Batalha Naval

INFRAESTRUTURAS COMPUTACIONAIS DISTRIBUIDAS

47094 Pedro Azevedo

48960 Ricardo Pedro

Engenheiro Porfírio Filipe

26 de junho de 2022

Licenciatura Engenharia Informática e Multimédia

Instituto Superior de Engenharia De Lisboa

Conteúdo

1. Introdução

1.1. Objetivos

Este projeto do âmbito da Unidade Curricular tem como objetivo desenvolver uma infraestrutura computacional capaz de realizar jogos de Batalha Naval entre 2 jogadores. Estes jogadores podem efetuar o seu registo com um *nickname*, uma *password* e também com uma foto, data de nascimento e cor.

A primeira parte deste projeto será implementada em modo texto, de modo a tratar dos aspetos essenciais como o jogo, a conexão e a gestão de contas. Na segunda parte do projeto irá ser expandido e implementado a versão World Wide Web da Batalha Naval.

A conexão entre servidor e cliente será realizada através de uma ligação *TCP* e a comunicação entre ambos através de documentos *XML*, sendo estes validados por ficheiros *XSD*.

Além da implementação web através de JSP e *Servelets*, será possível alterar o seu perfil (incluindo a idade e cor preferida), procurar por oponentes através do seu nome, onde cada utilizador terá 30 segundos por jogada.

Será ainda realizado um ranking com os melhores 10 jogadores ordenado pelas vitórias, com as suas respetivas fotos, sendo o desempate realizado através do tempo médio de jogo do utilizador, por fim será necessário garantir que o mesmo é preservado através de cópias de segurança.

2. Implementação

2.1. Diagrama Geral da Arquitetura

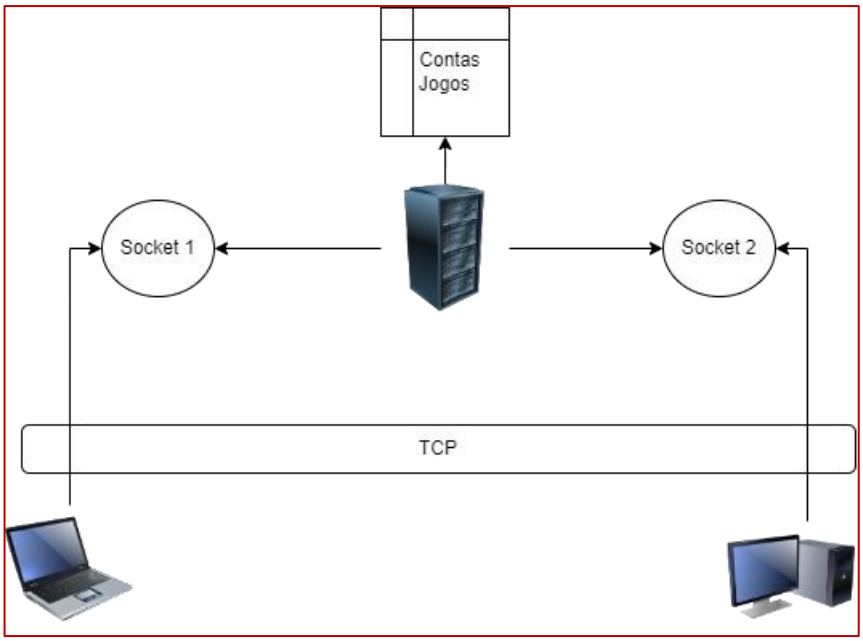

Figura 1 - Diagrama Geral da Arquitetura

Na Figura 1 - Diagrama Geral da Arquitetura representada o diagrama geral da arquitetura. Existe um Servidor que vai guardar as contas dos jogadores, e vai executar os vários jogos da Batalha Naval que podem decorrer em simultâneo. Os clientes vão conectar se o servidor através de uma ligação *TCP* apartir de sockets, cada cliente cria um socket novo.

O Servidor vai execuar Threads para cada Menu Inicial de cada jogador, e uma Thread para cada Jogo. Durante um jogo, o cliente simplesmente vai enviar as jogadas e receber o *feedback* das mesmas, assim, o servidor terá a lógica toda do jogo. Com isto temos uma aplicação mais segura, sendo que os cliente não têm acesso ao código fonte do jogo, minimizando o risco de ser manipulado.

A arquitetura realizada é constituida por uma classe java que executa as funções de Servidor, uma classe que a ser executada é um cliente na linha de comandos, em modo de texto, e para executar um cliente em formato web, é usado JSP's e uma classe Cliente, estas classes conectam se a classe do Servidor e comunicam entre si. Para alem destas, a aplicação contém quatro classes que constituem o jogo da Batalha Naval, *Jogo.java*, *Jogador.java*, *Casa.java* e *Barco.java*. Por fim, adicionamos a classe *XmlDoc.java*, que terá vários métodos para auxiliar a criação, validação e manipulação de *XML*, *XSD,* e *Strings* e dois *Servelets* para ajudar a fazer *upload* e *download* das imagens de perfil.

2.2. Protocolos de transporte (TCP)(sem alteração)

Para realizar uma comunicação entre o servidor e clientes, é usado o protoloco de transporte *TCP*, pois este protocolo, não preserva a fronteira de mensagens, é fiável, ou seja, garante a entrega, preserva a ordem de transmissão, e não admite duplicados, assim a troca de informação entre os clientes e o servidor (seja sobre o jogo ou contas) é mais segura. Este protocolo garante também uma comunicação bidirecional, assim como o facto de o processo poder trabalhar simultaneamente com várias conexões, permitindo assim vários jogos ao mesmo tempo. A maior vantagem ao usar o protocolo *TCP* é que este requer uma conexão estabelecida para que a informação e os dados sejam transmitidos.

Concluímos que este protocolo é mais indicado que o protocolo *UDP*, pois, nesta arquitetura, tem que se garantir a entrega dos dados, para que o jogo possa fluir corretamente. Para além disso, é necessário haver uma ligação permanente, algo que o protocolo *UDP* não permite.

Na arquitetura realizada, para estabelecer uma conexão *TCP*, na classe Servidor é criado um *ServerSocket* (objeto do Java da API java.net) com o porto 5025. O Servidor vai ficar a espera que outra máquina se conecte a este *socket*, utilizado o método *serverSocket.accept()*, sendo que só de avança no código, quando outra máquina se conecte. Podemos ver esta utilização na Listagem 1.

```
serverSocket = new ServerSocket(DEFAULT_PORT);


for(;;) {
   System.out.println("Servidor TCP concorrente aguarda ligacao no
porto " + DEFAULT_PORT + "...");

   // Espera ligacao do primeiro jogador
   System.out.println("Espero por jogador");
   Socket newSock = serverSocket.accept();
   sockets.add(newSock);
.
.
.
```

Listagem 1 - Iniciar ServerSocket

Para um cliente se conectar, este cria um *Socket* (objeto do Java da API java.net). Para se conectar ao *socket* do servidor, o cliente tem que colocar o endereço IP da máquina onde reside o servidor, o porto 5025 (para igualar o que o servidor indicou).

Para o Servidor enviar mensagens ao cliente, este receber e ao contrário, ambos têm que ter um *BufferReader* (para ler as mensagens recebidas), e o *PrintWriter* (para enviar as mensagens). Na Listagem 2 podemos ver a inicialização do *socket* que irá ligar ao *socket* do servidor, e a inicialização do *BufferReader* e do *PrintWriter.*

```
Socket socket = null;
BufferedReader is = null;
PrintWriter os = null;
boolean logout = true;

try {
   do {
      socket = new Socket(DEFAULT_HOST, DEFAULT_PORT);

      // Mostrar os parametros da ligação
      System.out.println("Ligacao: " + socket);

      // Stream para escrita no socket
      os = new PrintWriter(socket.getOutputStream(), true);

      // Stream para leitura do socket
      is = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
```

Listagem 2 - Socket cliente

2.3. Componentes e protocolos da aplicação

Quando o servidor é executado, é criado um *ServerSocket* que fica a espera que um cliente se conecte, realizando assim uma comunicação *TCP*. Do lado do cliente, é criada um *socket* com o IP da máquina onde reside o servidor, e com a mesma porta que se colocou no *ServerSocket*, neste caso o porto 5025.

Inicio de Sessão – Criar Conta

Após estabelecida a ligação entre o cliente e o socket, o cliente terá que escolher entre Criar uma Conta, ou Iniciar Sessão, no caso deste já se ter registado (Tópico desenvolvido no ponto 2.4).

Na segunda parte do trabalho, é necessário ir buscar a JSP *index.jsp* ao servidor, aí é criado um atributo com o cliente para depois efetuar a sua conexão, como já explicado anteriormente no ponto 2.2. De seguida é utilizado um *form* para obter tanto o *username* quanto a *password* de modo a iniciar a sessão, criando assim o *XML* para efetuar a sua validação. Se o mesmo for validado é então criado o atributo *name* com o *username* para ficar disponível na sessão (utilizando no menu) e é feito o *forward* do jogador para o menu. (Na *Listagem 3*,podemos ver uma parte da JSP *index.jsp*, primeira página que o cliente se depara).

```
<div>

    <a>Bem vindo ao Batalha Naval!</a><br />

    <form method="post">
        <a>Utilizador</a>
    <input type="text" style="width: 100px;"name="j_nickname" required><br />
    <a>Password</a> <input
            type="password" style="width: 100px;" name="j_password" required><br />
        <p id="passErrada" style="color: red">${erroPass}</p>
        <input type="submit" value="Login">
    </form>
    <a href="criarconta.jsp">Criar nova conta</a><br />
        <%
      String nickname = request.getParameter("j_nickname");
      String password = request.getParameter("j_password");

      if (nickname != null && password != null) {
         String inicioSessao_xml = cliente.inicioSessao(nickname, password);

         boolean validar = cliente.verificarConta(inicioSessao_xml, 1);
         if (validar) {
            session.setAttribute("name", nickname);
            cliente.name = nickname;
            request.getRequestDispatcher("/menu.jsp").forward(request, response);
         } else {
            request.setAttribute("erroPass", "O utilizador ou password que inseriste está incorreto.");
         }
      }
      %>
```

Listagem 3 - Excerto do index.jsp

Já na criação de uma conta, é se encaminhado para outra JSP, onde é possível adicionalmente escolher a data de nascimento e a sua cor preferida novamente através de um *form*, ao fazer *submit* a página é atualizada, e como já existe um nickname passamos para a criação do XML da criação de conta para ser validado e de seguida é apresentado o *form* capaz de fazer o *upload* da imagem de perfil, através da classe *FileUpload*.

Menu

Após o registo, na JSP do Menu (*menu.jsp*), o jogador tem várias opções. Este pode editar o seu perfil, listar o ranking dos 10 jogadores com mais vitorias, e pode começar um jogo novo. Ao selecionar uma opção, vai ser criado e enviado um *XML* com o pedido, que será validado do lado do cliente a partir do ficheiro *XSD protocolo.xsd*. Este ficheiro é constituído por dois elementos complexos, conta (que é usado quando é criada uma conta ou iniciada a sessão na mesma, ponto 2.4), e o menu. No elemento menu, pode se escolher entre 4 elementos, novoJogo (opção para iniciar um novo jogo), editarPerfil (opção para editar o perfil), ranking (opção para listar o ranking dos 10 jogadores com mais vitorias) e sair (opção para sair do jogo).

Perfil

Por um lado, ao escolher a opção do menu “editar perfil”, o jogador será encaminhado para a JSP *perfil.jsp* onde vai puder alterar o mesmo. A página terá o aspeto como vemos na imagem *Figura 2*. É apresentado o nome do utilizador seguido da imagem previamente escolhida. É possível submeter uma imagem nova, alterar a *password* assim como a cor. Podemos consultar a idade do utilizador e as suas estatísticas de jogo. Para o cálculo do tempo médio por jogo, após cada jogo é adicionado o somatório do tempo de cada jogada ao tempo que está no *xml* do utilizador, e ao carregar a página do perfil é calculada a médio com esse tempo e o número de jogos.

Figura 2 - perfil.jsp - na web

Esta parte de alterar o perfil foi adaptada, pois na primeira parte do trabalho, o jogador escolhe se quer alterar a *password* ou a imagem, fornecendo os dados da alteração.

Quando os dados são submetidos, (em qualquer das partes), é enviado o *XML* com os novos dados e é verificado com o *XSD* acima referido. O *XML* terá de conter o nome do jogador, a nova *password* e/ou imagem e/ou cor, em elementos pertencentes ao elemento pedido, que por sua vez é filho do elemento *editarPerfil*. O Servidor vai verificar se os dados são validos e se assim for alterará a os dados do ficheiro *XML* correspondente ao Jogador (ver 2.42.4 Estruturas de dados dos jogadores). O servidor envia um *XML* a confirmar se alterou com sucesso, para tal, o ficheiro tem o formato semelhando a de quando o cliente envia os dados, contudo, em vez do elemento “pedido” com os seus filhos, o ficheiro contém um elemento “resposta” com o contexto a dizer *true* ou *false*, sendo que se for *true*, regressa se a página do menu.

Ranking

Por outro lado, ao escolher a opção do menu “ranking”, é enviado um *XML* ao servidor, baseado no *XSD protocolo.xsd*, onde a estrutura é o elemento “pedido”, com o elemento pai “*ranking*” (que tem como elementos pais, “menu”, “protocolo”, respetivamente). O Servidor, ao receber esta mensagem coloca os 10 melhores jogadores no *XML*, o cliente ao receber mostra a lista dos jogadores como vemos na imagem *Figura 3*.

Figura 3 - ranking.jsp - na web

Para definir a ordem, o servidor vai ao *XML* de cada jogador e ordena por número de vitórias, para desempatar, é calculado a média de tempo por jogo.

A média é calculada utilizando o tempo total de jogo do utilizador (somatório do tempo total de cada jogo) que está guardado no *XML* do jogador, e o número de jogos realizados.

(Os tempos na figura são baixos pois os jogos realizados continham apenas 1 barco para efeitos de teste).

Jogo

Por fim, o jogador tem a opção de jogar um novo jogo. Neste caso, o cliente fica a espera que outro jogador selecione a mesma opção. O Servidor recebe um *XML* com o elemento "novoJogo", verificando se existe algum jogo criado com um jogador em espera, nesse caso, adiciona-o e começa o jogo. Se não existir nenhum jogo criado, ou seja, nenhum jogador em espera, é criado um novo jogo. Cada jogo é uma *Thread*.

Quando um jogo é iniciado, os jogadores vão para uma outra JSP onde recebem os tabuleiros, quer o seu (com os barcos numa disposição aleatória) quer o do oponente (onde só apresenta os barcos atingidos e as casas falhadas). Podemos ver um exemplo de um jogador na *Figura 4*.

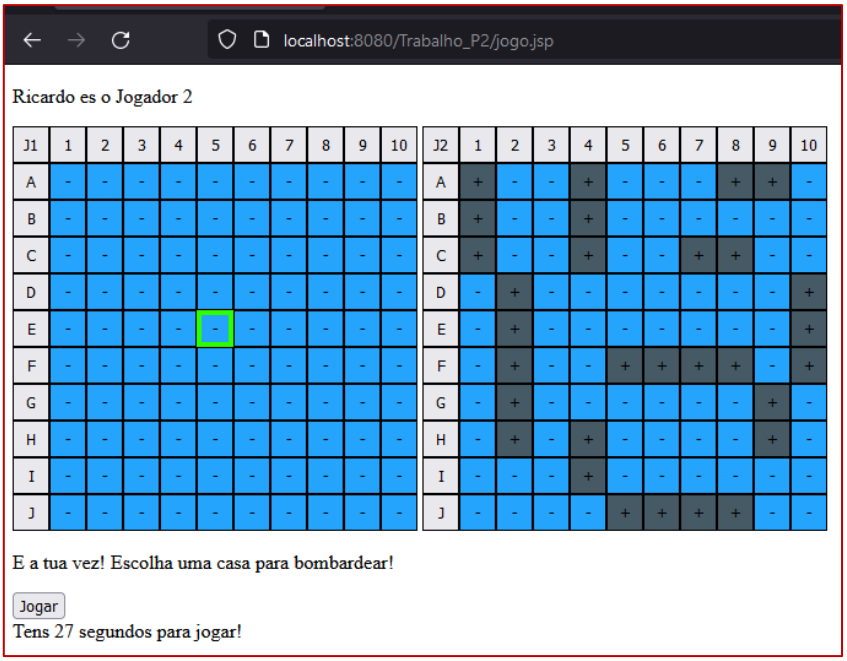

Figura 4 - jogo.jsp - web

Durante a jogada de um jogador, este pode clicar na casa que deseja bombardear e de seguida clicar no botão Jogar para submeter a jogada. É enviado para servidor um *XML* contém o elemento casa tendo como contexto a coordenada, este é validado a partir do ficheiro *xsd jogada.xsd*. O servidor verifica se a jogada é válida, e atualiza os tabuleiros (ver resultado no tópico *14*.

Cada casa será um objeto contendo dois Estados, (Barco, Água, Arder, Destruído, Escondido) sendo que um dos estados é para o tabuleiro do próprio jogador, e o outro é com o estado que o adversário pode ver (assim impede que quando é enviado o tabuleiro ao adversário, este não veja o mesmo estado que o dono do tabuleiro, escondendo os barcos não atingidos). Cada tabuleiro é constituído por um *ArrayList* de Casa, e para o servidor enviar para o cliente, é criado um *XML* que é validado pelo ficheiro *XSD tabuleiro.xsd*. Verificamos no ficheiro que o *XML* terá o elemento root "jogo", com o filho "tabuleiro" e este contém vários elementos "casa" que são as casas do tabuleiro dos jogadores. Este elemento é constituído por um atributo "coordenada", onde a coordenada em vez de ser a letra (A-J) e o número, é o número da linha e da coluna (para passar para o carater correspondente a letra, é converter o número da linha somado 64 para *Ascii*). O elemento casa tem como contexto o símbolo correspondente ao Estado da casa. O cliente ao receber o tabuleiro converte-o para botões com o *value* igual a sua cooredenada.

De modo a limitar as jogadas para 30 segundos, foi utilizada a função *startTimer* (em *JavaScript*) que no início de cada jogada começa um temporizador onde se o tempo é ultrapassado, é enviada uma jogada nula (coordenada igual a "00") para o servidor (foi necessário alterar o XSD) e assim o jogo interpreta essa jogada, passando a vez para o próximo jogador.

No final do jogo, é incrementado um jogo a cada jogador, uma vitória ao jogador vitorioso e é adicionado o somatório dos tempos de cada jogada, nos respetivos campos do *XML* de cada um dos jogadores. E o jogador é encaminhado para a página do menu inicial e iniciada uma nova *Thread.*

Para entrar num jogo, o jogador é encaminhado para uma página onde terá 3 opções, Criar Sala, Entrar numa Sala, Jogador Aleatorio.

No caso de o jogador escolher Criar Sala, este vai criar uma *thread novoJogo* e fica a espera que um jogador se junte a ele, chamamos este tipo de jogo, jogo privado, e o jogo criado é adicionado a uma lista de jogos (*novoJogo*) privados.

Na segunda opção, Entrar numa sala, o jogador vai entrar num jogo privado, este tem uma barra de pesquisa onde pode pesquisar o nome do jogador que pretende jogar. Nesta pesquisa tem um *auto complete,* onde o jogador pode selecionar o nome de uma lista de nomes, estes serão os nomes dos jogadores que criaram jogos privados. Basta então escrever as iniciais do nome, que o resto vai aparecendo num *dropdown*, sendo que aparecem os nomes que contem as letras escritas. Para completar o nome, basta pressionar a teclar "seta direita", que completa, ao submeter é adicionado o jogador a *thread* do jogo privado.

Por fim, o jogador pode selecionar a opção de Jogador Aleatorio, onde o servidor verifica se existe alguma *thread* de *novoJogo* criada, numa lista de jogos aleatórios, não contando com os jogos privados previamente referidos, e se sim e se esta só tenha uma conexão, um jogador, adiciona o utilizador a esta, iniciando o jogo. No caso de não existir nenhum jogo criado, ou só com uma conexão, é criado um jogo novo, um jogo aleatório, adicionando o jogador e este ficando a espera que outro jogador se conecte. Podemos ver esta condição na *Listagem 4*.

```
boolean emJogo = false;

for (NovoJogo jogo : this.jogosRandom) {
   if (jogo.getConnections().size() == 1) {
      System.out.println("A entrar num jogo");
      jogo.addConnection(this.connection, this.getUserName());
      jogo.start();
      emJogo = true;
      break;
   }
}
if (!emJogo) {
   System.out.println("Criar um jogo");
   NovoJogo newJogo = new NovoJogo(this.jogosRandom,
this.jogosPrivados, true);
   this.jogosRandom.add(newJogo);
   // System.out.println(this.getUserName());
   newJogo.addConnection(this.connection, this.getUserName());

}
```

Listagem 4 - Entrar em jogo aleatório

2.4. Estruturas de dados dos jogadores

Quando um novo jogador cria uma conta, este escolhe um *nickname* (que terá que ser único, entre 3 e 15 caracteres), uma *password* (com um tamanho mínimo de 5 caracteres e um máximo de 15), uma imagem, seleciona a sua data de nascimento e uma cor. Ao guardar a conta, esta terá um parâmetro que contém o número de vitorias do jogador, o número de jogos e o tempo total que demora cada jogo. É armazenado um ficheiro *XML*, em que o nome é o *nickname* do jogador. O servidor contém uma pasta denominada por "imagens" que guarda as imagens cujo nome é o do usuário.

Após o jogador preencher os dados, estes são enviados no formato *XML* para o servidor que depois é validado por um *XSD*, onde caso seja válido, verifica se o *nickname* é único. Para tal, cria se um ficheiro *XML* (o nome do ficheiro é o *nickname*) e verifica se o ficheiro já existe com a função *File.exists()*. Ao validar o XML com o *XSD*, consegue se verificar se a *password* e o *nickname* têm as dimensões corretas.

Ao realizar a segunda parte do trabalho, certas alterações foram feitas nos *XSDs.*

No *jogada.xsd*, no elemento casa, adicionou se na restrição, a possibilidade do *value* ser "00" para que se possa enviar uma jogada nula no caso de o jogador ficar sem tempo. Como podemos ver na *Listagem 5*.

```
<xs:element name="casa" >
   <xs:simpleType>
      <xs:restriction base="xs:string">
         <xs:pattern value="([A-J]([\d]|[1][0]))|([0][0])"></xs:pattern>
      </xs:restriction>
   </xs:simpleType>
</xs:element>
```

Listagem 5 - excerto de jogada.xsd

No ficheiro *protocolo.xsd*, foi feita uma alteração no elemento *novoJogo*, sendo agora necessário inserir um dos seguintes elementos, `jogadorAleatorio`, para quando se quer jogar um jogo com um jogador aleatório, `criarJogoPrivado`, para quando se quer criar um jogo privado, e jogar com alguém em específico, `escolherJogador`, para escolher com quem se quer jogar de entre os jogadores que criaram jogos privados. Nesta última opção, é necessário ainda acrescentar o elemento pedido e listar, quando o cliente envia para o servidor para pedir a lista de jogadores, ou então, resposta mais os jogadores, para o servidor enviar os jogadores ao cliente.

Quando é iniciada uma *thread* do menu inicial, é atualizado o ranking, e este é guardado num ficheiro *xml,* validado pelo *xsd, protocolo.xsd*. É atualizado neste local pois, sempre que acaba um jogo, os jogadores vão para o menu inicial, estando sempre a atualizar quando um jogo termina.

No *auto complete* usado para procurar jogadores que criaram jogos privados (ver penúltimo parágrafo do ponto - *Jogo* – na secção *2.3*) , primeiro é guardado os nomes dos jogadores que criaram os jogos, é escrito num ficheiro *txt* sendo cada linha um jogador (*jogadoresDisponiveis.txt*). Para fazer a lista para o aparecer na barra de procura, é executada uma leitura linha a linha para retirar os nomes.

3. Aplicação

Ao iniciar o servidor, este fica à espera de duas conexões de clientes e que efetuem o seu registo ou início de sessão e vão para o Menu Inicial, com vemos na **Erro! A origem da referência não foi encontrada.**.

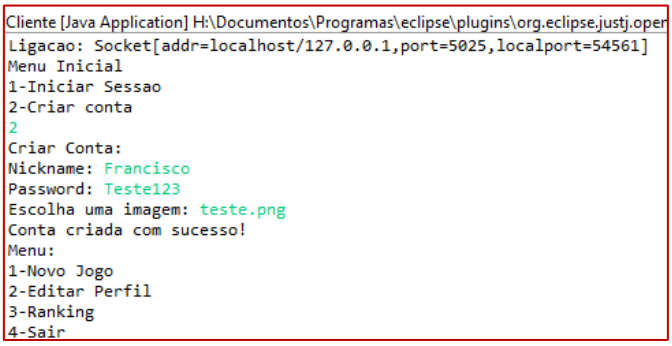

Figura 5 - Criar Conta -pojeto parte 1

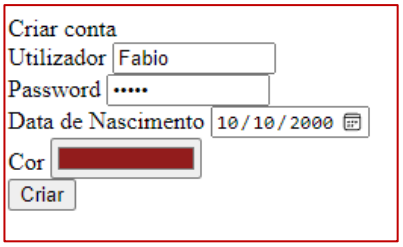

Figura 6 - Criar conta - projeto parte 2

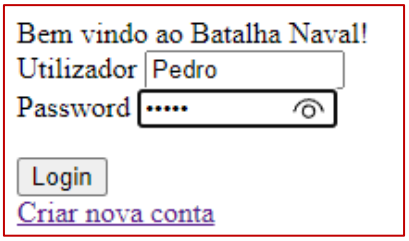

Figura 7 - Inicio de sessão - projeto parte 2

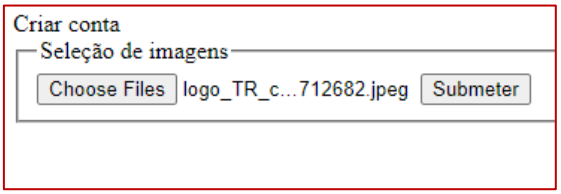

Figura 8 - Submeter imagem - projeto parte 2

No menu inicial o utilizador pode escolher porprocurar por um jogo, editar o seu perfil, ver o ranking ou sair do jogo.

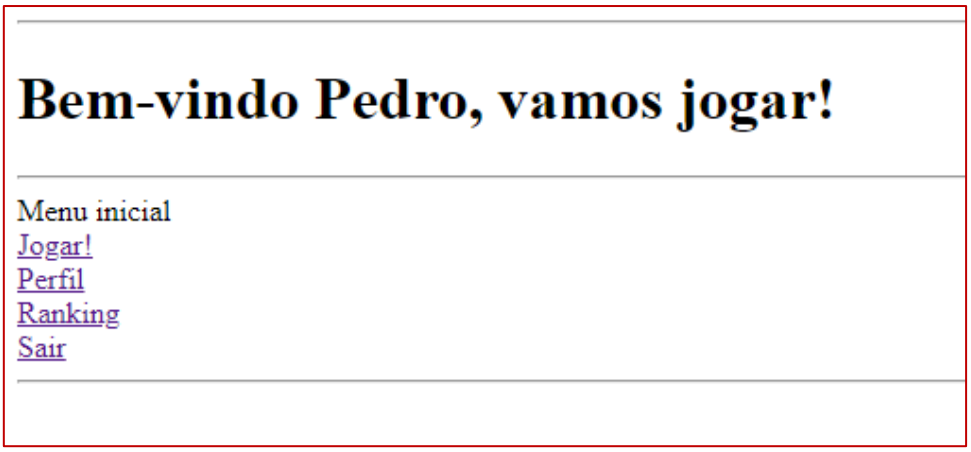

Figura 9 - Menu Inicial -projeto parte 2

Ao editar o perfil, é possível alterar a password, a sua imagem, a sua cor preferida e visualizar as suas estatísticas de jogo.

```
Menu:
1-Novo Jogo
2-Editar Perfil
3-Ranking
4-Sair
2
Alterar:
1-Password
2-Imagem
2
Nova imagem: foto.png
A espera.........
Perfil alterado com sucesso
```

Figura 10 - Alterar imagem perfil - projeto parte 1

Figura 11 - Editar perfil - projeto parte 2

Assim como é possível também obter o *top* 10 de utilizadores com mais vitórias, juntamente com o fator de desempate (média de tempo por jogo) e a sua imagem de perfil.

```
Menu:
1-Novo Jogo
2-Editar Perfil
3-Ranking
4-Sair
3
A espera.........
Ranking
Ricardo - 2 Vitorias
Pedro - 1 Vitoria
Joao - 0 Vitorias
Francisco - 0 Vitorias
```

Figura 12 - Ranking - projeto parte 1

Figura 13 - Ranking - projeto parte 2

Ao selecionar jogar é possível criar uma sala para jogar com um jogador específico, ou seja, o outro jogador basta selecionar entrar numa sala ou então jogar com um jogador aleatório.

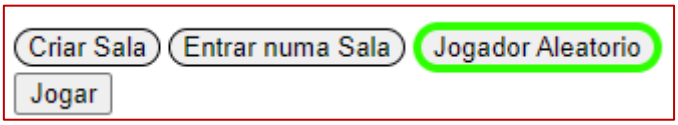

Figura 14 - Menu jogo - projeto parte 2

Entrando na sala de um jogador é possível procurar por nome e utilizar a ferramenta *AutoComplete.*

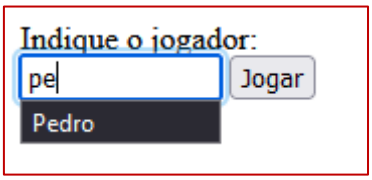

Figura 15 - AutoComplete - projeto parte 2

É feita a espera por um adversário, e ao começar, é definido aleatoriamente a posição dos barcos e também quem é o primeiro a jogar.

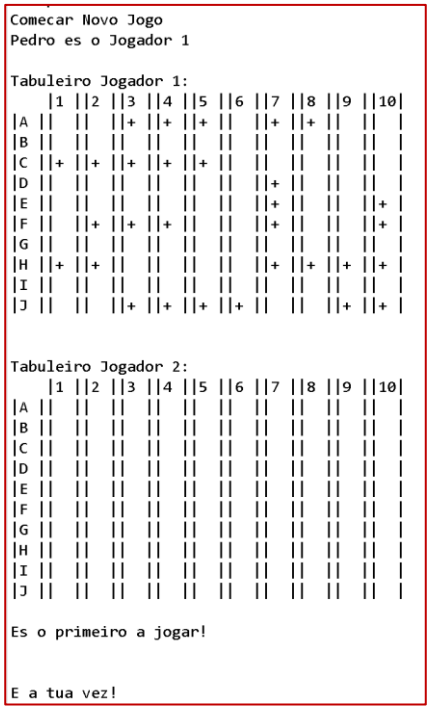

Figura 16 - Inicio de um novo jogo - projeto parte 1

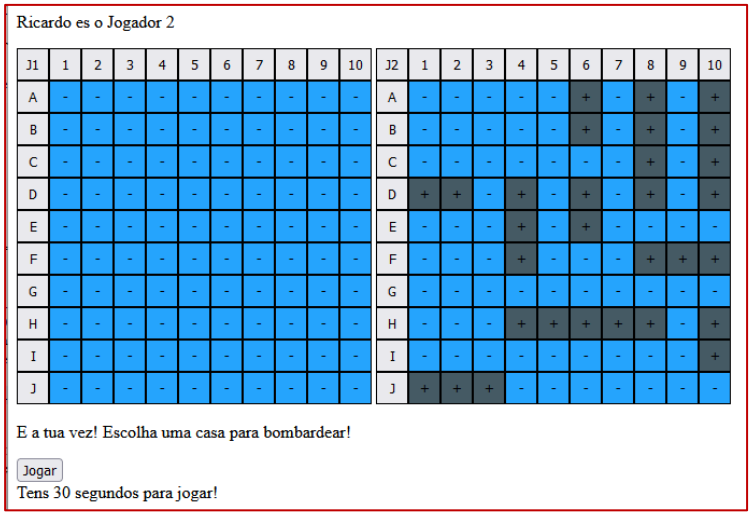

Figura 17 - Inicio de jogo - projeto parte 2

Os barcos são representados no tabuleiro como símbolo "+" ou cor cinza, no caso de já terem sido bombardeados são representados com o símbolo "#" ou cor laranja e se já foram destruídos, representados com o símbolo "$" ou cor vermelha e no tabuleiro que representa o do jogador adversário, contudo, apenas será enviado a localização dos barcos atingidos e os tiros na água (representados com o símbolo "X" ou cor azul).

J1	1	2	3	4	5	6	7	8	9	10
A	-	-	-	-	-	-	-	-	-	-
B	-	+	-	-	+	-	+	-	-	-
C	-	+	-	-	+	-	+	-	-	-
D	-	+	-	-	-	-	+	-	-	-
E	-	-	+	+	+	-	+	-	-	-
F	+	+	-	-	-	-	+	-	-	-
G	-	-	+	+	+	+	-	-	-	+
H	+	+	-	-	-	-	$	-	-	+
I	-	-	-	-	-	-	$	-	-	-
J	-	$	$	$	$	-	$	-	-	-

J2	1	2	3	4	5	6	7	8	9	10
A	-	-	-	-	-	-	-	-	-	-
B	-	-	x	-	x	-	-	-	-	-
C	-	-	-	-	#	-	-	-	-	-
D	-	-	-	x	#	x	-	-	-	-
E	-	-	x	-	#	-	-	-	-	-
F	-	-	-	-	#	-	-	-	-	-
G	-	-	-	-	-	-	-	-	-	-
H	-	-	-	-	-	-	-	-	-	-
I	-	-	-	-	-	-	-	-	-	-
J	-	-	-	-	-	-	-	-	-	-

Figura 18 - Vista do jogador 1 no decorrer do jogo - projeto parte 2

Uma casa quando é selecionada fica a verde e depois é necessário pressionar no botão Jogar! O jogador está limitado a 30 segundos por jogada, passando de vez se não o fizer.

E a tua vez! Escolha uma casa para bombardear!

Jogar

Tens 30 segundos para jogar!

Figura 19 - Timer e botão de submeter jogada - projeto parte 2

Ao efetuar uma jogada verifica se é válida, que tipo de casa estamos a atingir e quantos barcos faltam.

```
Escolha uma casa para bombardear (inserir letra e a seguir o numero, tudo junto) ->A1
Casa valida
Casa A1 bombardeada! Acertaste em:
Agua!!
Faltam 10 barcos
```

Figura 20 - Jogada - projeto parte 1

Por fim, na última jogada é informado a ambos jogadores o vencedor, e a este é lhe atribuído a vitória de modo a atualizar o ranking.

```
Tabuleiro Jogador 2:
    |1 ||2 ||3 ||4 ||5 ||6 ||7 ||8 ||9 ||10|
|A ||  ||  ||  ||  ||+ ||+ ||+ ||+ ||  ||  |
|B ||  ||  ||  ||  ||  ||  ||  ||  ||+ ||  |
|C ||$ ||$ ||$ ||$ ||$ ||  ||+ ||  ||+ ||  |
|D ||  ||  ||  ||  ||  ||  ||+ ||  ||  ||+ |
|E ||  ||  ||  ||  ||  ||  ||+ ||  ||  ||+ |
|F ||$ ||$ ||  ||  ||  ||  ||+ ||  ||  ||+ |
|G ||  ||  ||  ||+ ||  ||+ ||  ||  ||  ||  |
|H ||$ ||$ ||  ||+ ||  ||+ ||  ||  ||  ||  |
|I ||  ||  ||  ||+ ||  ||+ ||  ||  ||  ||  |
|J ||  ||  ||  ||  ||  ||  ||  ||+ ||+ ||  |


Tabuleiro Jogador 1:
    |1 ||2 ||3 ||4 ||5 ||6 ||7 ||8 ||9 ||10|
|A ||  ||  ||$ ||$ ||$ ||x ||$ ||$ ||  ||  |
|B ||  ||  ||  ||  ||  ||  ||  ||  ||  ||  |
|C ||# ||# ||# ||  ||# ||  ||  ||  ||  ||  |
|D ||  ||  ||  ||  ||  ||  ||$ ||  ||  ||  |
|E ||  ||  ||  ||  ||  ||  ||$ ||  ||  ||$ |
|F ||  ||$ ||$ ||$ ||  ||  ||$ ||  ||  ||$ |
|G ||  ||  ||  ||  ||  ||  ||  ||  ||  ||  |
|H ||$ ||$ ||  ||  ||  ||  ||$ ||$ ||$ ||$ |
|I ||  ||  ||  ||  ||  ||  ||  ||  ||  ||  |
|J ||  ||  ||$ ||$ ||$ ||$ ||  ||  ||$ ||$ |

Escolha uma casa para bombardear (inserir letra e a seguir o numero, tudo junto) ->C4
Casa valida
Casa C4 bombardeada! Acertaste em:
PORTA_AVIOES
Destruiste o Barco!
Todos os barcos foram destruidos! Ganhaste!

Mais uma vitoria para ti!
```

Figura 21 - Vitória - projeto parte 1

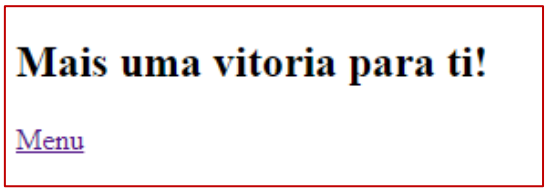

Figura 22 - Vitória - projeto parte 2

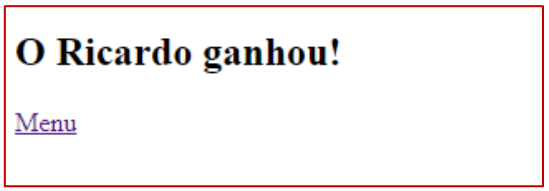

Figura 23 - Derrota - projeto parte 2

4. Conclusões

Com a realização da primeira parte deste projeto foi nos possível obter um maior conhecimento de como funciona não só o protocolo *TCP*, assim como a comunicação para com o cliente e o servidor, neste caso através de *XML* e validação dos mesmos com ficheiros *XSD*.

Já na segunda e última parte do projeto, através da matéria lecionada em aula e da conclusão do mesmo adquirido uma melhor fundamento e experiência em relação ao funcionamento de *JavaServer Pages* (JSP), assim como *Servelets.* Foi também utilizado o protocolo *HTML*, desde *tags*, estrutura, técnicas eficientes no envio de dados para o servidor (*GET* e *POST*) e *JavaScript* associado a essas mesmas *tags*. Sendo ainda reforçado novamente a importância dos protocolos e validações dos mesmos entre a troca de informação.

A utilização desta arquitetura em termos de segurança é relativa à informação que o *XML* contém, assim como a maneira como são tratados esses dados, que neste caso cada jogador envia apenas a sua informação e toda a sua lógica está dentro do servidor, permitindo assim uma maior segurança.

Existia um problema de segurança na medida em que os *forms* do *HTML* estavam a transmitir a password do utilizador para o url do mesmo (através do atributo *id*=””), que foi resolvido utilizando o método *POST*.

Relativamente à tolerância de falhas, como são utilizados ficheiros *XSD* do lado do servidor de modo a validar a informação que este recebe e envia, acabamos por ter uma aplicação que restringe muito bem a troca de informação entre o cliente e o servidor. Ainda mais quando na segunda parte do trabalho a maioria do *input* do utilizador é controlada por botões ou *forms* limitando assim falhas na nossa aplicação.

Uma das grandes vantagens desta arquitetura é também a utilização do protocolo TCP que permite a conexão de diferentes clientes para se jogarem diferentes jogos em simultâneo. Foi implementado também com sucesso a procura por um jogador através do *AutoComplete* e entrar na sala que esse jogador cria.

A respeito à compatibilidade com os browsers mais comuns, a aplicação foi testada no *Google Chrome*, *Firefox*, *Microsoft Edge* e *Brave* com sucesso em todos, seja entre perfis ou até mesmo no jogo.

A melhorar na nossa aplicação seria quando um cliente desconecta se, o jogador adversário ficar a saber do acontecimento, produzindo um erro quando trocar a vez de jogar. Adicionalmente, quando o servido vai abaixo, para além de se perder as conexões com os clientes, os jogos a decorrer, são perdidos. Poderia ser melhorado também o *design* e estrutura visual da aplicação e o problema em relação a caminhos relativos ao tentar aceder e guardar ficheiros.

Por fim, ao nosso ver, esta aplicação fica apenas a desejar relativamente a como a informação é guardada e enviada (por *XML*). A utilização de uma base de dados facilitaria muitos processos de implementação assim como de uso, tornando a experiência tanto do utilizador, como do programador melhor.