UNI-FACEF

Curso: Engenharia de Software

Disciplina: Estrutura de Dados II

Relatório Técnico – Versão WEB

Alunos: Pablo Vinicius Soares Vitor 25235; Patrick Soares Vitor 25237

Professor: Alexandre Gomes

Sumário

1. Introdução

Este trabalho tem como objetivo aplicar os conhecimentos adquiridos na disciplina de Estrutura de Dados II através do desenvolvimento de uma aplicação em JavaScript. A proposta envolve a manipulação de dados de alunos, utilizando um array de objetos, com operações de cadastro e relatórios baseados em algoritmos de ordenação clássicos (Bubble Sort e Selection Sort).

2. Fundamentação Teórica

2.1 Estruturas de Dados Utilizadas

A estrutura escolhida foi o Array de Objetos, que permite armazenar de forma organizada dados heterogêneos (nome, RA, idade, sexo, média e resultado).

- Vantagens: simplicidade de implementação, flexibilidade na adição e remoção de registros.

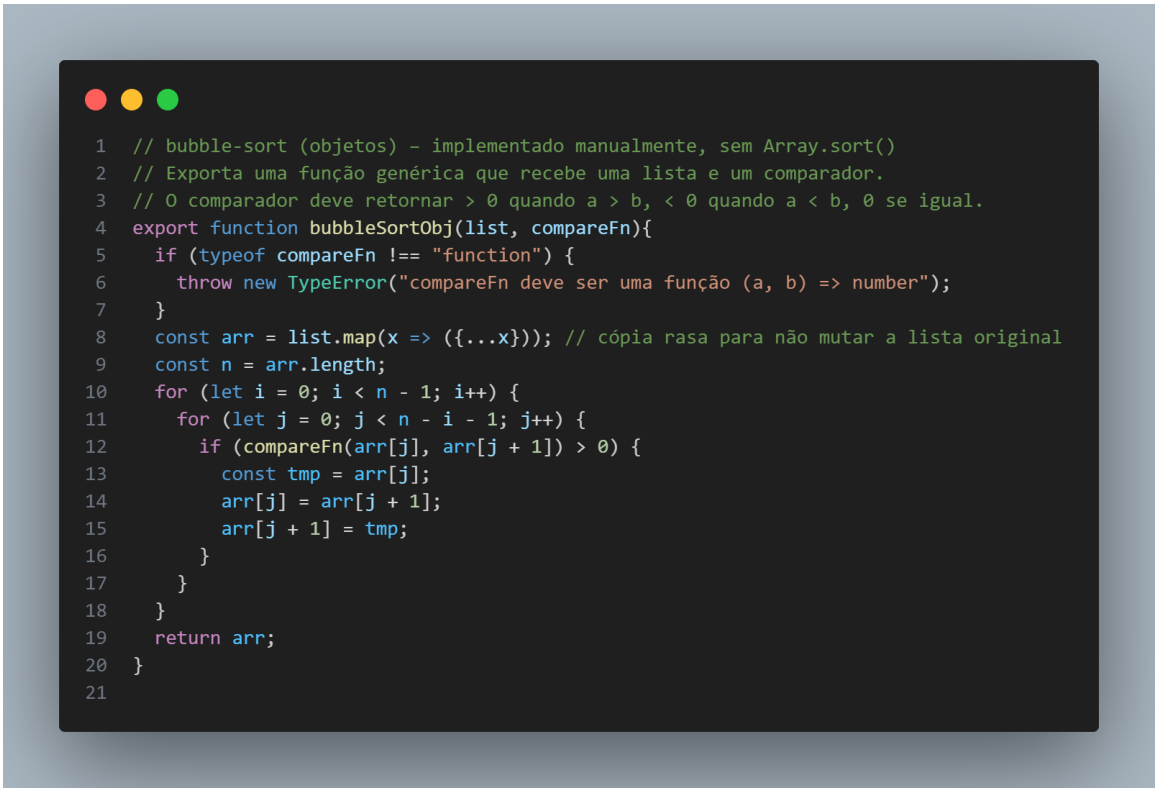
- Desvantagens: desempenho limitado em buscas e ordenações quando comparado a estruturas mais complexas (árvores ou listas encadeadas).

```
function cadastrarAluno({nome, ra, idade, sexo, media}){
  const aluno = {
    nome: String(nome||'').trim(),
    ra: Number(ra),
    idade: Number(idade),
    sexo: String(sexo||'').trim(),
    media: Number(media),
    resultado: calcularResultado(media)
  };
  alunos.push(aluno);
  renderTabela(alunos);
}

// ===== Ordenações específicas exigidas =====
// 1) Crescente por Nome (Bubble)
function ordenarPorNomeCrescente(list){
  return bubbleSortObj(list, (a, b) => {
    return a.nome.localeCompare(b.nome, 'pt-BR', {sensitivity: 'base'});
  });
}

// 2) Decrescente por RA (Selection)
function ordenarPorRADecrescente(list){
  return selectionSortObj(list, (a, b) => a.ra > b.ra);
}

// 3) Aprovados por Nome (crescente)
function relatorioAprovadosPorNome(list){
  const aprovados = [];
  for(const a of list){ if(a.resultado === 'Aprovado') aprovados.push(a); }
  return ordenarPorNomeCrescente(aprovados);
}
```

2.2 Algoritmos de Ordenação

2.2.1 Bubble Sort (Nome crescente)

O Bubble Sort compara elementos adjacentes e troca quando fora de ordem, repetindo o processo até não haver mais trocas. Complexidade: $O(n^2)$. Foi aplicado para ordenar a lista por Nome em ordem crescente.

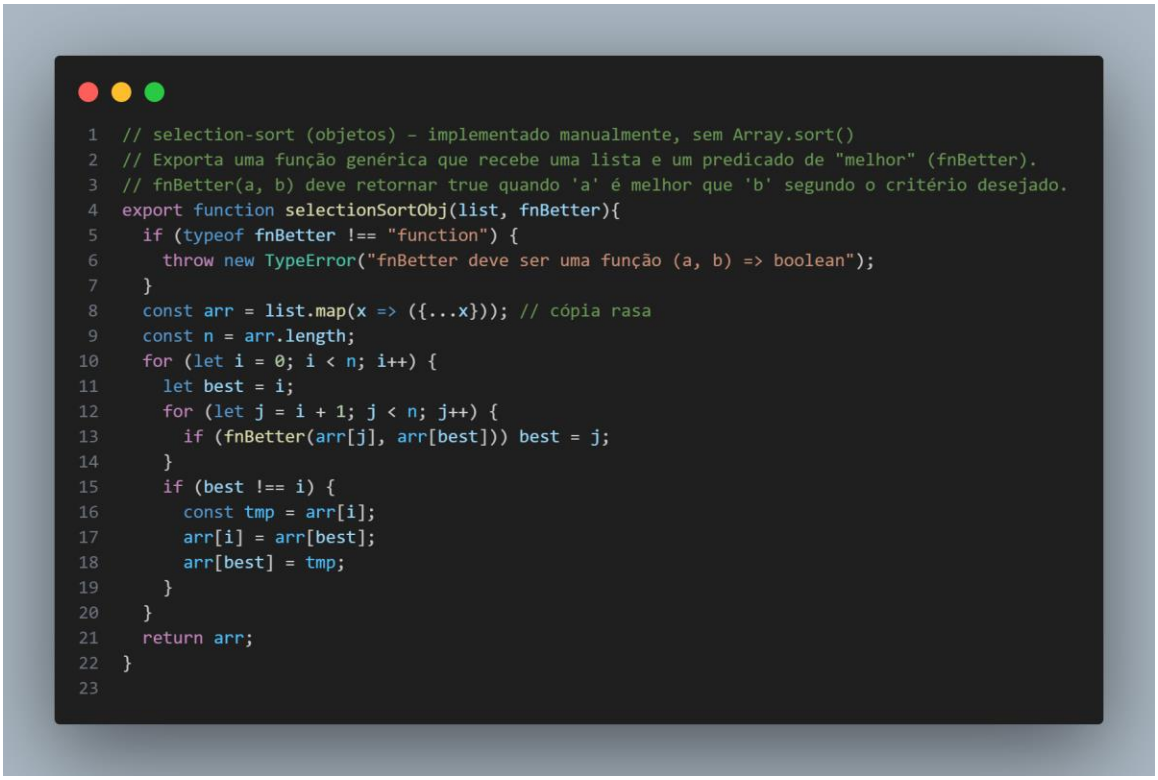


```
// bubble-sort (objetos) – implementado manualmente, sem Array.sort()
// Exporta uma função genérica que recebe uma lista e um comparador.
// O comparador deve retornar > 0 quando a > b, < 0 quando a < b, 0 se igual.
export function bubbleSortObj(list, compareFn){
  if (typeof compareFn !== "function") {
    throw new TypeError("compareFn deve ser uma função (a, b) => number");
  }
  const arr = list.map(x => ({...x})); // cópia rasa para não mutar a lista original
  const n = arr.length;
  for (let i = 0; i < n - 1; i++) {
    for (let j = 0; j < n - i - 1; j++) {
      if (compareFn(arr[j], arr[j + 1]) > 0) {
        const tmp = arr[j];
        arr[j] = arr[j + 1];
        arr[j + 1] = tmp;
      }
    }
  }
  return arr;
}
```

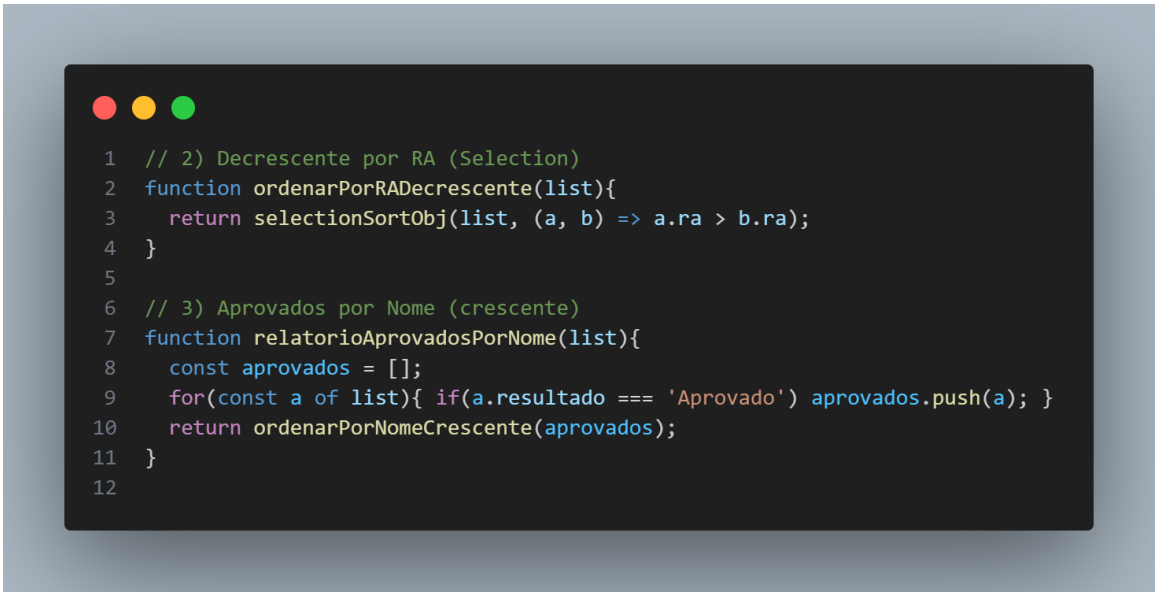
```
// 1) Crescente por Nome (Bubble)
function ordenarPorNomeCrescente(list){
  return bubbleSortObj(list, (a, b) => {
    return a.nome.localeCompare(b.nome, 'pt-BR', {sensitivity: 'base'});
  });
}
```

2.2.2 Selection Sort (RA decrescente)

O Selection Sort seleciona repetidamente o maior elemento e o posiciona na ordem correta. Complexidade: $O(n^2)$. Foi aplicado para ordenar a lista por RA em ordem decrescente.



```
// selection-sort (objetos) – implementado manualmente, sem Array.sort()
// Exporta uma função genérica que recebe uma lista e um predicado de "melhor" (fnBetter).
// fnBetter(a, b) deve retornar true quando 'a' é melhor que 'b' segundo o critério desejado.
export function selectionSortObj(list, fnBetter){
  if (typeof fnBetter !== "function") {
    throw new TypeError("fnBetter deve ser uma função (a, b) => boolean");
  }
  const arr = list.map(x => ({...x})); // cópia rasa
  const n = arr.length;
  for (let i = 0; i < n; i++) {
    let best = i;
    for (let j = i + 1; j < n; j++) {
      if (fnBetter(arr[j], arr[best])) best = j;
    }
    if (best !== i) {
      const tmp = arr[i];
      arr[i] = arr[best];
      arr[best] = tmp;
    }
  }
  return arr;
}

```



```
// 2) Decrescente por RA (Selection)
function ordenarPorRADecrescente(list){
  return selectionSortObj(list, (a, b) => a.ra > b.ra);
}

// 3) Aprovados por Nome (crescente)
function relatorioAprovadosPorNome(list){
  const aprovados = [];
  for(const a of list){ if(a.resultado === 'Aprovado') aprovados.push(a); }
  return ordenarPorNomeCrescente(aprovados);
}

```

3. Metodologia

Ambiente WEB com HTML + CSS + JavaScript (ES Modules). Execução via VS Code + extensão Live Server.

• Entradas: formulário no HTML (Nome, RA, Idade, Sexo, Média).

• Processamento: array de objetos; ordenações manuais (Bubble e Selection).

• Saída: tabela dinâmica atualizada instantaneamente; exportação CSV; impressão otimizada.

4. Implementação (HTML, CSS, JS)

A seguir, destacam-se os principais trechos de código com orientação de captura:

4.1 `index.html` – estrutura (Arquivo: AtividadePraticaED/frontend/src/index.html)

• Header e descrição – linhas 03 a 22.

• Formulário "Cadastrar Aluno" – linhas 26 a 61.

• Seção "Relatórios" – linhas 62 a 73.

• Tabela "Alunos" – linhas 74 a 92.

• Bloco "Sobre o Projeto" – linhas 93 a 109.

• Barra de Suporte (Exportar, Imprimir, Topo) – linhas 111 a 121.

4.2 `script.mjs` – lógica da aplicação (Arquivo: AtividadePraticaED/frontend/src/script.mjs)

• Modelo e cadastro – linhas 9 a 20.

• Ordenação por Nome (Bubble) – linha 24 (capturar função e chamada; 96 linhas).

• Ordenação por RA (Selection) – linha 31 (capturar função e chamada; 100 linhas).

• Aprovados por Nome – linha 36 (capturar função e chamada; 104 linhas).

• Listeners dos botões – a partir da linha 78 até o final.

4.3 `stylo.css` – estilos (Arquivo: AtividadePraticaED/frontend/src/stylo.css)

• Texto de descrição

• Seção Sobre

• Barra de suporte

• Estilos de impressão

5. Relatórios e Evidências (PRINTS)

Gerar os seguintes prints na interface WEB:

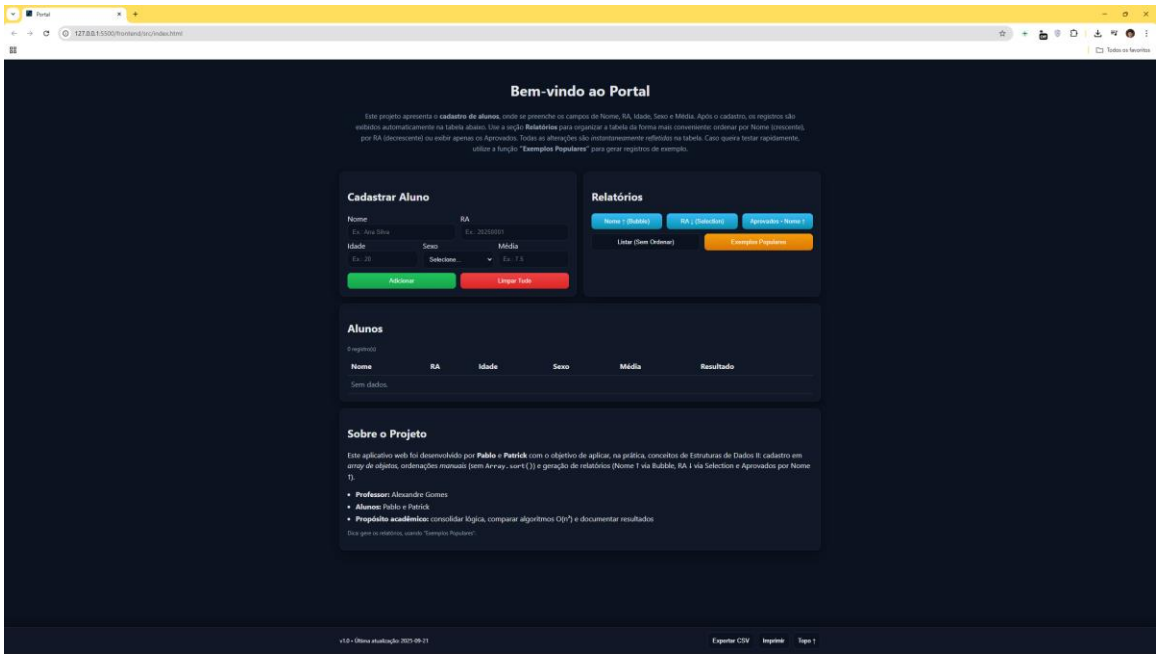
5.1 Tela inicial com descrição e cards

5.2 Cadastro realizado

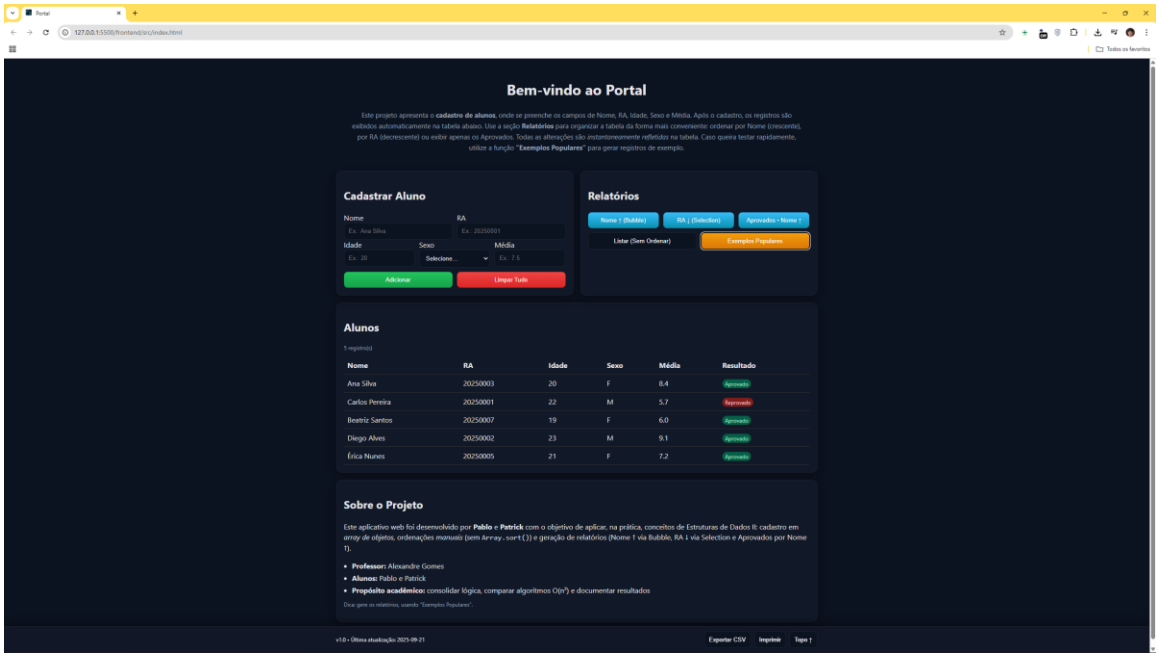
5.3 Relatório por Nome (crescente)

5.4 Relatório por RA (decrescente)

5.5 Relatório de Aprovados por Nome (crescente)

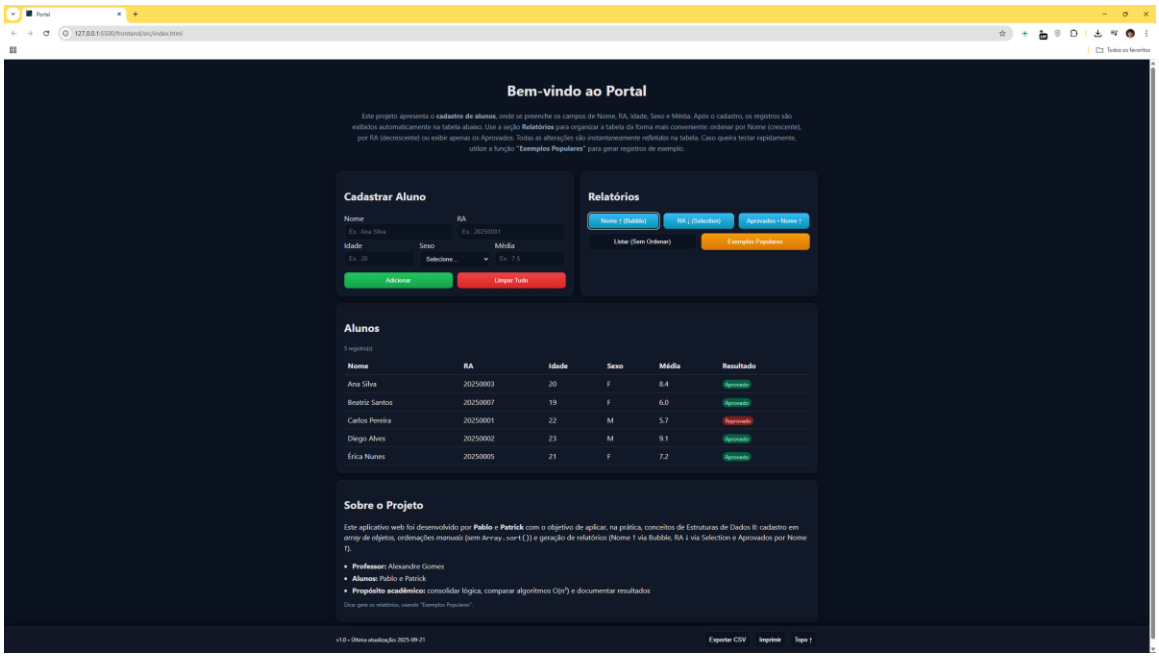


[1]IMG – 5.1

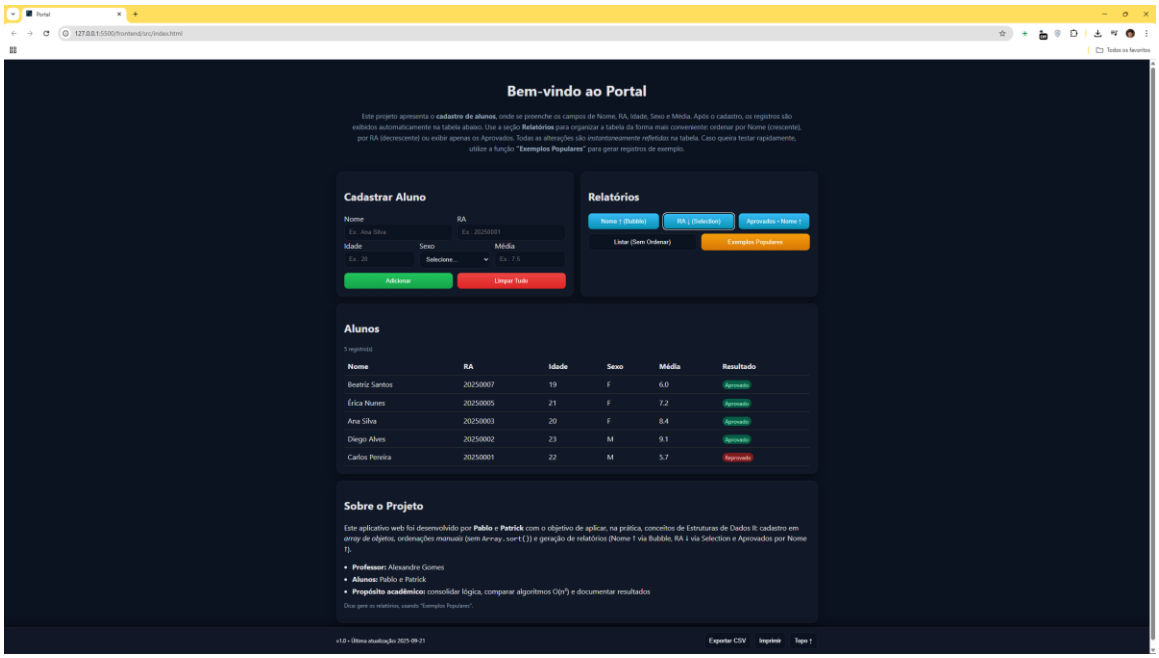


[2]IMG – 5.2

[1] IMG-5.1



[3]IMG – 5.3

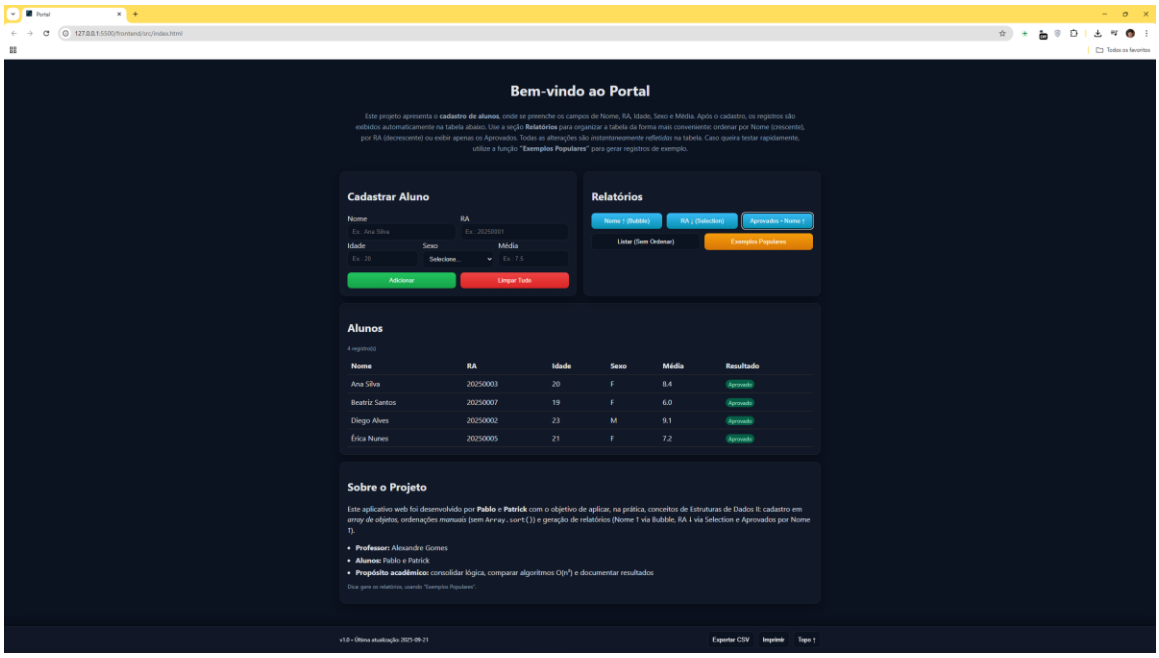


[4]IMG – 5.4

2

3

4



[5]IMG – 5.5

6. Discussão e Resultados

Os algoritmos Bubble e Selection apresentam complexidade $O(n^2)$, adequados para fins didáticos e conjuntos pequenos. O Bubble mostrou-se estável para a ordenação por Nome; o Selection, não estável por definição, foi suficiente para RA decrescente. A modularização (módulos de ordenação + script de interface) facilitou a clareza e a manutenção.

7. Conclusão

O objetivo foi alcançado: o sistema WEB permite cadastrar alunos, visualizar resultados em tempo real e gerar relatórios com ordenações manuais, sem uso de Array.sort(). Como evolução, recomenda-se a adoção de algoritmos O(n log n) (Merge/Quick) e paginação da tabela para grandes volumes.

Referências

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. Algoritmos: Teoria e Prática. 3. ed. Elsevier, 2012.

SEDGEWICK, R.; WAYNE, K. Algorithms. 4. ed. Addison-Wesley, 2011.

MDN Web Docs – JavaScript. Disponível em: https://developer.mozilla.org/

5