**Desenvolvimento de um sistema web para cadastro e ordenação de estudantes**

**[[1]](#footnote-1)**

Development of a web system for student registration and ordering

Felipe Silveira Pessoa

Faculdade de Tecnologia de Franca – Fatec Franca

[felipe.pessoa01@fatec.sp.gov.br](mailto:felipe.pessoa01@fatec.sp.gov.br)

Israel Fonseca Pessoni

Faculdade de Tecnologia de Franca – Fatec Franca

[israel.pessoni@fatec.sp.gov.br](mailto:israel.pessoni@fatec.sp.gov.br)

**Resumo**

Este artigo apresenta o desenvolvimento de um sistema web para cadastro e ordenação de estudantes, implementado com HTML, CSS e JavaScript. O objetivo principal é possibilitar o gerenciamento eficiente de registros acadêmicos, garantindo integridade dos dados e clareza na exibição das informações. A aplicação contempla funcionalidades como validação de entradas, prevenção de cadastros duplicados, limitação de registros, exibição dinâmica em tabelas e ordenação utilizando o algoritmo Selection Sort. O sistema foi estruturado com requisitos funcionais e não funcionais que asseguram usabilidade, segurança e desempenho adequados. A metodologia adotada permitiu a implementação de diferentes formas de ordenação e filtros, além da persistência local dos dados. Os resultados demonstram que o sistema atende ao propósito educacional, promovendo a prática de algoritmos em um contexto aplicado e contribuindo para a aprendizagem de lógica de programação. Conclui-se que a solução proposta é simples, acessível e eficaz, podendo ser expandida para futuras integrações e funcionalidades. Assim, reforça-se sua relevância como recurso acadêmico e tecnológico.

# Palavras-chave: CADASTRO DE ESTUDANTES; ORDENAÇÃO; ALGORITIMO; PROGRAMAÇÃO WEB; SELEÇÃO.

**Abstract**

This article presents the development of a web system for student registration and ordering, implemented with HTML, CSS and JavaScript. The main objective is to enable the efficient management of academic records, ensuring data integrity and clarity in the display of information. The application includes features such as input validation, prevention of duplicate registrations, registration limitation, dynamic display in tables and sorting using the Selection Sort algorithm. The system has been structured with functional and non-functional requirements that ensure adequate usability, security and performance. The methodology adopted allowed the implementation of different forms of sorting and filters, in addition to the local persistence of data. The results demonstrate that the system meets the educational purpose, promoting the practice of algorithms in an applied context and contributing to the learning of programming logic. It is concluded that the proposed solution is simple, accessible and effective, and can be expanded for future integrations and functionalities. Thus, its relevance as an academic and technological resource is reinforced.

**Keywords:** STUDENT REGISTRATION; SORTING; ALGORITHM; WEB PROGRAMMING; SELECTION SORT.

1. **INTRODUÇÃO**

Este relatório apresenta o desenvolvimento de um sistema de cadastro e ordenação de estudantes, utilizando as tecnologias web JavaScript (JS), HTML e CSS.  O gerenciamento de dados acadêmicos, mesmo em pequena escala, exige mecanismos eficientes de controle e organização para garantir a clareza e a integridade das informações. Diante disso, o problema central abordado por este trabalho é: **Como desenvolver uma aplicação web front-end simples e didática que implemente funcionalidades essenciais de gerenciamento de dados, como validação, controle de duplicidade e, principalmente, ordenação de registros, utilizando um algoritmo fundamental de Estrutura de Dados?**

O sistema tem como finalidade possibilitar o cadastro de estudantes, garantindo integridade dos dados, organização e clareza na exibição das informações. Além disso, buscou-se aplicar um algoritmo de ordenação para demonstrar conceitos fundamentais de lógica de programação.

A **justificativa** para este estudo reside na importância do ensino prático de algoritmos. O desenvolvimento deste sistema oferece um ambiente prático onde o estudante pode visualizar a manipulação de dados e o funcionamento do algoritmo **Selection Sort** em tempo real.

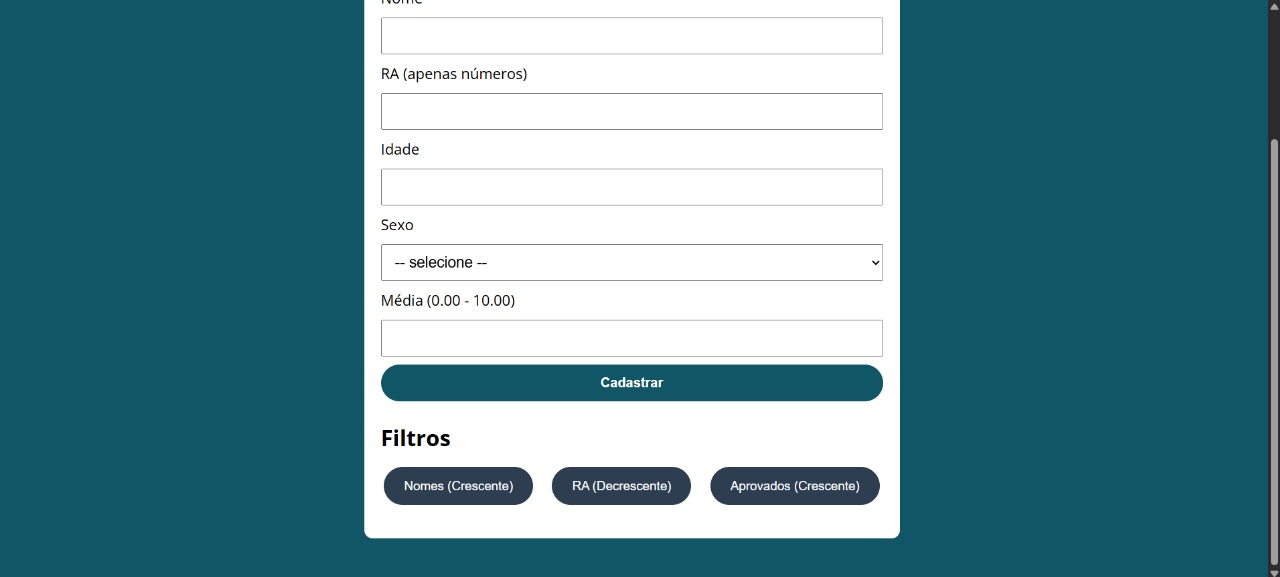
O projeto foi construído em três camadas principais: HTML para a estrutura, CSS para a estilização e JavaScript para a lógica de funcionamento, conforme será detalhado na seção de Método. As funcionalidades implementadas incluem o cadastro com verificação de dados obrigatórios, validação para impedir duplicidade, limite máximo de estudantes e a ordenação via algoritmo Selection Sort. A interface principal é demonstrada pela Figura 1 e Figura 2, e construída pelas principais linhas de código HTML, dadas pelo Quadro 1, e as linhas de código CSS, dadas pelo Quadro 2.

Figura 1: Interface de Cadastro de Alunos (HTML/CSS)



Fonte 1: Elaboração dos Autores (2025).

Figura 2: Interface de Filtros do Cadastro dos Alunos (HTML/CSS)

****

Fonte 2: Elaboração dos Autores (2025).

Quadro 1 – Principais marcações do HTML para o Formulário de Cadastro

|  |
| --- |
| <section class="container">  <h1><i>CADASTRO ALUNOS FATEC </i></h1>  <p>Preencha os dados do aluno e clique em "Cadastrar".</p>  <!--====CADASTRO ====== -->  <form id="formulario" autocomplete="off" method="get">  <label for="nome">Nome</label>  <input type="text" id="nome\_HTML" required>  <label for="ra">RA (apenas números)</label>  <input type="number" id="ra\_HTML" required>  <label for="idade">Idade</label>  <input type="number" id="idade\_HTML" min="0" required>  <label for="sexo">Sexo</label>  <select id="sexo\_HTML" required>  <option value="">-- selecione --</option>  <option value="M">Masculino</option>  <option value="F">Feminino</option>  <option value="O">Outro</option>  </select>  <label for="media">Média (0.00 - 10.00)</label>  <input type="number" id="media\_HTML" step="0.01" min="0" max="10" required>  <button id="botao-cadastrar\_HTML" type="submit" >Cadastrar</button>  </form>  <!--======FILTRO ====== -->  <h2 style="margin-top:30px;"> Filtros </h2>  <div class="filtros">  <button id="nomes\_crescente\_HTML" type="button">Nomes (Crescente)</button>  <button id="ra\_decrescente\_HTML" type="button">RA (Decrescente)</button>  <button id="aprovados\_crescente\_HTML" type="button">Aprovados (Crescente)</button>  </div> |

Fonte 3: Elaboração dos Autores (2025)

Quadro 2 – Principais marcações do CSS para o Formulário de Cadastro

|  |
| --- |
| @import url('https://fonts.googleapis.com/css?family=Open+Sans:400,700&display=swap');  :root { --primary-color: rgb(17, 86, 102); --primary-color-darker: rgb(9, 48, 56);}  \* { box-sizing: border-box; outline: 0;}  body { margin: 0; padding: 0; background: var(--primary-color); font-family: 'Open sans', sans-serif; font-size: 1.1em; line-height: 1.5em;}  .container {max-width: 640px; margin: 50px auto; background: #fff; padding: 20px; border-radius: 10px;}  form input, form label, form button, form select {display: block; width: 100%;margin-bottom: 10px;}  form input, form select {font-size: 18px; height: 44px; padding: 0 12px;}  form input:focus, form select:focus { outline: 1px solid var(--primary-color);}  form button { border: none; background: var(--primary-color); color: #fff; font-size: 16px;  font-weight: 700; height: 44px; cursor: pointer; margin-top: 10px; border-radius: 25px;  transition: all 0.2s ease-in-out;}  form button:hover { background: var(--primary-color-darker); transform: scale(1.05);}  .filtros { display: flex; justify-content: space-around; flex-wrap: wrap; gap: 15px; margin-top: 15px;}  .filtros button { border: none; background: #2c3e50; color: white; padding: 14px 24px;  cursor: pointer; border-radius: 50px; font-size: 15px; transition: all 0.25s ease-in-out;}  .filtros button:hover { background: #1a252f; transform: scale(1.1);} |

Fonte 4: Elaboração dos Autores (2025).

# FUNDAMENTAÇÃO TEÓRICA

O desenvolvimento de sistemas interativos, como o apresentado neste trabalho, fundamenta-se na interseção de três pilares da ciência da computação: estruturas de dados, algoritmos e tecnologias de interface. A gestão de informações, mesmo em pequena escala, exige uma organização lógica que permita a manipulação eficiente dos dados. Para este sistema, a estrutura de dados central é um array (vetor), que armazena os registros dos estudantes em memória.

Para a tarefa de organização, foram empregados algoritmos de ordenação, que são um campo clássico de estudo e essenciais para otimizar a recuperação e apresentação de informações. Dentre as diversas opções, o algoritmo Selection Sort foi o escolhido. Sua lógica operacional consiste em iterar pela lista, selecionar o menor (ou maior) elemento e trocá-lo com o elemento na primeira posição. O processo é repetido para a sub-lista restante, garantindo que, ao final, a coleção esteja ordenada. Embora sua complexidade de tempo seja quadrática, O(n²), o que o torna ineficiente para grandes volumes de dados, sua implementação é notavelmente simples e de baixo custo de memória, sendo uma escolha ideal para cenários didáticos e aplicações com um conjunto de dados limitado, como o proposto neste projeto.

A camada de interação com o usuário foi construída sobre o tripé de tecnologias web front-end. O HTML (HyperText Markup Language) fornece a estrutura semântica do conteúdo, o CSS (Cascading Style Sheets) é responsável pela estilização e apresentação visual, e o JavaScript implementa a lógica de programação, a manipulação de eventos e a interação dinâmica com o usuário. A integração dessas tecnologias é o que permite a criação de uma aplicação web funcional, acessível e responsiva.

Com base neste suporte teórico, o Capítulo 3 detalha a metodologia de desenvolvimento, e o Capítulo 4 (Análise e Resultados) ilustrará a comprovação empírica da eficácia do sistema, através de Figuras que demonstram o fluxo de trabalho, desde a inserção de dados e validações, até a organização final dos registros pelos filtros de ordenação. Juntamente, com suas linhas de código-fonte dadas através dos Quadros.

1. **MÉTODO**

O sistema foi estruturado em três camadas principais: HTML, responsável pela estrutura da interface; CSS, utilizado para a estilização e apresentação visual; e JavaScript, que implementa toda a lógica de funcionamento. Para a organização dos registros, foi adotado o algoritmo Selection Sort. A escolha se justifica pela simplicidade de implementação e baixo custo de memória, sendo adequado para a proposta do sistema.

As funcionalidades implementadas foram guiadas por um conjunto de requisitos funcionais e não funcionais, sendo eles:

* 1. **Requisitos Funcionais:**

Para uma organização sistemática, os requisitos funcionais foram estruturados em cinco módulos distintos, agrupados por área de funcionalidade. O primeiro deles, detalhado na Tabela 1, compreende os requisitos referentes ao cadastro de alunos:

Tabela 1- Cadastro de alunos

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Descrição | Tipo | Prioridade |
| RF001 | Permitir o cadastro de aluno com campos: nome, RA, idade, sexo e média. | Entrada/Armazenamento | Alta |
| RF002 | Exigir RA numérico e único; impedir cadastro com RA já existente. | Validação/Controle | Alta |
| RF003 | Validar limites dos campos: idade >= 0; média entre 0.00 e 10.00; nome não vazio; sexo selecionado. | Validação | Alta |
| RF004 | Permitir reset do formulário e foco no campo nome após cadastro. | Usabilidade | Média |
| RF005 | Limitar cadastro ao máximo configurável (ex.: 10 registros) e notificar usuário ao atingir limite. | Controle | Média |
| RF006 | Calcular automaticamente o resultado (Aprovado/Reprovado) com critério média >= 6.00 = Aprovado. | Processamento/Regra | Alta |
| RF007 | Exibir mensagem de erro amigável quando dados obrigatórios estiverem faltando ou inválidos. | Usabilidade/Validação | Alta |
| RF008 | Não inserir registros inválidos; em caso de entrada inválida, não alterar a lista existente. | Controle | Alta |
| RF009 | Permitir remoção manual de um aluno cadastrado (caso implementado). | Processamento/Controle | Média |

Fonte 5: Elaboração dos autores (2025)

A Tabela 2, por sua vez, detalha os requisitos funcionais do módulo de 'Ordenação e Filtros', especificando as regras para a manipulação e exibição dos dados:

Tabela 2- Ordenação e Filtros

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Descrição** | **Tipo** | **Prioridade** |
| RF010 | Ordenar lista por nome em ordem crescente (botão "Nomes (Crescente)"). | Saída/Processamento | Média |
| RF011 | Ordenar lista por RA em ordem decrescente (botão "RA (Decrescente)"). | Saída/Processamento | Média |
| RF012 | Filtrar e ordenar apenas alunos aprovados em ordem alfabética (botão "Aprovados (Crescente)"). | Saída/Processamento | Média |
| RF013 | Permitir futura adição de filtros (idade, faixa de média) sem alterar a estrutura base. | Extensibilidade | Baixa |

Fonte 6: Elaboração dos autores (2025)

O módulo de 'Exibição e Relatório', cujos requisitos estão detalhados na Tabela 3, estabelece as diretrizes para a correta apresentação visual dos dados e para a interação do sistema com o usuário.

Tabela 3- Exibição e Relatório

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Descrição** | **Tipo** | **Prioridade** |
| RF014 | Exibir tabela com colunas: Nome, RA, Idade, Sexo, Média, Resultado. | Saída | Alta |
| RF015 | Mostrar mensagem clara quando não houver alunos cadastrados. | Usabilidade/Saída | Média |
| RF016 | Permitir exportação simples para CSV (download) com os registros atuais. | Saída/Integração | Média |
| RF017 | Suportar exibição responsiva (rolagem/overflow) quando a lista for longa. | Usabilidade | Média |

Fonte 7: Elaboração dos autores (2025)

As especificações para a 'Persistência e Integração' dos dados são detalhadas na Tabela 4. Estes requisitos definem como as informações são salvas para persistir entre sessões de uso e preveem mecanismos de interoperabilidade, como importação e exportação:

Tabela 4- Persistência e Integração

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Descrição** | **Tipo** | **Prioridade** |
| RF018 | Persistir dados localmente (ex.: localStorage) para manter lista entre sessões do navegador. | Armazenamento | Média |
| RF019 | Permitir importação/exportação de dados em JSON/CSV para backup e intercâmbio. | Entrada/Saída | Baixa |
| RF020 | Prever ponto de integração com API/servidor para salvar dados remotamente (opcional). | Integração | Baixa |

Fonte 8: Elaboração dos autores (2025)

Concluindo a especificação dos módulos, a Tabela 5 apresenta os requisitos de 'Segurança, Acessibilidade e Usabilidade'. Estes são critérios transversais que garantem não apenas a proteção do sistema, mas também uma interação intuitiva e compatível com tecnologias assistivas.

Tabela 5- Segurança, Acessibilidade e Usabilidade

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Descrição** | **Tipo** | **Prioridade** |
| RF021 | Garantir foco acessível e labels corretos para campos (compatível com leitores de tela). | Usabilidade/Acessibilidade | Alta |
| RF022 | Evitar execução de scripts pela entrada do usuário (sanitização de inputs) ao exibir dados. | Segurança | Alta |
| RF023 | Fornecer mensagens e alertas claros para ações do usuário (cadastro, erro, limite atingido). | Usabilidade | Média |

Fonte 9: Elaboração dos autores (2025)

* 1. **Requisitos Não Funcionais:**

Por fim, a Tabela 6 detalha os requisitos não funcionais, que definem os atributos de qualidade e as condições de funcionamento do sistema, em vez de suas funcionalidades diretas:

Tabela 6- Requisitos não funcionais

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Tipo** | **Descrição** | **Prioridade** |
| RNF001 | Usabilidade | Interface simples e intuitiva; cadastro ágil com validações inline. | Alta |
| RNF002 | Desempenho | Operações de exibição, ordenação e validação devem ser instantâneas para até 100 registros (<1s). | Alta |
| RNF003 | Confiabilidade | Dados persistidos localmente devem sobreviver recarregamento do navegador; mecanismos básicos de recuperação. | Média |
| RNF004 | Segurança | Sanitização de entradas para evitar XSS; não armazenar dados sensíveis desnecessários. | Alta |
| RNF005 | Compatibilidade | Funcionar nos navegadores modernos (Chrome, Edge, Firefox) e em versões mobile responsivas. | Média |
| RNF006 | Manutenibilidade | Código organizado e comentado, facilitando futuras extensões (import/export, integração servidor). | Média |
| RNF007 | Acessibilidade | Conformidade básica com WCAG: labels, foco, contrastes aceitáveis e navegação por teclado. | Alta |
| RNF008 | Portabilidade | Layout responsivo para uso em desktop e tablet; formulário utilizável em telas menores com rolagem. | Média |
| RNF009 | Custos | Solução simples baseada em web (HTML/CSS/JS) sem dependências pagas para protótipo. | Alta |

Fonte 10: Elaboração dos autores (2025)

# 4. ANÁLISE E RESULTADOS

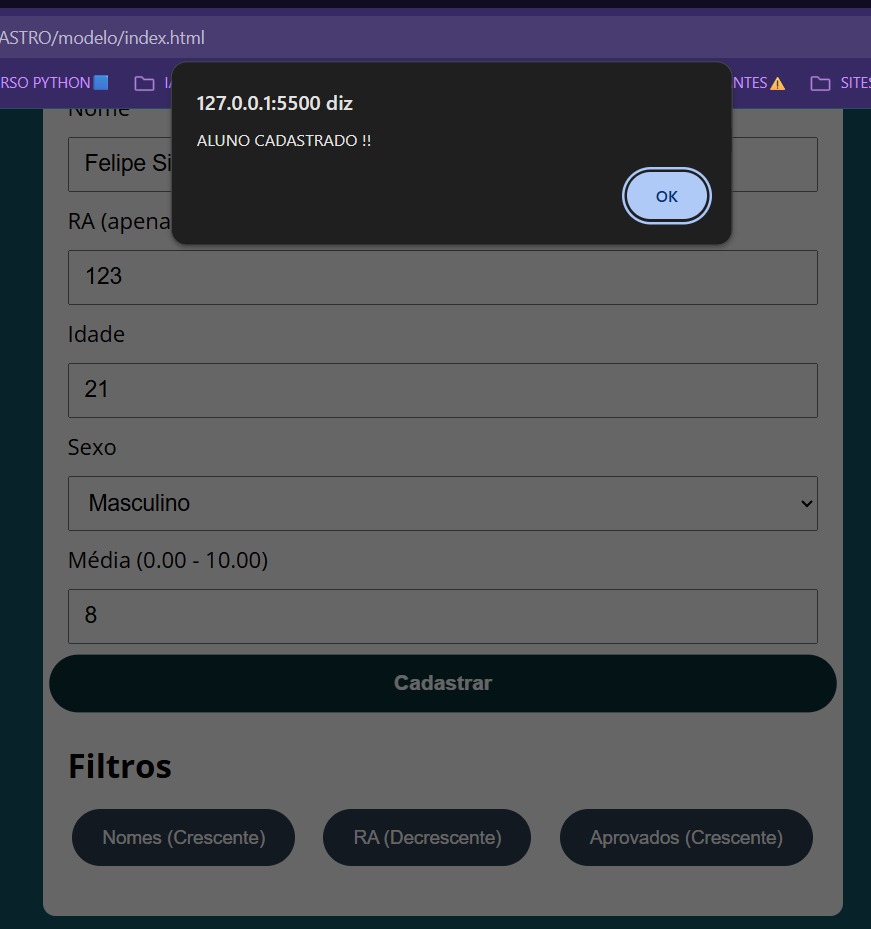
O sistema desenvolvido atende com sucesso aos requisitos propostos, funcionando como uma aplicação prática para o gerenciamento de dados acadêmicos. Os usuários podem cadastrar, visualizar e organizar os estudantes de forma eficiente, garantindo a integridade das informações inseridas.

Além disso, a implementação do cadastro de alunos demonstrou-se robusta, com as validações de campos obrigatórios, limites de valores e unicidade do RA operando conforme o esperado. Ao atingir o limite de 10 alunos, o sistema notifica o usuário, prevenindo a sobrecarga da lista. A exibição dos dados em tabela é clara, e o cálculo automático do resultado do aluno (Aprovado/Reprovado) funciona corretamente, adicionando uma regra de negócio simples e útil.

Ademais, aplicação do algoritmo Selection Sort foi bem-sucedida e permitiu a implementação das três funcionalidades de ordenação propostas: por nome (crescente), por RA (decrescente) e por alunos aprovados (crescente). Embora não seja o algoritmo mais performático, para o limite de registros definido, seu desempenho é instantâneo, alinhado ao requisito não funcional de performance. O sistema cumpre, assim, seu propósito educacional de demonstrar a aplicação de um algoritmo de ordenação em um contexto real.

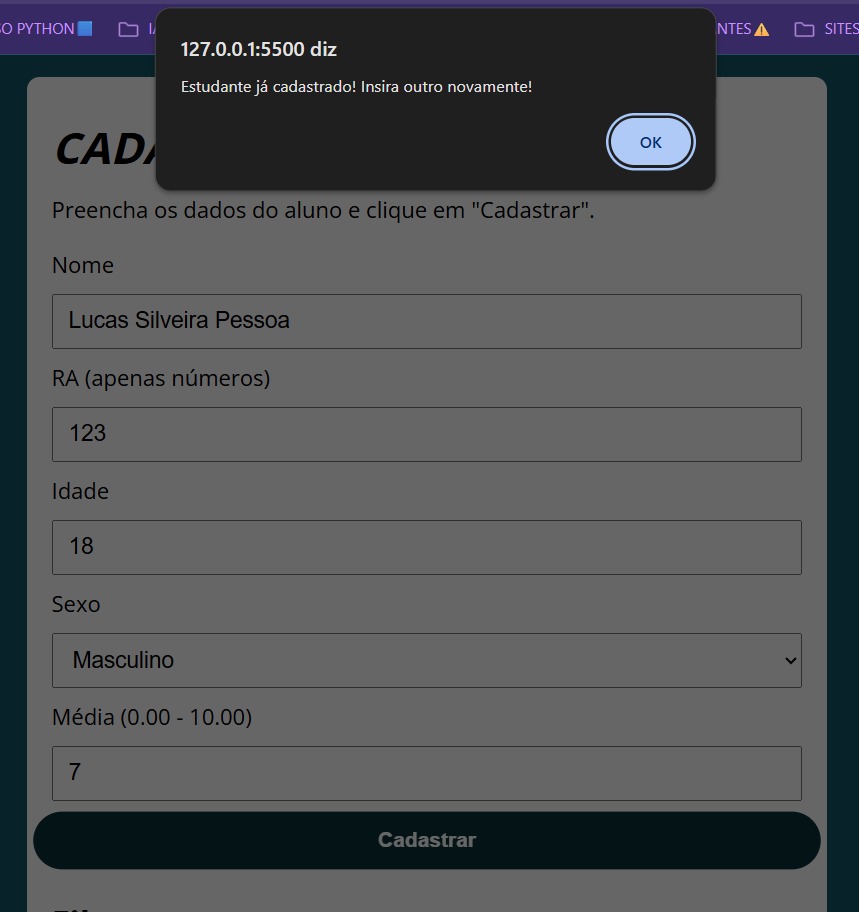
A seguir, a comprovação do funcionamento do sistema é detalhada através de Figuras e Quadros, correlacionando o resultado prático com os Requisitos Funcionais (RF) e Não Funcionais (RNF) definidos na Seção 3.

Figura 3: Cadastro de aluno novo



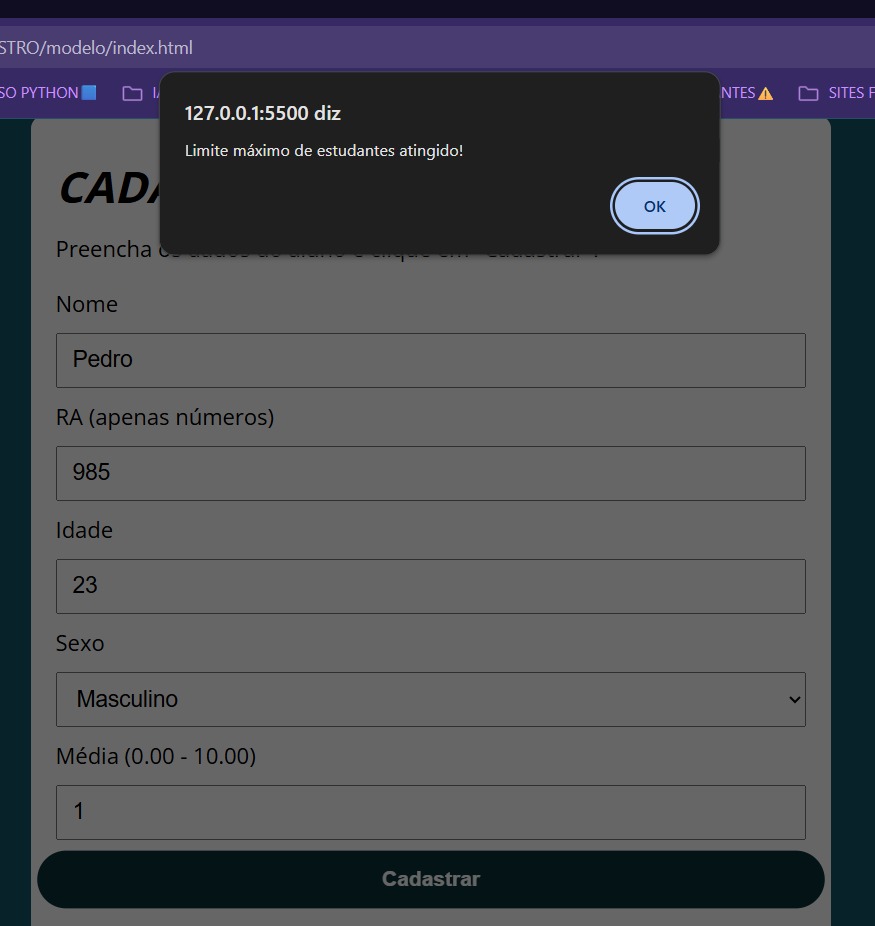
Fonte 11: Elaboração dos Autores (2025).

Figura 4: Cadastro Aluno RA ERRADO – RA já existe no Sistema



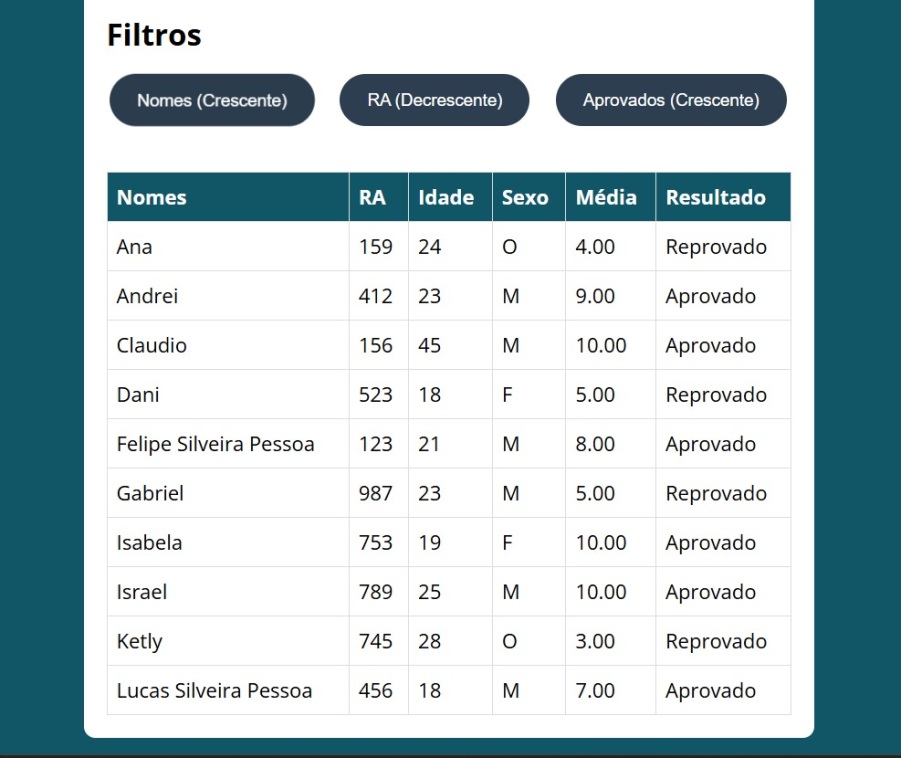
Fonte 12: Elaboração dos Autores (2025).

Figura 5: Limite máximo de cadastro atingido



Fonte 13: Elaboração dos Autores (2025).

Figura 6: Filtros - Nomes (Crescentes)



Fonte 14: Elaboração dos Autores (2025).

Figura 7: Filtros - Nomes (Crescentes)



Fonte 15: Elaboração dos Autores (2025).

Figura 8: Filtros – Aprovados (Crescente)



Fonte 16: Elaboração dos Autores (2025).

Quadro 3 - Código JavaScript - Cadastro

|  |
| --- |
| const MAX\_ESTUDANTES = 10 // LIMITE DE ESTUDANTES CADASTRADOS  const formJS = document.querySelector('#formulario')  const relatorioJs = document.querySelector('#RELATORIO\_HTML')  /\* ========== CADASTRO ===================== \*/  let lista\_estudantes = []  function criaPessoa(nome, ra, idade, sexo, media) {  if (!nome || !ra || !idade || !media) {  RespostaIncorreta()  return  } else {  const resultado = media >= 6.0 ? "Aprovado" : "Reprovado"  return { nome, ra, idade, sexo, media, resultado }  }  }  function RespostaIncorreta() {  return alert('Inserção de Dados Incorreta! Tente novamente')  }  /\*========= CADASTRO ======================== \*/  function buscaEstudanteRepetido(vetor, raNovoEstudante) {  for (let i = 0; i < vetor.length; i++) {  if (vetor[i].ra === raNovoEstudante) return true  }  return false  }  function adic\_estudante\_cadastro(evento) {  evento.preventDefault()  let nome = document.querySelector('#nome\_HTML').value  let ra = Number(document.querySelector('#ra\_HTML').value)  let idade = Number(document.querySelector('#idade\_HTML').value)  let sexo = document.querySelector('#sexo\_HTML').value  let media = Number(document.querySelector('#media\_HTML').value)  const existe\_esse\_estudante = buscaEstudanteRepetido(lista\_estudantes, ra)  if (existe\_esse\_estudante == true) {  alert('Estudante já cadastrado! Insira outro novamente!')  return  }  if (lista\_estudantes.length >= MAX\_ESTUDANTES) {  alert('Limite máximo de estudantes atingido!')  return  }  lista\_estudantes.push(criaPessoa(nome, ra, idade, sexo, media))  formJS.reset()  document.querySelector('#nome\_HTML').focus()  alert('ALUNO CADASTRADO !!')  }  formJS.addEventListener('submit', adic\_estudante\_cadastro) |

Fonte 17: Elaboração dos Autores (2025).

Quadro 4 - Código JavaScript – Geração de Tabela

|  |
| --- |
| /\* ======== FUNÇÃO GENÉRICA DE EXIBIÇÃO======== \*/  function exibirTabelaEstudantes(lista) {  relatorioJs.innerHTML = ''  if (lista.length === 0) {  relatorioJs.innerHTML = '<p>Nenhum estudante ainda cadastrado!</p>'  return  }  const tabela = document.createElement('table')  let titulo = `  <thead>  <tr>  <th> Nomes </th>  <th> RA </th>  <th> Idade </th>  <th> Sexo </th>  <th> Média </th>  <th> Resultado </th>  </tr>  </thead>  `  let corpoTabela = '<tbody>'  for (let estudante of lista) {  corpoTabela += `  <tr>  <td>${estudante.nome}</td>  <td>${estudante.ra}</td>  <td>${estudante.idade}</td>  <td>${estudante.sexo}</td>  <td>${estudante.media.toFixed(2)}</td>  <td>${estudante.resultado}</td>  </tr>  `  }  corpoTabela += '</tbody>'  tabela.innerHTML = titulo + corpoTabela  relatorioJs.appendChild(tabela)  } |

Fonte 18: Elaboração dos Autores (2025).

Quadro 5 - Código JavaScript – Ordenação Selection Sort

|  |
| --- |
| /\* =========== ORDENAÇÃO SELECTION SORT ========== \*/  /\* USAREMOS SELECT SORT \*/  function selectionSort(vetor, funcComp) {  for (let pos\_select = 0; pos\_select < vetor.length - 1; pos\_select++) {  let pos\_menor = pos\_select + 1  for (let i = pos\_menor + 1; i < vetor.length; i++) {  if (funcComp(vetor[pos\_menor], vetor[i])) pos\_menor = i    }  if (funcComp(vetor[pos\_select],vetor[pos\_menor])) {  [vetor[pos\_select], vetor[pos\_menor]] = [vetor[pos\_menor] ,vetor[pos\_select]]    }  }  }  /\* ======== ORDENAÇÃO NOMES CRESCENTE ========= \*/  function acao\_do\_Botao\_NOMES() {  selectionSort(lista\_estudantes, (obj1, obj2) => obj1.nome > obj2.nome)  exibirTabelaEstudantes(lista\_estudantes)  }  const botaoOrdenaNomes = document.querySelector('#nomes\_crescente\_HTML')  botaoOrdenaNomes.addEventListener('click',acao\_do\_Botao\_NOMES)  /\* ======== ORDENAÇÃO RA DECRESCENTE ========== \*/  function Ord\_RA\_Decrescente() {  selectionSort(lista\_estudantes, (obj1, obj2) => obj1.ra < obj2.ra)  exibirTabelaEstudantes(lista\_estudantes)  }  const botaoOrdenaRA = document.querySelector('#ra\_decrescente\_HTML')  botaoOrdenaRA.addEventListener('click',Ord\_RA\_Decrescente)  /\* ======== ORDENAÇÃO APROVADOS POR NOME ========== \*/  function Ord\_Aprovados\_Crescente() {  let aprovados = []   for (let i = 0; i < lista\_estudantes.length; i++) {    if (lista\_estudantes[i].resultado === "Aprovado") {  aprovados.push(lista\_estudantes[i])  }  }  selectionSort(aprovados, (obj1, obj2) => obj1.nome > obj2.nome)  exibirTabelaEstudantes(aprovados)  }  const botaoOrdenaAprovados = document.querySelector('#aprovados\_crescente\_HTML')  botaoOrdenaAprovados.addEventListener('click',Ord\_Aprovados\_Crescente) |

Fonte 19: Elaboração dos Autores (2025).

# 5. CONSIDERAÇÕES FINAIS

Conclui-se que o sistema desenvolvido cumpre a proposta de integrar as funcionalidades de cadastro e ordenação de estudantes de forma simples, eficiente e didática. A escolha das tecnologias web (HTML, CSS e JavaScript) garantiu a acessibilidade e a compatibilidade da solução em diferentes ambientes.

A utilização do algoritmo Selection Sort, embora simples, foi fundamental para consolidar os conceitos básicos de algoritmos e sua aplicação prática. O projeto atingiu seus objetivos, resultando em uma ferramenta funcional que não apenas gerencia registros de estudantes, mas também serve como um objeto de estudo valioso para a compreensão de princípios de programação estruturada e manipulação de dados.

Conclui-se que o sistema desenvolvido cumpre a proposta de integrar as funcionalidades de cadastro e ordenação de estudantes de forma simples, eficiente e didática. A escolha das tecnologias web (HTML, CSS e JavaScript) garantiu a acessibilidade e a compatibilidade da solução em diferentes ambientes.

**5.1. Limitações e Trabalhos Futuros**

Apesar de cumprir seus objetivos educacionais, o sistema apresenta algumas limitações que indicam caminhos para trabalhos futuros:

Persistência de Dados: Os dados estão armazenados apenas na memória volátil (Array JavaScript) e se perdem ao recarregar a página. A persistência em Local Storage ou a integração com um banco de dados (RF020) seria necessária para um sistema em produção.

Escalabilidade do Algoritmo: O Selection Sort é ineficiente para grandes volumes de dados. Experimentos complementares poderiam envolver a implementação e comparação de algoritmos mais eficientes, como o Quick Sort ou Merge Sort, para fins de análise de desempenho (RNF002).

Funcionalidades de CRUD: O sistema atualmente só suporta o "C" (Create) do CRUD. A adição das funcionalidades de Remoção (RF009) e Update (edição) completaria o gerenciamento de registros.

# 6. REFERÊNCIAS

# SILVIO, J. Hacia una educación virtual de calidad, pero con equidad y pertinencia. Revista de Universidad y Sociedad del Conocimiento, RUSC, v.3, n.1, p. 5, 2006.

VÁSQUEZ, M. P. **El rol y el perfil del docente en la educación a distancia.** Universidad Centroccidental Lisandro Alvarado, Venezuela, 2008. [www.icergua.org/latam/pdf/09-segsem/01-04-ph4/doc06.pdf](http://www.icergua.org/latam/pdf/09-segsem/01-04-ph4/doc06.pdf) acessado em 15/04/2021.

1. O código-fonte completo deste projeto está disponível para análise e replicação no repositório GitHub: [↑](#footnote-ref-1)