CURSO TECNOLÓGICO SUPERIOR EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

ESTRUTURA DE DADOS

GIULIA FACIOLI

MARIA EDUARDA CUBERO SANTOS

Programa para Gestão Estudantil

FRANCA/SP

ABRIL/2025

Sumário

Introdução

Este documento descreve o desenvolvimento de um sistema para gerenciar informações de alunos, criado como um trabalho prático para a disciplina de Estrutura de Dados. O sistema permite cadastrar, armazenar e exibir dados como nome, RA, idade, sexo, média e resultado dos alunos. Para organizar esses dados, utilizamos o array, uma estrutura de dados que armazena os alunos em sequência, para persistência dos dados, utilizamos o Local Storage. Além disso, implementamos o algoritmo Selection Sort para ordenar os alunos por nome e RA.

Desenvolvimento

Implementação em Javascript

Este segmento do documento detalha a implementação em JavaScript responsável pela lógica central do painel de gerenciamento de informações estudantis, proposto como parte do trabalho prático da disciplina de Estrutura de Dados. O código desenvolvido visa não apenas atender aos requisitos funcionais de cadastrar, armazenar (Nome, RA, Idade, Sexo, Média, Resultado) e exibir os dados dos alunos, mas também aplicar conceitos fundamentais de manipulação e organização de conjuntos de dados. A seguir, cada componente chave do script será analisado, explicando como as estruturas de dados foram escolhidas e utilizadas para gerenciar a coleção de registros estudantis e como os algoritmos de ordenação foram implementados para permitir a classificação eficiente por Nome (crescente e decrescente) e RA, demonstrando a aplicação prática dos princípios estudados na disciplina.

Backend - Código ‘cadastro.mjs’

Detalhando cada parte do Javascript:

```
import { stringificarObj } from "./auxiliares.mjs";
```

Esta linha importa a função stringificarObj de um módulo externo chamado auxiliares.mjs. Esta função é responsável por converter um objeto JavaScript em uma representação de string (texto) para ser exibida na página.

```
class Aluno {
    constructor(nome, ra, idade, sexo, media, resultado) {
        this.nome = nome;
        this.ra = ra;
        this.idade = idade;
        this.sexo = sexo;
        this.media = media;
        this.resultado = resultado;
    }

    toHTML(){
        return `<p>Nome: ${this.nome}, RA: ${this.ra}, Idade: ${this.idade}, Sexo: ${this.sexo}, Média: ${this.media},
        Resultado: ${this.resultado ? 'Aprovado' : 'Reprovado'}</p>`;
    }
};
```

Explicação:

- **class Aluno { ... }**: Define uma classe chamada Aluno.

- **constructor(nome, ra, idade, sexo, media, resultado) { ... }**: Os parâmetros definidos aqui (nome, ra, idade, sexo, media, resultado) são usados para inicializar as propriedades do objeto Aluno.
- **toHTML(){ ... }**: Este é um método que, quando chamado em um objeto Aluno, ele retorna uma string de HTML formatada que representa as informações do aluno dentro de um parágrafo (<p>). O resultado é exibido como "Aprovado" se a propriedade resultado for true, e "Reprovado" caso contrário (usando o operador ternário ?.

```
19  function validarForm(){
20      const camposObrigatorios = document.querySelectorAll('.form-aluno [required');
21      for(let campo of camposObrigatorios){
22          if(!campo.value){
23              return false
24          }
25      }
26      return true;
27  }
```

- contrário (usando o operador ternário ? :).

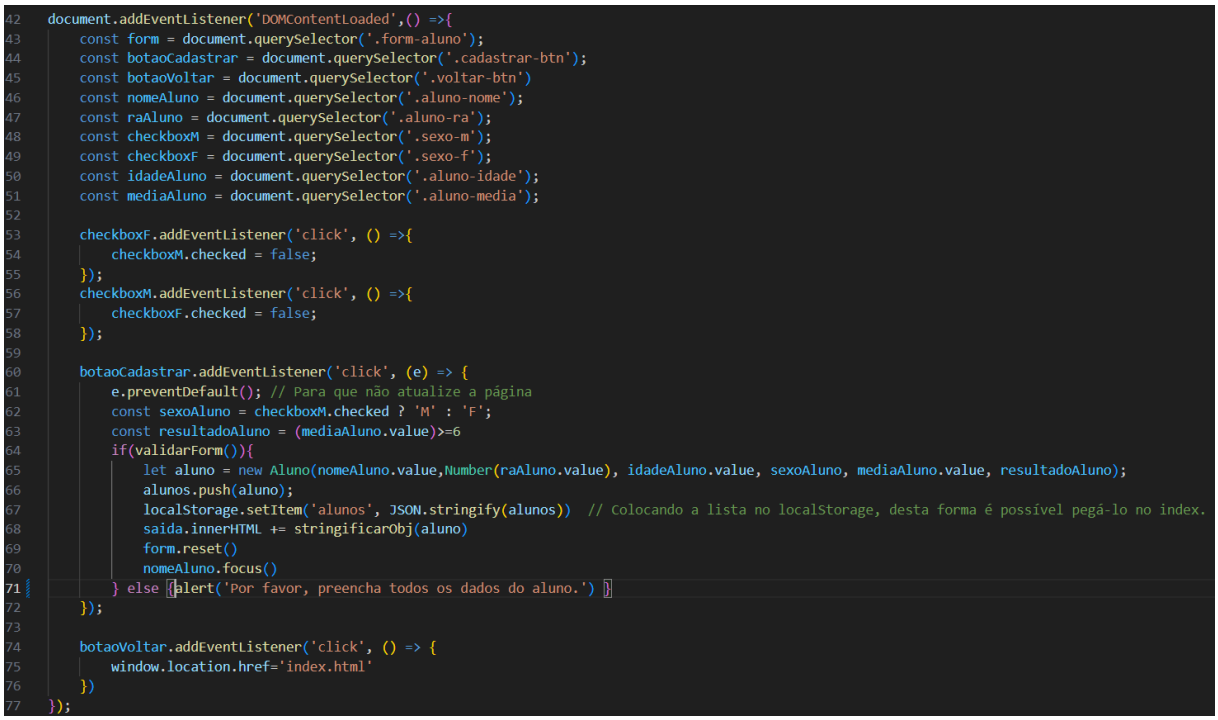
Explicação:

- **function validarForm(){ ... }**: Define uma função chamada validarForm que não recebe nenhum argumento.
- **const camposObrigatorios = document.querySelectorAll('.form-aluno [required');:** Esta linha usa document.querySelectorAll para selecionar todos os elementos dentro de um elemento com a classe form-aluno que possuem o atributo required. O resultado é uma lista de todos os campos obrigatórios dentro do formulário.
- **for(let campo of camposObrigatorios){ ... }:** Este é um loop que itera sobre cada elemento (campo) na lista de camposObrigatorios.
- **if(!campo.value){ return false }:** Dentro do loop, esta condição verifica se o valor (value) do campo atual está vazio (ou seja, !campo.value é true). Se um campo obrigatório estiver vazio, a função imediatamente retorna false, indicando que a validação falhou.
- **return true;:** Se o loop terminar sem encontrar nenhum campo obrigatório vazio, a função retorna true, indicando que o formulário é válido.

```
29  const saida = document.querySelector('.output-alunos');
30  const alunosSalvosJSON = localStorage.getItem('alunos');
31  let alunos = [];
32  if (alunosSalvosJSON){
33      // Se existir uma lista salva como JSON no local storage,
34      // ela será convertida. Do contrário "alunos" continua sendo uma lista vazia.
35      alunos = JSON.parse(alunosSalvosJSON)
36      alunos.forEach(element => {
37          saida.innerHTML += stringificarObj(element)
38      });
39  }
```

Explicação:

- **const saida = document.querySelector('.output-alunos');:** Seleciona o primeiro elemento no documento HTML que possui a classe output-alunos e armazena uma referência a ele na constante saida. Este elemento é onde as informações dos alunos serão exibidas.
- **const alunosSalvosJSON = localStorage.getItem('alunos');:** Tenta recuperar um item do localStorage do navegador com a chave 'alunos'. O localStorage permite que os navegadores armazenem dados em pares de chave/valor localmente no computador do usuário. Se nenhum item com essa chave existir, alunosSalvosJSON será null. O valor armazenado é sempre uma string.
- **let alunos = [];:** Inicializa uma variável chamada alunos como um array vazio. Esta variável será usada para armazenar os objetos Aluno.
- **if (alunosSalvosJSON){ ... }:** Esta condição verifica se algum dado foi recuperado do localStorage (ou seja, se alunosSalvosJSON não é null).
- **alunos = JSON.parse(alunosSalvosJSON):** Se dados foram encontrados, eles são convertidos de uma string JSON para um array de objetos JavaScript usando JSON.parse().
- **alunos.forEach(element => { saida.innerHTML += stringificarObj(element) });:** Este loop itera sobre cada element (que presumivelmente é um objeto representando um aluno) no array alunos recuperado do localStorage. Para cada aluno, a função stringificarObj(element) é chamada para obter sua representação em string, e essa string é adicionada ao conteúdo HTML do elemento saida usando +=. Isso significa que os alunos previamente cadastrados e salvos serão exibidos quando a página carregar.



```
document.addEventListener('DOMContentLoaded',() =>{
    const form = document.querySelector('.form-aluno');
    const botaoCadastrar = document.querySelector('.cadastrar-btn');
    const botaoVoltar = document.querySelector('.voltar-btn')
    const nomeAluno = document.querySelector('.aluno-nome');
    const raAluno = document.querySelector('.aluno-ra');
    const checkboxM = document.querySelector('.sexo-m');
    const checkboxF = document.querySelector('.sexo-f');
    const idadeAluno = document.querySelector('.aluno-idade');
    const mediaAluno = document.querySelector('.aluno-media');

    checkboxF.addEventListener('click', () =>{
        checkboxM.checked = false;
    });
    checkboxM.addEventListener('click', () =>{
        checkboxF.checked = false;
    });

    botaoCadastrar.addEventListener('click', (e) => {
        e.preventDefault(); // Para que não atualize a página
        const sexoAluno = checkboxM.checked ? 'M' : 'F';
        const resultadoAluno = (mediaAluno.value)>=6
        if(validarForm()){
            let aluno = new Aluno(nomeAluno.value,Number(raAluno.value), idadeAluno.value, sexoAluno, mediaAluno.value, resultadoAluno);
            alunos.push(aluno);
            localStorage.setItem('alunos', JSON.stringify(alunos))  // Colocando a lista no localStorage, desta forma é possível pegá-lo no index.
            saida.innerHTML += stringificarObj(aluno)
            form.reset()
            nomeAluno.focus()
        } else {alert('Por favor, preencha todos os dados do aluno.') }
    });

    botaoVoltar.addEventListener('click', () => {
        window.location.href='index.html'
    })
});
```

- **document.addEventListener('DOMContentLoaded',() =>{ ... });:** Este código garante que a função dentro das chaves **{}** será executada somente depois que todo o conteúdo HTML da página tiver sido completamente carregado pelo navegador. Isso é importante para garantir que os elementos do DOM (como formulários e botões) possam ser encontrados usando **document.querySelector.**

Dentro da função **DOMContentLoaded:**

- **Seleção de elementos do DOM:** Várias constantes são declaradas para armazenar referências a elementos HTML específicos do formulário e botões usando document.querySelector com seus respectivos seletores de classe.

Manipulação dos checkboxes de sexo:

- **checkboxF.addEventListener('click', () =>{ checkboxM.checked = false; });:** Adiciona um listener de evento para o clique no checkbox feminino (checkboxF). Quando clicado, ele desmarca o checkbox masculino (checkboxM).
- **checkboxM.addEventListener('click', () =>{ checkboxF.checked = false; });:** Similarmente, adiciona um listener para o clique no checkbox masculino, desmarcando o feminino. Isso garante que apenas um sexo seja selecionado.

Evento de clique no botão "Cadastrar":

- **botaoCadastrar.addEventListener('click', (e) => { ... });:** Adiciona um listener de evento para o clique no botão com a classe cadastrar-btn.
- **e.preventDefault():** Impede o comportamento padrão do botão de submit de um formulário, que geralmente é recarregar a página.
- **const sexoAluno = checkboxM.checked ? 'M' : 'F';:** Determina o sexo do aluno com base no estado do checkbox masculino. Se estiver marcado, o sexo é 'M', caso contrário, é 'F'.
- **const resultadoAluno = (mediaAluno.value)>=6:** Calcula o resultado do aluno. Se o valor da média for maior ou igual a 6, **resultadoAluno** será **true** (aprovado), caso contrário, será **false** (reprovado).
- **if(validarForm()){ ... }:** Chama a função validarForm() para verificar se todos os campos obrigatórios do formulário foram preenchidos.
- **let aluno = new Aluno(...):** Se o formulário for válido, um novo objeto **Aluno** é criado usando os valores dos campos do formulário. Note que **Number(raAluno.value)** converte o valor do RA para um número.
- **alunos.push(aluno);:** O novo objeto Aluno é adicionado ao final do array alunos.
- **localStorage.setItem('alunos', JSON.stringify(alunos)):** O array alunos (agora contendo o novo aluno) é convertido para uma string JSON usando JSON.stringify() e armazenado novamente no localStorage com a chave 'alunos', persistindo os dados entre as sessões do navegador.
- **saida.innerHTML += stringificarObj(aluno):** A representação em string do novo aluno (obtida através de stringificarObj) é adicionada ao conteúdo do elemento saida, exibindo o novo aluno na página.
- **form.reset():** Reseta o formulário, limpando todos os campos.
- **nomeAluno.focus():** Define o foco de volta para o campo de nome do aluno, facilitando a entrada de um novo aluno.
- **else { alert('Por favor, preencha todos os dados do aluno.') }:** Se validarForm() retornar false (algum campo obrigatório está vazio), um alerta é exibido para o usuário.

Evento de clique no botão "Voltar":

- **botaoVoltar.addEventListener('click', () => { window.location.href='index.html' });:** Adiciona um listener de evento para o clique no botão com a classe **voltar-btn**. Quando clicado, ele redireciona a janela do navegador para a página **index.html.**

Backend - Código ‘algoritmos.mjs’

Estrutura da Função:

```
CRUD_Alunos > backend > JS algoritmos.mjs > ...
1   export function selectionSort(array, fnComparacao){
```

- **export:** Indica que esta função pode ser importada e utilizada em outros módulos do seu código JavaScript.
- **Array (Parâmetro):** Este é o array que você deseja ordenar. Ele será modificado diretamente pela função.
- **fnComparacao (Parâmetro)**: Este é um parâmetro que espera uma função de comparação. Essa função será responsável por determinar a ordem entre dois elementos do array. Ela deve receber dois argumentos (os elementos a serem comparados) e retornar true se o primeiro elemento deve vir depois do segundo na ordem desejada, e false caso contrário. Isso torna o selectionSort genérico, permitindo ordenar em ordem crescente, decrescente ou com base em critérios mais complexos.

Primeiro Loop: Posição Selecionada (posSel)

```
2       for(let posSel=0; posSel<array.length - 1;posSel++){
```

- Este é o loop principal que itera sobre o array. A variável posSel representa a posição do elemento que estamos considerando como o início da parte não ordenada do array.
- Ele começa em 0 (o primeiro elemento) e vai até array.length - 2 (o penúltimo elemento). Por que não até o último? Porque quando chegamos ao penúltimo elemento, se todos os elementos anteriores estiverem em suas posições corretas, o último elemento automaticamente estará na posição correta, não havendo necessidade de outra iteração.

- Em cada iteração deste loop, o elemento na posição posSel é considerado temporariamente como o menor (ou maior, dependendo da fnComparacao) da parte não ordenada.

  Inicialização da Posição do Menor (posMenor)

```
3         let posMenor = posSel+1;
```

  - Dentro do primeiro loop, para cada posSel, inicializamos uma variável chamada posMenor. Assumimos inicialmente que o elemento na posição imediatamente seguinte a posSel (posSel + 1) é o menor elemento da parte não ordenada que ainda vamos analisar.

  Segundo Loop: Encontrando o Menor Elemento

```
4         for(let i = posMenor+1;i<array.length; i++){ // Ne
5             if(fnComparacao(array[posMenor], array[i])){
6                 posMenor=i;
7             };
8         };
```

  - Este é um loop aninhado que começa da posição seguinte a posMenor (posMenor + 1) e vai até o final do array (array.length).
  - O objetivo deste loop é percorrer a parte não ordenada do array (começando de posMenor + 1) e encontrar o índice do elemento que é realmente o menor (ou maior, de acordo com a fnComparacao).
  - **if (fnComparacao(array[posMenor], array[i])):**
  - Aqui, a função de comparação fnComparacao é chamada com dois elementos do array: o elemento atualmente considerado o menor (array[posMenor]) e o elemento na posição atual do loop interno (array[i]).
  - Se fnComparacao retornar true, isso significa que, de acordo com o critério de comparação, array[posMenor] deve vir depois de array[i]. Em outras palavras, array[i] é menor (ou deveria vir antes) que o atual array[posMenor].
  - Nesse caso, atualizamos posMenor para i, pois encontramos um elemento menor (ou que deve vir antes).

```
            };
            if(fnComparacao(array[posSel], array[posMenor]) ){ // Comparando o valor isol
                [ array[posSel], array[posMenor] ] = [ array[posMenor], array[posSel] ];
            };
        };
    };
```

- Após o segundo loop terminar, posMenor conterá o índice do menor elemento encontrado na parte não ordenada do array (começando de posSel + 1).
- Agora, comparamos o elemento na posição selecionada (array[posSel]) com o menor elemento encontrado (array[posMenor]) usando a fnComparacao.
- **if (fnComparacao(array[posSel], array[posMenor])):**
- Se fnComparacao retornar true, significa que o elemento na posição selecionada (array[posSel]) deve vir depois do menor elemento encontrado (array[posMenor]). Portanto, precisamos trocar esses dois elementos.
- **[array[posSel], array[posMenor]] = [array[posMenor], array[posSel]];:**
- Esta é uma forma concisa em JavaScript (usando a sintaxe de destructuring assignment) de trocar os valores de dois elementos em um array sem a necessidade de uma variável temporária. O valor de array[posMenor] é **atribuído a array[posSel], e o valor de array[posSel] é atribuído a** array[posMenor].

Backend - Código ‘auxiliares.mjs’

```
export function stringificarObj(obj) {
    return `<br>Nome: ${obj.nome}, \
    RA: ${obj.ra}, \
    Idade:${obj.idade}, \
    Sexo: ${obj.sexo}, \
    Média: ${obj.media}, \
    Resultado: ${obj.resultado ? '<span style="color:green">Aprovado</span>' : '<span style="color:red">Reprovado</span>'}\<br>`
}

export function tabelarObj(obj){
    return `
    <tr>
    <td>${obj.nome}</td>
    <td>${obj.ra}</td>
    <td>${obj.idade}</td>
    <td>${obj.sexo}</td>
    <td>${obj.media}</td>
    <td>${obj.resultado?'Aprovado':'Reprovado'}</td>
    </tr>
    `
}
```

O arquivo auxiliares.mjs abriga duas funções de conversão para html, são elas:

stringificarObjeto:

- Pega um objeto de aluno e retorna uma string HTML formatada para exibição em linha.
- Inclui quebras de linha (
) e formata o resultado como "Aprovado" em verde ou "Reprovado" em vermelho usando a tag com estilos CSS.

tabelarObj:

- Pega um objeto de aluno e retorna uma string HTML que representa uma linha (<tr>) de uma tabela HTML.

O resultado é exibido como "Aprovado" ou "Reprovado" (sem formatação de cor).

Frontend - Código ‘index.mjs’

Este código é a parte do sistema que o usuário vê e interage na página inicial (index.html). Ele faz o seguinte:

- Pega os dados dos alunos salvos no navegador (se houver).
- Espera a página carregar completamente para garantir que todos os elementos estejam disponíveis.
- Pega os botões e o menu de seleção da página.
- Quando o usuário clica no botão "Gerar relatório", o código pega a opção escolhida (ordenar por nome, RA ou listar aprovados) e mostra o relatório na tela.
- Se o usuário clicar em "Cadastrar aluno", o código abre a página de cadastro (cadastro.html).
- Se clicar em "Limpar lista", o código apaga todos os dados salvos e atualiza a página.

```
import { htmlRelatorio, tipoRelatorio } from './relatorio.mjs';

const listaAlunos = JSON.parse(localStorage.getItem('alunos'));
const res = document.querySelector('.res');

if(listaAlunos){
    console.log(listaAlunos);
} else {
    console.log('Não foi encontrado no Local storage.')
}
```

```
document.addEventListener('DOMContentLoaded', () => { // Para evitar problemas no query
    const cadastrarBotao = document.querySelector('.cadastrar-btn')
    const relatorioBotao = document.querySelector('.gerar-rel-btn')
    const opcaoRelatorio = document.querySelector('.tipo-rel')
    const apagarLista = document.querySelector('.limpar-btn')

    // Clique no botão de gerar relatório
    relatorioBotao.addEventListener('click',() => {
        switch(opcaoRelatorio.value){
            case 'rel-nome':
                res.innerHTML = htmlRelatorio(2)
                tipoRelatorio(2,listaAlunos);
                break
            case 'rel-ra':
                res.innerHTML = htmlRelatorio(3)
                tipoRelatorio(3,listaAlunos);
                break;
            case 'rel-aprovados':
                res.innerHTML = htmlRelatorio(4)
                tipoRelatorio(4, listaAlunos);
                break;
            default:
                alert('Nenhuma opção de relatório foi selecionada!');
        }
    });
    // Clique no botão de cadastrar aluno
    cadastrarBotao.addEventListener('click', () => {
        window.location.href='cadastro.html';
    });
    // Clique no botão de limpar lista
    apagarLista.addEventListener('click', () => {
        localStorage.clear();
        alert('A lista foi limpa com sucesso!')
        window.location.reload()
    })
});
```

Frontend - Código ‘relatorio.mjs’

Este código é responsável por gerar os relatórios de alunos na página inicial.

- Importa a função selectionSort para ordenar os alunos e a função tabelarObj para mostrar os dados em tabela.
- A função htmlRelatorio cria a estrutura HTML da tabela do relatório, mudando o título de acordo com a opção escolhida (ordem crescente por nome, decrescente por RA, ou lista de aprovados).

- A função tipoRelatorio decide qual tipo de ordenação será aplicada (por nome ou RA) e, se for o caso, filtra a lista para mostrar apenas os alunos aprovados.
- A função checkarAprovados verifica se o aluno foi aprovado.
- A função passarListaParaOHtml pega a lista de alunos (já ordenada e filtrada, se necessário) e coloca os dados de cada aluno em uma linha da tabela na página.

Fontes e conteúdo relacionado

Estrutura de Dados Utilizada: Array (Vetor)

O algoritmo Selection Sort opera diretamente sobre uma estrutura de dados fundamental: o array (também conhecido como vetor).

Definição: Um array é uma coleção linear de elementos, onde cada elemento é armazenado em posições contíguas de memória. Cada posição possui um índice numérico que permite o acesso direto a qualquer elemento da coleção.

Algoritmo Selection Sort

Características Importantes para o Selection Sort

Acesso Indexado: O Selection Sort se beneficia da capacidade de acessar elementos do array diretamente através de seus índices (por exemplo, array[i]). Isso é crucial para percorrer o array, comparar elementos e realizar as trocas.

Mutabilidade: O Selection Sort é um algoritmo de ordenação "in-place", o que significa que ele ordena o array diretamente, modificando a ordem dos elementos dentro do próprio array original. A natureza mutável do array é essencial para realizar as trocas de elementos necessárias para a ordenação.

Algoritmo Selection Sort: Vantagens e Desvantagens

O Selection Sort é um algoritmo de ordenação simples e intuitivo, mas possui características específicas em termos de desempenho.

Vantagens Selection Sort:

Simplicidade e Fácil Implementação: O Selection Sort é um dos algoritmos de ordenação mais fáceis de entender e implementar. Sua lógica é direta: encontrar o menor (ou maior) elemento e colocá-lo na sua posição correta.

Bom para Pequenos Conjuntos de Dados: Para arrays pequenos, a sobrecarga de algoritmos mais complexos pode não compensar, e o Selection Sort pode ter um desempenho razoável.

Número Mínimo de Trocas (Swaps): Uma característica notável do Selection Sort é que ele realiza o número mínimo possível de trocas para ordenar um array. Em cada iteração do loop externo, ocorre no máximo uma troca. Isso pode ser vantajoso em situações onde a operação de troca é computacionalmente cara.

Desempenho Previsível: O tempo de execução do Selection Sort é relativamente consistente, não dependendo significativamente da ordenação inicial dos dados. Ele sempre realizará aproximadamente o mesmo número de comparações.

Desvantagens Selection Sort:

Ineficiente para Grandes Conjuntos de Dados: A principal desvantagem do Selection Sort é sua ineficiência para arrays grandes. Sua complexidade de tempo é de O (n elevado a 2) no pior caso, caso médio e melhor caso, onde n é o número de elementos no array. Isso significa que o tempo de execução cresce quadraticamente com o tamanho da entrada, tornando-o impraticável para grandes volumes de dados.

Muitas Comparações: Embora minimize o número de trocas, o Selection Sort realiza um número significativo de comparações. No pior caso, ele realiza aproximadamente n (n-1)/2 comparações.

Não é Estável: Um algoritmo de ordenação é considerado estável se elementos com o mesmo valor mantêm sua ordem relativa original após a ordenação. O Selection Sort não é inerentemente estável. Em algumas implementações, a ordem relativa de elementos iguais pode ser alterada durante as trocas.

Conclusão

Este segmento detalha a implementação em JavaScript responsável pela lógica central do painel de gerenciamento de informações estudantis, proposto como parte do trabalho prático da disciplina de Estrutura de Dados. O código desenvolvido visa atender aos requisitos funcionais de cadastrar, armazenar (Nome, RA, Idade, Sexo, Média, Resultado) e exibir os dados dos alunos, aplicando conceitos fundamentais de manipulação e organização de conjuntos de dados. A estrutura de dados primária utilizada para o armazenamento da coleção de registros estudantis é o array. Esta escolha permite o armazenamento sequencial dos objetos Aluno, facilitando a iteração e o acesso aos dados. Cada aluno é representado por um objeto da classe Aluno, encapsulando suas informações relevantes. A persistência dos dados entre sessões do navegador é garantida através do localStorage, que armazena a coleção de alunos em formato JSON.

Para a funcionalidade de ordenação dos registros estudantis, foi implementado o algoritmo Selection Sort. Este algoritmo opera diretamente sobre o array de alunos, percorrendo-o repetidamente para encontrar o menor (ou maior, dependendo do critério de ordenação) elemento não ordenado e movê-lo para sua posição correta. A implementação permite a classificação eficiente dos alunos por Nome (em ordem crescente e decrescente) e por Registro Acadêmico (RA), demonstrando a aplicação prática dos princípios de algoritmos de ordenação estudados na disciplina.

Em suma, este trabalho prático demonstra a aplicação dos conhecimentos adquiridos na disciplina, fornecendo uma base funcional para o gerenciamento de informações estudantis.