nome = alunos.user
    ORDER BY
      jogadores.elo
```

```
      DESC
      LIMIT 100;`,(err, rows) =>{
      if(!err){

        res.status(200).send(rows)

      } else{
        res.status(500).send(err)
        console.log(err)
      }
    })
});
```

Exemplo 4.8: GET cem melhores jogadores de sempre.

– "/top100atual" - Devolve uma lista com a informação dos cem melhores jogadores do jogo do ano letivo atual, com base no seu *elo*.

```
router.get('/top100atual', function(req, res) {
  let hoje = new Date();
  let mes = hoje.getMonth()
  let top = hoje.getFullYear()

  if(mes < 8 ){top -= 1}
  top = String(top).slice(-2)

  connection.query(`SELECT
      alunos.nome,
      jogadores.elo,
      alunos.escola,
      alunos.turma
    FROM
      jogadores
    LEFT JOIN alunos ON jogadores.nome = alunos.user
    WHERE SUBSTRING(alunos.turma, 4 ,2) = ?
    ORDER BY
    jogadores.elo
    DESC
    LIMIT 100;`,[top],(err, rows) =>{
      if(!err){

        res.status(200).send(rows)

      } else{
        res.status(500).send(err)
        console.log(err)
      }
```

```
30    })
});
```

Exemplo 4.9: GET dos cem melhores jogadores do ano letivo atual.

- POST
  - "/"- Insere um novo jogador na base de dados.

```
router.post('/', (req, res) => {
  const params = req.body

  connection.query('INSERT INTO jogadores SET ?', params, (err, rows)
   => {
      if(!err) {
        res.status(201).send(rows)
      } else {
        if(err.errno == 1054){
          res.status(400).send(err)
          console.log(err)
        }else{
          res.status(500).send(err)
          console.log(err)
        }
      }
  })
});
```

Exemplo 4.10: POST de um jogador.

- PUT
  - "/"- Atualiza a informação de um jogador. Esta informação (id, nome e elo), é passada como um objeto *JSON* e encontra-se no *body* do pedido.

```
router.put('/', function(req, res) {
   const { nome, elo} = req.body
   console.log("PUT ",req.body)
   connection.query('UPDATE jogadores SET elo = ? WHERE nome = ?', [elo
   , nome] , (errows)  => {
     if(!err) {
       res.status(200).send(rows)
     } else {
       if(err.errno == 1048){
         res.status(400).send(err)
         console.log(err)
       }else{
```

```
            res.status(500).send(err)
            console.log(err)
        }
    }
  })
});
```

Exemplo 4.11: PUT de um jogador.

4.3 AUTENTICAÇÃO

Relativamente ao processo de autenticação dos seus utilizadores, a plataforma do *Hypatiamat* já dispunha de um sistema para esse efeito mas os seus jogos não incorporam esse sistema. Como se pretende emparelhar os utilizadores para jogarem entre si, classificá-los de acordo com um ranking, entre outros, é necessário que os utilizadores se autentiquem de forma a ser possível identificar cada um deles.

O sistema de autenticação elaborado no jogo *SAMduelJr* é constituído por duas opções. Primeiramente, é possível fazer *login* com as credenciais que os utilizadores já possuem da plataforma, que no caso são o seu nome de utilizador e a sua password, sendo que para a validação das mesmas é utilizada uma API que o *Hypatiamat* já possui para esse efeito. Em alternativa, na eventualidade de existirem utilizadores que tenham intenção de jogar mas que não possuam credencias ou simplesmente não queiram fazer login com as suas, é possível entrar no jogo através da opção de visitante, onde é atribuído ao utilizador um identificador próprio.2.3.4

Para a geração destes identificadores optou-se por utilizar o módulo do **NanoId**, uma vez que este se revelou ser vantajoso em comparação a outros. 2.3.4

No caso dos visitantes, qualquer pessoa pode usar este modo de autenticação para entrar no jogo. Estes não são associados a nenhum *rank*, mas são associados internamente a um tipo específico, de forma a identificar que se trata de um visitante.

```
function loginVisitante() {
  userType = 2;
  username = "visitante-"+nanoid(10);
  rank="";
}
```

Exemplo 4.12: Inicialização de um visitante

Para o formato do identificador, decidiu-se definir como *"visitante-(identificador gerado pelo NanoId)"*, de modo a tornar-se mais legível o nome de utilizador de cada visitante. De seguida, podemos ver alguns exemplos destes identificadores.

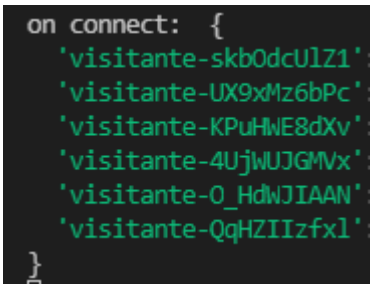


Figura 17: Exemplos de identificadores gerados usando NanoId

Relativamente aos utilizadores que tenham as credenciais da plataforma, é enviado um pedido com as mesmas para a *API* de autenticação que o Hypatiamat possuí, sendo que apenas alunos podem fazer *login* no jogo. Se o utilizador for um aluno e as suas credenciais estiverem corretas, verifica-se se já existe entrada deste utilizador na base de dados dos jogadores e, se existir, calcula-se o *rank* do jogador a partir do *elo* associado ao mesmo, para apresentar na interface o *rank* deste. Se não existir, é criado uma entrada na base de dados com o seu nome e atribui-se um *elo* predefinido para o mesmo. Isto significa que todos os utilizadores, na primeira vez que entram no jogo, começam todos com o mesmo *rank*.

Segue-se a interface desenhada para a página de *login* do jogo.

Figura 18: Login

4.4 SERVIDOR DE GESTÃO

A realização de um jogo *web* multijogador levanta uma problemática: Como efetuar uma comunicação em tempo real entre os jogadores e o servidor do jogo. Esta problemática já foi discutida previamente 3, sendo que nesta dissertação se optou por utilizar uma arquitetura **client-server** 3.2. Esta é mais robusta e adequada ao cenário em questão, uma vez que é necessária a existência de um servidor para gerir as conexões entre jogadores, certificando que estas permanecem abertas. Adicionalmente, é necessário um servidor em que seja possível adicionar e gerir alguma lógica de jogo.

Relativamente à comunicação entre cliente e servidor, é utilizada a tecnologia de **WebSocket** 3.4. De forma a trabalhar com esta tecnologia, decidiu-se utilizar a biblioteca **Socket.IO** 2.3.3.

O Socket.IO necessita de duas componentes: um cliente, que é o jogo web SAMduelJr e de um servidor, que no caso está a funcionar em **Node.js** 2.3.1. Node.js é um software de código aberto, multiplataforma, baseado no interpretador JavaScript de alto desempenho V8, apresentando um paradigma de *I/O* não bloqueante, orientado a eventos. Para além das vantagens já mencionadas do Node.js 2.3.1, acrescenta-se também o facto de o Node.js ser o mais adequado para trabalhar com Socket.IO, um dos motivos pelos quais se escolheram estas duas tecnologias.

4.4.1 *Estruturas de dados*

Uma vez que se verificou ser necessário manter registo de todos os utilizadores que estão *online* no jogo e se os mesmos se encontram ocupados ou não, foram definidas estruturas de dados para guardar informação necessária para manter controle destes casos, entre outros.

Inicialmente, sempre que um jogador entra no jogo após efetuar o *login*, é estabelecida uma conexão *WebSocket* com o servidor, abrindo assim um canal de comunicação bidirecional com o mesmo. Do lado do servidor é guardada informação do nome de utilizador do jogador e do identificador do *socket* da conexão estabelecida, associado a este. Esta informação é guardada na estrutura *usersSockets*, permitindo assim manter registo de quando um jogador entra ou sai do jogo. O identificador do *socket* associado a cada jogador serve para o servidor comunicar com cada um deles. Em suma, a *usersSockets* é responsável por manter registo de todos os jogadores online no SAMDuelJr.

```
let usersSockets = new Object();
```

Exemplo 4.13: Inicialização da estrutura de dados *usersSockets*.

Adicionalmente, também se verificou ser necessário manter registo de todos os jogadores que se encontravam numa partida, para não permitir que outros jogadores pudessem convidá-los para outra partida. Com isto, foi criada a estrutura de dados *usersOnGame* para manter registo destes jogadores.

```
1 let usersOnGame = new Object();
```

Exemplo 4.14: Inicialização da estrutura de dados *usersOnGame*.

Por fim, como será explicado mais adiante, o sistema de fila de espera implementado no jogo tem quatro modos de jogos, que representam as partidas à melhor de 3,5,7 e 9.

Toda a vez que determinado jogador entrar na fila de espera de qualquer um destes quatro modos, do lado do servidor será guardado informação referente ao jogador em cada uma das estruturas associada a cada um dos modos, de forma a só emparelhar jogadores que queiram jogar o mesmo modo. Sempre que o jogador para de procurar por uma partida, no modo de fila de espera, é removido destas mesmas estruturas.

```
let searchingGame3 = new Object();
let searchingGame5 = new Object();
let searchingGame7 = new Object();
let searchingGame9 = new Object();
```

Exemplo 4.15: Inicialização das estruturas de dados para o sistema de fila de espera.

4.5 SISTEMA DE JOGO

Com a premissa de se ver implementada a funcionalidade de multijogador online neste jogo, foi necessário criar uma lógica de jogo nova. Como já vimos anteriormente, o SAMDuelJr já fornecia um modo de jogo contra o computador e um modo multijogador, mas requeria que os dois jogadores estivessem fisicamente juntos e jogassem no mesmo dispositivo (ecrã dividido).

A peça essencial que permite estas partidas entre jogadores de forma online é o servidor de gestão e as conexões *WebSocket* do *Socket.IO*, uma vez que é através destes que existe a troca de informação entre os jogadores. Como veremos nas secções seguintes, dentro da vertente multijogador online, existem dois tipos de modos, o de convite e o de fila de espera. Independentemente de como o fazem, os dois modos acabam sempre por emparelhar dois jogadores para entrarem numa partida. Este emparelhamento é feito através da funcionalidade *Rooms* 2.3.3 do *Socket.IO*,que junta um subgrupo de jogadores num *room*, permitindo enviar mensagens diretamente para o *room*, que são transmitidas também a cada jogador dentro dele. Isto é útil para enviar mensagens e eventos para dois jogadores simultaneamente como, por exemplo, quando começa um jogo, acaba uma ronda, quando algum jogador se desconecta, etc. De seguida, relembrando como funciona a lógica de jogo do SAMDuelJR 2.3.4, será explicado de uma forma mais geral como funciona cada partida multijogador implementada.

Inicialmente, quando os jogadores são emparelhados, existe um temporizador de 3 segundos para os mesmo se prepararem. De seguida, é gerado o quadro de dezoito números inteiros de um a nove para cada um dos jogadores. Para além disso, é também gerado o resultado temporário e a operação a ser usada. As operações possíveis geradas podem variar consoante o sistema de jogo selecionado. No de convite é possível escolher quais das

três operações aritméticas(adição, subtração e multiplicação) serão usadas no jogo mas, no sistema de fila, não é possível escolher e são sempre usadas as três.

Figura 19: Interface de partida multijogador.

Após isto, se quando o jogador selecionar os dois números, estes não derem o resultado temporário mostrado, é reproduzido um som de resposta errada. Por sua vez, se estes números forem a resposta correta, é dado um som de resposta correta, desaparecendo os dois números selecionados do tabuleiro de jogo. Logo a seguir é gerada outra operação de forma aleatória, bem como outro resultado. Sempre que um jogador acerta nos dois números corretos para cada par de operação/resultado, de maneira a garantir que o oponente deste jogador consiga ver esta atualização do seu lado de jogo, é enviado para o servidor a informação referente aos dois números, nova operação e resultado gerados. Esta informação

é depois redirecionada para o respetivo oponente, sendo atualizado do seu lado a jogada feita pelo adversário. Este processo dá-se para os dois jogadores e repete-se até um dos jogadores acabar com os números do seu tabuleiro. Quando isto acontecer, a ronda termina sendo apresentado o resultado da partida até ao momento.

Figura 20: Interface do final de ronda.

Neste menu, no final de cada ronda, existe um temporizador de trinta segundos, para permitir que os jogadores cliquem no botão de jogar para sinalizar que estão pronto para voltar ao jogo.

Figura 21: Interface do final de ronda após clicar em jogar.

No caso do jogador não sinalizar que está pronto, este perderá o jogo.

Figura 22: Interface caso jogador não clique em jogar.

Dependendo do número de rondas que cada jogo tem, que depende se o jogo é à melhor de 3,5,7 ou 9, este processo acima descrito acaba até determinado jogador chegar a certo número de rondas. Quando um jogador ganha ou perde é reproduzido um som de vitória ou derrota, respetivamente, sendo mostrado o menu final de jogo.

Figura 23: Interface do final do jogo em caso de vitória.

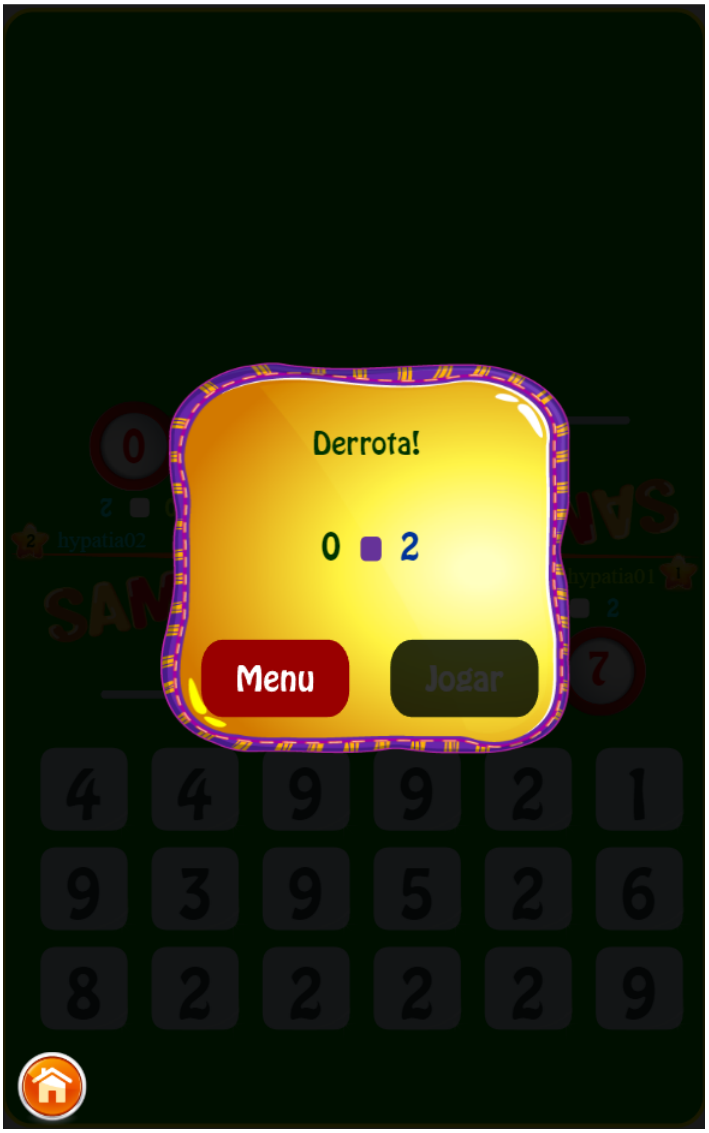


Figura 24: Interface do final do jogo em caso de vitória.

Ao fim da cada partida são efetuado os ajustes de *rank* a cada um dos jogadores, conforme explicado na secção acima 4.1.

Importa referir que durante toda a partida existe um sistema de controlo de desconexão. Sempre que um jogador é desconectado do jogo por algum motivo, por exemplo, quando fecha a página web do jogo ou quando perde a ligação à internet, é avisado o oponente que a partida terminou devido à desconexão do jogador(figura 4.5), sendo também creditada a vitória ao oponente. Adicionalmente, sempre que um jogador decide sair da partida, retornando ao menu principal (figura 4.6), a partir da interface do jogo (figura 4.5), será creditada a vitória para o seu oponente, sendo também avisado do sucedido.

Figura 25: Interface caso oponente se desconecte ou saia da partida.

Figura 26: Interface caso jogador queira abandonar a partida.

Figura 27: Interface do menu principal do jogo.

4.5.1 *Sistema de convite*

De forma a permitir que um jogador consiga convidar outro para uma partida, foi implementado o sistema de convite. Um convite pode ser efetuado de duas formas, convidando outro jogador através da lista de jogadores online ou diretamente através do nome de utilizador de um jogador.

Primeiramente, olhemos para o sistema de convite através da lista de jogadores online. Para este modo, foi desenvolvida uma interface com a lista de todos os jogadores online no jogo (figura 4.5.1), apresentando o nome de utilizador e o *rank* de cada um deles. Para

enviar um convite é apenas necessário clicar no nome do jogador pretendido e confirmar o envio do pedido (figura 4.5.1). O seu oponente será notificado do convite, como podemos ver na figura 4.5.1

Figura 28: Interface da lista dos jogadores online.

Figura 29: Interface confirmar convite.

Figura 30: Interface de um convite.

Relativamente ao convite através do identificador de jogador, apenas é necessário inserir o nome de utilizador do jogador a convidar e confirmar o envio do convite (figura 4.5.1). De referir que, através deste tipo de convite, os jogos não contam para *rank*, enquanto que no sistema anterior contam. O oponente será notificado do convite, como podemos ver na figura 4.5.1

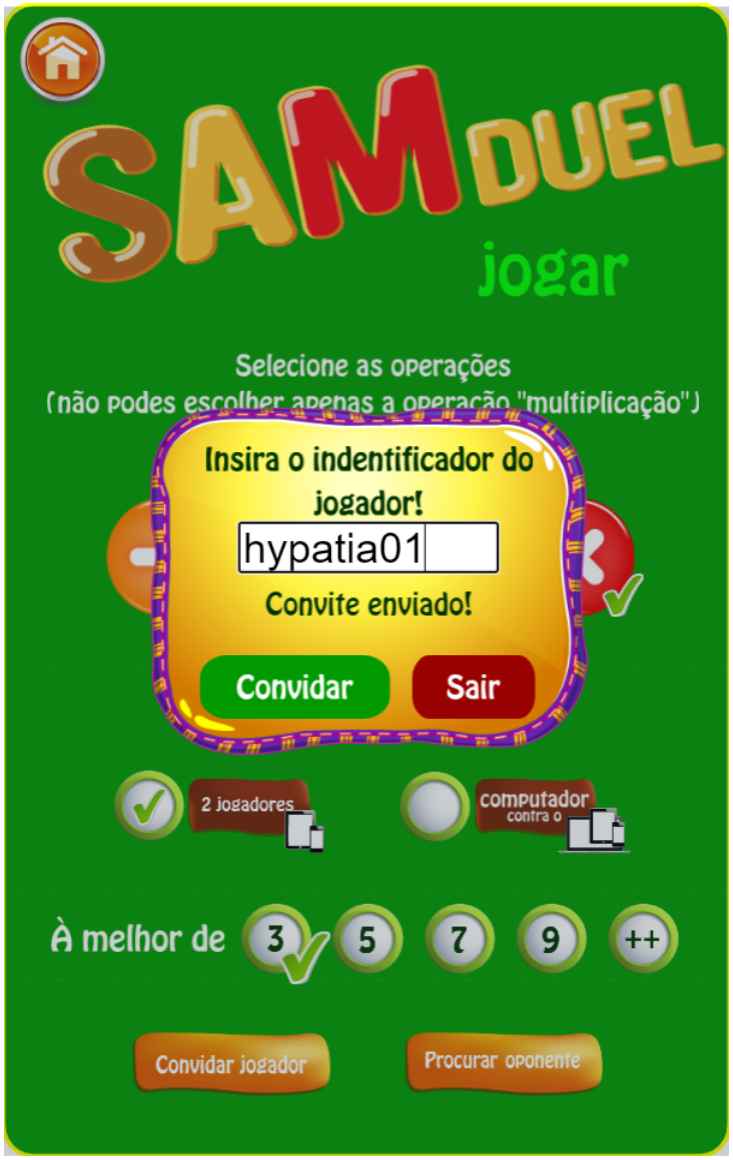


Figura 31: Interface de menu para convidar com identificador.

Figura 32: Interface de um convite através de identificador.

Por fim, independentemente do sistema de convite usado, caso o convite seja aceite, os jogadores entrarão numa partida e, caso seja recusado, o jogador que enviou o convite será notificado (figura 4.5.1).

Figura 33: Interface da notificação de convite recusado.

4.5.2 *Sistema de fila de espera*

No que diz respeito ao sistema de fila de espera, este foi desenvolvido para permitir um emparelhamento justo com outro jogador que também pretenda jogar. Neste modo, mal exista um jogador adequado para iniciar a partida, esta inicia-se ao contrário dos outros sistemas de convite em que é necessário convidar um jogador e este aceitar jogar.

A fila de espera funciona da seguinte forma: primeiramente um jogador tem de escolher quantas rondas pretende jogar, ou seja, escolher o "à melhor de". Existem quatro opções, à melhor de 3,5,7 ou 9. Após escolher uma destas opções, o jogador, clicando em "Procurar

oponente", ficará numa lista de espera até encontrar um oponente adequado. Mal seja encontrado um oponente, a partida começará.

Neste sistema, o emparelhamento dos jogadores é feito com vista a tornar as partidas mais equilibradas, isto é, o nível do *rank* do oponente que um jogador poderá ser emparelhado será, no máximo, um *rank* superior ao seu e no mínimo um *rank* inferior ao seu *rank* atual. Por exemplo, um jogador de *rank* 3 apenas poderá encontrar jogadores desde *rank* 2 a *rank* 4.

Neste modo de jogo são sempre usadas todas as operações aritméticas e os jogos contam sempre para o *rank*. Na figura seguinte é possível ver a interface quando determinado jogador procura por oponente 4.5.2.

Figura 34: Interface sistema de fila.

4.6 TABELA DOS 100 MELHORES CLASSIFICADOS

Com o objetivo de registar os cem melhores jogadores do SAMDuelJr, desenvolveu-se uma tabela para apresentar a informação destes. Esta tabela surgiu também para incentivar a competição entre os jogadores de forma a lutarem por um lugar neste "top 100".

Esta tabela regista os cem melhores jogadores do jogo divididos em duas classes, os cem melhores jogadores desde a criação do jogo e os cem melhores jogadores do ano letivo atual. A tabela apresenta informação relativa aos jogadores, nomeadamente o nome de utilizador, o seu elo, a sua escola e a turma a que pertence. Esta informação encontra-se ordenada do jogador com elo mais elevado para o jogador com menos. Para aceder a este "top 100"é necessário clicar no troféu que se encontra no menu principal 4.6. Na figura seguinte podemos ver a interface implementada para esta tabela.

Figura 35: Interface da tabela dos 100 melhores classificados.

Figura 36: Interface do menu principal.

4.7 ARQUITETURA DA APLICAÇÃO

No início desta dissertação, foi realizada uma análise à estrutura do SAMduelJr, onde inicialmente verificámos que o jogo se encontrava desenvolvido em *Adobe Animate*, constituído por uma camada de *HTML* e uma de *JavaScript* 2.2. Após a implementação das tecnologias apresentadas anteriormente, a estrutura da aplicação assemelha- se ao esquema abaixo apresentado:

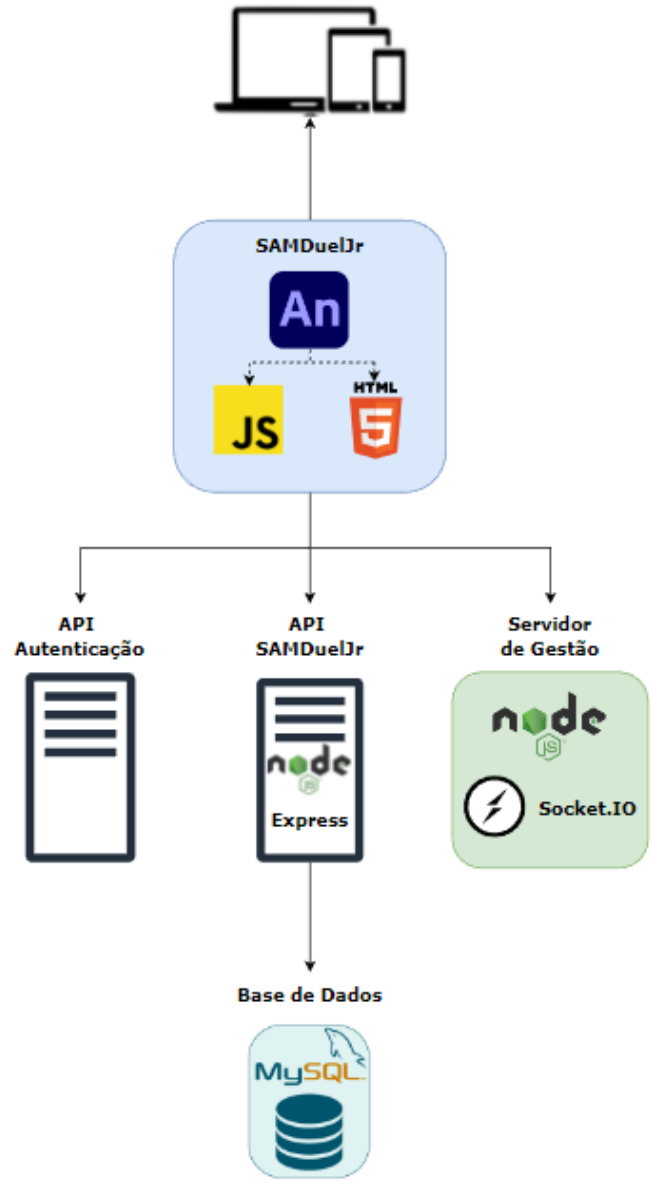


Figura 37: Arquitetura do jogo

Como se pode verificar pela imagem acima apresentada, temos o jogo feito com *Adobe Animate*, sendo que no *Animate* se gera uma camada de *HTML* e uma de *JavaScript*. De seguida, o jogo comunica com a API externa do *Hypatiamat* para realizar a autenticação dos seus utilizadores. Para além disso, existe um servidor desenvolvido em **Node.js**, responsável pela gestão da comunicação entre os utilizadores. Este faz uso da tecnologia de *Websockets* através da biblioteca do **Socket.IO**. Adicionalmente, o servidor é responsável por gerir alguma da lógica de jogo. Por fim, o SAMDuelJr utiliza a *API* desenvolvida nesta

dissertação, a qual foi construida usando **Node.js** e **Express**. Esta API é usada para aceder à base de dados de **MySQL** que alberga a informação necessária ao funcionamento do multijogador online.

Concluindo, ao longo deste capítulo são abordadas todas as funcionalidades desenvolvidas nesta dissertação, desde o sistema de ranking, API e Base de dados até ao sistema de autenticação dos utilizadores, servidor de gestão e ao novo sistema de jogo. Por fim, é apresentada a estrutura final de toda a aplicação. Com isto, falta apenas a componente da integração destas funcionalidades em ambiente de produção, que vai ser abordada no capítulo seguinte.

5

DEPLOYMENT DA APLICAÇÃO

Neste capítulo de deployment, será abordado o processo de integração do servidor de gestão e da API ,desenvolvidos nesta dissertação, na Plataforma Hypatiamat e consequente passagem para ambiente de produção.

Primeiramente foi utilizado a ferramenta *Github* [3] para fazer o controlo de versões, tendo sido criados dois repositórios, um para a API de dados e outro para o servidor de gestão. Estes repositórios são úteis no processo de desenvolvimento, para atualizar a aplicação sempre que existirem novas versões.

A disponibilização das aplicações em ambiente de produção tem como objetivo testar as diversas funcionalidades neste contexto, permitindo receber opiniões relevantes sobre o seu funcionamento e utilidade. Deste forma é possível corrigir eventuais erros que não ocorram num contexto de desenvolvimento.

Para transitar as aplicações construídas para ambiente de produção foi necessário trabalhar conjuntamente com os administradores do servidor e domínio do **Hypatiamat**. Foi então solicitada a criação de dois domínios, um para a API de dados (*apiserversocket.hypatiamat.com*) e outro para o servidor de gestão (*serversocket.hypatiamat.com*).

Após se terem ultrapassado algumas dificuldades no *deployment* das aplicações nestes domínios, os dois serviços ficaram totalmente funcionais e prontos para serem testados e usados. Com isto, foi possível receber feedback útil sobre o funcionamento do jogo, que permitiu o reajuste de determinadas funcionalidades e a correção de erros até ao momento desconhecidos.

Estando todas as componentes desta dissertação em ambiente de produção e todas as correções necessárias feitas, foi possível atingir o grande objetivo desta dissertação, jogar o modo multijogador de forma online.

5.1 MÓDULO PM2

Em ambiente de produção é necessária uma forma de gestão do funcionamento do servidor da API e do servidor de gestão. Enquanto que em ambiente de desenvolvimento ações como iniciar, reiniciar, parar e atualizar os servidores são feitas de uma forma, em

ambiente de produção é necessária outra abordagem para realizar estas operações. Para isto, foi utilizado o módulo do *PM2* [11].

O *PM2* é um gesto de processos que permite administrar aplicações *online*. Este módulo oferece uma interface bastante útil e intuitiva, fornecendo uma série de comandos que permitem iniciar, reiniciar, parar os servidores, etc. Adicionalmente, o *PM2* permite visualizar dados de funcionamentos e de erro das nossas aplicações, oferecendo uma funcionalidade importante como o reinício automático das aplicações em caso de erro, levando a uma constante disponibilidade das mesmas.

6

CONCLUSÃO E TRABALHO FUTURO

Ao longo desta dissertação foram abordados diversos temas.

Inicialmente, contextualizou-se o projeto Hypatiamat, bem como a motivação por detrás desta dissertação. Em relação ao objetivo desta dissertação, o multijogador online, foram planificadas as funcionalidades a implementar e os conceitos necessários para esse fim.

No capítulo seguinte, foi feito um estudo sobre outras plataformas de apoio ao ensino da matemática, tal como o Hypatiamat, e das tecnologias e ferramentas a utilizar no desenvolvimento desta dissertação. Por fim, neste capítulo, foi feita uma análise aos conceitos e medidas definidas referentes ao funcionamento do jogo a ser reformulado, bem como à sua estrutura inicial.

De seguida, foi efetuada uma abordagem a uma componente mais teórica, relativamente a uma das problemáticas mais importantes desta dissertação, a comunicação em tempo real. Neste capítulo foi feita uma análise sobre este tema e sobre diversas formas de o abordar.

Na secção de desenvolvimento foi detalhado todo o processo de desenvolvimento das funcionalidades e sistemas implementados, necessários para o cumprimento de todos os objetivos propostos. Sendo explicados temas como o sistema de ranking, API e base de dados, o servidor de gestão, novos modos de jogo, etc. Por fim, foi apresentada a estrutura final de toda a aplicação desenvolvida.

Finalmente, o capítulo 5 apresenta todo o processo de integração das aplicações desenvolvidas na Plataforma Hypatiamat e consequente disponibilização das mesmas em ambiente de produção para os seus utilizadores.

Relativamente a trabalho futuro, pretende-se realizar uma série de testes de usabilidade com um número considerável de alunos de diversas escolas, de forma a testar a aplicação de uma forma mais séria e em maior escala, com vista a analisar o comportamento do jogo na presença de uma elevada utilização em simultâneo. Adicionalmente, gostariamos de desenvolver novos modos de jogos como, por exemplo, torneios. Estes torneios, juntamente com o modo multijogador online, facilitariam imenso a realização dos diversos campeonatos que o Hypatiamat realiza.

BIBLIOGRAFIA

[1] *Base de dados*. URL: https://www.oracle.com/database/what-is-database/. Acedido em 10/10/2022.

[2] *Express*. URL: https://expressjs.com/. Acedido em 19/10/2022.

[3] *GitHub*. URL: https://github.com/. Acedido em 19/10/2022.

[4] *Hypatiamat*. URL: https://www.hypatiamat.com/apresentacao.php. Acedido em 04/02/2022.

[5] *Hypatiamat: SAMduelJr*. URL: https://www.hypatiamat.com/jogos/samdDuel/SAMduelJr.html. Acedido em 04/02/2022.

[6] *IETF*. URL: https://www.rfc-editor.org/rfc/rfc4122. Acedido em 27/09/2022.

[7] Robin Jan Maly. *Comparison of Centralized (Client-Server) and Decentralized (Peer-to-Peer) Networking*. 2013. URL: https://pub.tik.ee.ethz.ch/students/2002-2003-Wi/SA-2003-16.pdf.

[8] *MySQL*. URL: https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html. Acedido em 19/10/2022.

[9] *NanoId*. URL: https://github.com/ai/nanoid#readme. Acedido em 04/02/2022.

[10] *Node.js*. URL: https://nodejs.org/en/about/. Acedido em 19/10/2022.

[11] *PM2*. URL: https://pm2.keymetrics.io/docs/usage/quick-start/. Acedido em 19/10/2022.

[12] Rodrigo Silveira. *Multiplayer Game Development with HTML5*. Mai. de 2015. ISBN: 978-1-78528-310-9.

[13] *Socket.IO*. URL: https://socket.io/docs/v4/. Acedido em 04/02/2022.

[14] *WebSocket Protocol*. URL: https://datatracker.ietf.org/doc/html/rfc6455. Acedido em 04/02/2022.