

Centro Universitário das Américas – FMU  
Sistemas da Informação

Breno Magalhães Gonçalves 2939184

Gabriel Canha Santaella 2408209

Nadine Lima Marques 1814343

William Saturnino Marques 2441705

São Paulo

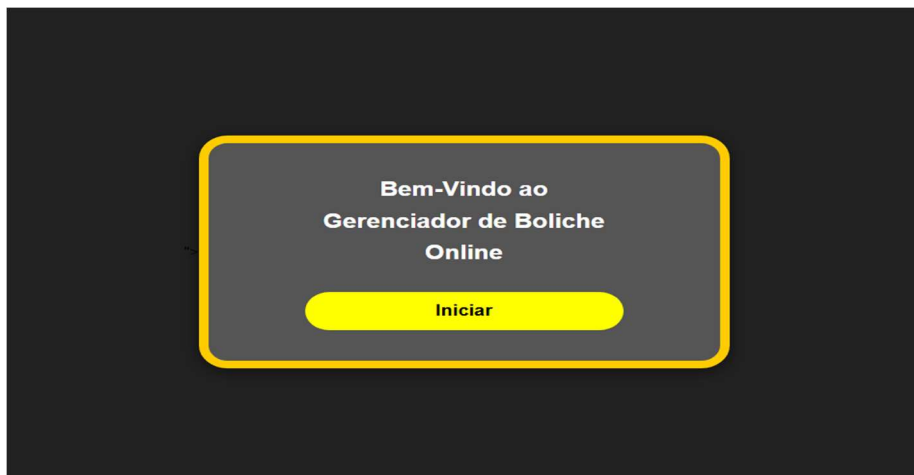
2025

## Introdução

O boliche, ao longo dos anos, tem se consolidado como uma das atividades recreativas mais apreciadas por pessoas de diferentes idades, estilos de vida e perfis sociais. Muito mais do que apenas um esporte competitivo, o boliche é também uma excelente forma de lazer e socialização, sendo frequentemente escolhido como opção para confraternizações entre amigos, reuniões familiares, encontros casuais e até mesmo eventos corporativos. Diante desse cenário de popularidade crescente, e visando unir entretenimento e tecnologia, desenvolvemos um projeto inovador: um gerenciador de partidas de boliche totalmente online, utilizando as linguagens fundamentais da web HTML, CSS e JavaScript.

A proposta deste sistema é transformar a experiência tradicional do boliche em algo ainda mais acessível, moderno e automatizado. Ao integrar a tecnologia ao ambiente do jogo, conseguimos criar uma plataforma que não apenas substitui o gerenciamento manual das partidas, mas também o aprimora significativamente. Com uma interface visualmente atrativa, intuitiva e responsiva, o gerenciador oferece ao usuário uma experiência clara, direta e altamente funcional, permitindo o acompanhamento em tempo real de informações importantes como a rodada atual, pontuação de cada jogador, ordem de jogadas e até mesmo o ranking final após o término da partida.

## 1- Tela Inicial



A primeira tela que o jogador terá contato é a tela de início, de boas-vindas, no qual utilizamos o arquivo index.html e o style.css pra estilizar a plataforma.

Código HTML:


```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Gerenciador de Boliche</title>
  <link rel="stylesheet" href="css.style/style.css">
</head>
<body>
  <main class="container">
    <h1>Bem-Vindo ao <br>Gerenciador de Boliche Online</h1>
    <button type="submit" class="login" onclick="iniciarCadastro()">Iniciar</button>
  </main>
</body>
</html>

<script src="main.js"></script>
</body>
</html>
```

Classificamos cada tag para facilitar a estilização no css e utilizamos o JavaScript para dar funcionalidade do botão iniciar pra trocar a tela assim que o usuário clicar.


## 2- Estrutura do JS utilizado nessa etapa com nomenclatura MAIN:

```

JS main.js >  adicionarJogador
1  function iniciarCadastro() {
2      window.location.href = 'cadastro.html';
3  }
4

```

### 3- Tela de edição e inserção de dados.



Utilizamos o placeholder para o campo de digitação do usuário e criamos buttons para a edição desses dados.

## Estrutura de Cadastro do Jogador com HTML:

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Gerenciador de Boliche 🍷</title>
  <link rel="stylesheet" href="./css.style/cadastro.css">
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.5.0/css
</head>
<body>
  <div class="container">
    <div class="box">
      <h1>Gerenciador de Boliche 🍷</h1>
      <input type="text" id="nomeJogador" placeholder="Digite o nome do jogador">
      <div id="mensagemErro" class="mensagem-erro"></div>
      <button onclick="adicionarJogador()">Adicionar</button>

      <ul id="listaJogadores"></ul>

      <div class="configuracoes">
        <label for="numRodadas">Número de Rodadas:</label>
        <input type="number" id="numRodadas" min="1" max="10" value="10">
      </div>

      <button onclick="concluirCadastro()" class="concluir">Concluir</button>
    </div>
  </div>

  <script src="main.js"></script>
```

Criamos uma div com mensagem de erro para melhor visualização caso o jogador não siga as regras básicas, a seguir a estrutura do java:

```
let jogadores = [];

function adicionarJogador() {
  const nome = document.getElementById('nomeJogador').value.trim();
  const mensagemErro = document.getElementById('mensagemErro');
  mensagemErro.textContent = '';

  if (!nome) {
    mensagemErro.textContent = 'Digite um nome válido.';
    return;
  }
  if (! /^[A-Za-zÀ-ÿ\s]+$/.test(nome)) {
    mensagemErro.textContent = 'Digite um nome válido (apenas letras).';
    return;
  }
  if (jogadores.length >= 6) {
    mensagemErro.textContent = 'Máximo de 6 jogadores por pista.';
    return;
  }

  jogadores.push(nome);
  document.getElementById('nomeJogador').value = '';
  atualizarLista();
}
```

A mensagem de erro está vazia pois a resposta está em cada `textContent` caso o usuário não obedeça às regras bases. Se for digitado um 'espaço' a mensagem de erro solicitando nome válido aparece.

Importante ressaltar que foi criado uma variável global de jogadores pois utilizamos ela em mais de uma função do Java.

Se for digitado simbologia, números ou qualquer coisa diferente de letras o gerenciador também não aceita e solicita um nome válido com letras.

Se inserir mais de seis jogadores, e utilizamos o `length` para contagem da lista, o gerenciador retorna que a capacidade de jogadores foi alcançada.

Utilizamos a `.push` fora da estrutura `if` caso o usuário tenha seguido as regras e adicionamos o nome na lista.

#### 4- Função atualizar e editar lista:

```
function atualizarLista() {
  const ul = document.getElementById('listaJogadores');
  ul.innerHTML = '';
  jogadores.forEach((jogador, index) => {
    const li = document.createElement('li');
    li.innerHTML = `
      ${index + 1}. <span>${jogador}</span>
      <button onclick="editarJogador(${index})">✎</button>
      <button onclick="removerJogador(${index})">✖</button>
    `;
    ul.appendChild(li);
  });
}

function editarJogador(index) {
  const novoNome = prompt("Editar nome do jogador:", jogadores[index]);
  if (novoNome && novoNome.trim() !== "") {
    jogadores[index] = novoNome.trim();
    atualizarLista();
  }
}

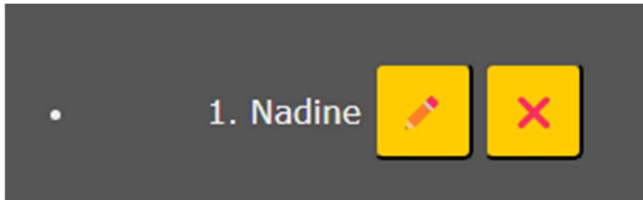
function removerJogador(index) {
  if (confirm(`Deseja remover o jogador ${jogadores[index]}?`)) {
    jogadores.splice(index, 1);
    atualizarLista();
  }
}
```

Criamos uma lista não ordenada (`ul`) na lista jogadores do HTML e apagamos para garantir que as linhas `li` sejam apenas com o conteúdo que o usuário irá inserir.

Utilizamos função `forEach` para percorrer o array `Jogadores` pelo índice = `index`.

Utilizamos o `index+1` para mostrar a posição correta do jogador, tendo em vista que o índice padrão de arrays começa com 0.

Criamos também os botões de editar e remover o jogador, e ele aparecerá ao lado da li que criamos usando o `append.child`.



## 5- Botões de editar e adicionar jogadores.

```
function editarJogador(index) {
  const novoNome = prompt("Editar nome do jogador:", jogadores[index]);
  if (novoNome && novoNome.trim() !== "") {
    jogadores[index] = novoNome.trim();
    atualizarLista();
  }
}

function removerJogador(index) {
  if (confirm(`Deseja remover o jogador ${jogadores[index]}?`)) {
    jogadores.splice(index, 1);
    atualizarLista();
  }
}
```

Utilizamos o índice para acessar a lista dos jogadores e dar funcionalidade aos botões criados na etapa anterior.

O remover jogador utiliza a funcionalidade `splice` para remover jogador além da funcionalidade `if` para o usuário confirmar com certeza que deseja fazer a remoção.

Função concluir cadastro:

```
function concluirCadastro() {
  const rodadas = parseInt(document.getElementById('numRodadas').value);
  if (!rodadas || rodadas <= 0) return alert('Informe um número válido de rodadas.');
```

```
  if (jogadores.length < 1) return alert('Adicione pelo menos um jogador.');
```

```
  localStorage.setItem('jogadores', JSON.stringify(jogadores));
  localStorage.setItem('rodadas', rodadas);
  window.location.href = 'tabela.html';
}
```

Utilizamos a const ao invés de uma variável let pois não iremos mudar após definir o seu valor.

Se o usuário tentar iniciar a partida sem adicionar jogadores o código retorna uma mensagem de erro solicitando pelo menos um jogador.

Utilizamos o localStorage para armazenar os jogadores no navegador e usamos o json.stringify para desfazer o array e transformar em texto e assim poder usar o localStorage.

## 6- Atualização de Interface

```
const jogadores = JSON.parse(localStorage.getItem('jogadores')) || [];  
const numRodadas = parseInt(localStorage.getItem('rodadas')) || 10;  
  
let rodadaAtual = 1;  
let indiceJogador = 0;  
let jogadas = jogadores.map(() => Array(numRodadas).fill({ pontos: [], total: 0 }));  
  
function atualizarInterface() {  
  document.getElementById('rodadaAtual').innerText = rodadaAtual;  
  document.getElementById('vezJogador').innerText = jogadores[indiceJogador];  
  atualizarTabela();  
}
```

Utilizamos variáveis globais pois será usadas nas demais functions.

A const jogadores busca o nome dos jogadores no localStorage e se não tiver nada retorna vazio, e const rodadas busca o número de rodadas que deve ser pré estabelecido na etapa anterior e caso não esteja parametrizado ele usa 10 como padrão.

No let jogadas cada jogador possui um array específico com o numRodadas.

A função atualizarInterface apresenta a rodada atual e o jogador da vez.



## 7- Função atualizar tabela:

Código HTML e JavaScript:

```
<html lang="pt-BR">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Tabela - Partida de Boliche</title>
  <link rel="stylesheet" href="/css.style/tabela.css">
  <script defer src="tabela.js"></script>
</head>
<body>
  <div class="container">
    <h1>Partida em andamento 🎳</h1>
    <h2>Rodada <span id="rodadaAtual"></span></h2>
    <h2>Vez de: <span id="vezJogador"></span></h2>
    <div id="jogada"></div>
    <div id="tabelaPontuacao"></div>
  </div>
</body>
</html>
```

```
function atualizarTabela() {
  const tabela = document.getElementById('tabelaPontuacao');
  let html = '<table border="1">';
  html += '<tr><th>Jogador</th>';
  for (let r = 1; r <= numRodadas; r++) {
    const classe = r > rodadaAtual ? 'class="oculto"' : '';
    html += `<th ${classe}>R${r}</th>`;
  }
  html += '<th>Total</th></tr>';

  jogadores.forEach((jogador, i) => {
    html += `<tr><td>${jogador}</td>`;
    let total = 0;
    for (let r = 0; r < numRodadas; r++) {
      const jogada = jogadas[i][r];
      const pontos = jogada.pontos.join(', ');
      total += jogada.total;
      const classe = r + 1 > rodadaAtual ? 'class="oculto"' : '';
      html += `<td ${classe}>${pontos}</td>`;
    }
    html += `<td>${total}</td></tr>`;
  });
  html += '</table>';
  tabela.innerHTML = html;
}
```

Utilizamos a div que está no HTML (Tabela Pontuação). Criamos a primeira tabela com o <tr> e fazemos a edição do cabeçalho com <th> a titulação fica dessa forma:

Partida em andamento 🎮		
Rodada 1		
Vez de: nadine		
Jogador	R1	Total
nadine		0

Ao final da função fechamos a tabela com o </table> e inserimos ela no html com o innerHTML.

Ademais, criamos a class oculto para ocultar as rodadas que ainda não ocorreram, junto com css.

```
function registrarJogada(j1, j2) {
  jogadas[indiceJogador][rodadaAtual - 1] = {
    pontos: j1 === 10 ? [10] : [j1, j2],
    total: j1 + j2
  };

  indiceJogador++;
  if (indiceJogador >= jogadores.length) {
    indiceJogador = 0;
    rodadaAtual++;
  }

  if (rodadaAtual > numRodadas) {
    document.getElementById('vezJogador').innerText = 'Partida encerrada 🏆';
    return;
  }
  atualizarInterface();
}

atualizarInterface();
```

A função registra jogada tem os parâmetros (j1 e j2), e registra a jogada do jogador atual e calcula o total de pontos.

Colocamos o código caso o jogador faça um strike no pontos: j1 === 10 ? [10] : [j1, j2], do contrário será contabilizado com uma jogada comum.

Avançará para o próximo jogador usando o indiceJogador ++, e se o indiceJogador ultrapassar o número de jogadores, ele volta ao primeiro jogador (indiceJogador = 0) e irá para a próxima rodada (rodadaAtual++).

Utilizamos mais um if para encerrar o jogo, se a rodada atual for maior que o número total de rodas estabelecidas anteriormente.

Utilizamos o textContent para apresentar a mensagem de partida encerrada.

## 8- Utilização do CSS no projeto:

```
<link rel="stylesheet" href="assets/style-tabela.css">
```

Utilizamos dessa maneira fazendo com que a linha ligasse um arquivo CSS ao arquivo HTML. CSS (Cascading Style Sheets) é a linguagem usada para estilizar uma página, ou seja, mudar cores, tamanhos, fontes, bordas, etc.

```
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Gerenciador de Boliche 🏠</title>
  <link rel="stylesheet" href="assets/style-index.css">
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.5.0/css/all.min.css">
</head>
```

Dentro da tag <head>, tem duas linhas que ligam folhas de estilo externas

A aplicação de estilos foi realizada por meio da inclusão de duas folhas de estilo no documento HTML principal, conforme descrito abaixo:

Foi incluído um arquivo de estilo localizado na pasta “assets”, através da seguinte linha de código:

```
<link rel="stylesheet" href="assets/style-tabela.css">
```

Este arquivo contém regras de estilo que afetam diretamente os elementos da interface, como: Centralização de conteúdo com container e box a personalização de botões (button), campos de entrada (input) e listas (ul, li). Cores, espaçamentos, fontes e bordas dos componentes.

Essas definições são fundamentais para que o sistema apresente um layout organizado e intuitivo para o usuário final.

Além da folha de estilo local, também foi utilizada a biblioteca externa Font Awesome:

```
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.5.0/css/all.min.css">
```

Esta biblioteca foi incorporada com o objetivo de adicionar ícones vetoriais à interface, que contribuem para uma navegação mais visual e interativa, como ícones de adição, exclusão e edição de jogadores.