
Documento Técnico Didático

Projeto: Registro e Monitoramento de Temperatura via Navegador

Tecnologias utilizadas: HTML5 + JavaScript

Objetivo: Demonstrar a interação entre entrada, processamento, armazenamento e saída em um sistema simples executado no navegador.

◆ 1. Estrutura HTML (Interface do Usuário)

<!DOCTYPE html>

Declara que o documento segue o padrão HTML5.

<html lang="pt-BR">

Define que a linguagem da página é o **português brasileiro**.

<head>

<meta charset="UTF-8" />

<title>Exemplo Simples: Entrada, Processamento, Armazenamento e Saída</title>

</head>

- <meta charset="UTF-8" />: Garante suporte a acentos e caracteres especiais.
- <title>: Título que aparecerá na aba do navegador.

<body>

<h3>Temperatura (°C)</h3>

Cabeçalho da seção onde o usuário irá inserir dados (componente **de entrada**).

<input id="inputTemp" type="number" placeholder="Digite a temperatura" />

- Cria um campo de entrada de **número decimal**.
- id="inputTemp": Serve para referenciar o elemento no JavaScript.
- Relacionado ao **componente de entrada** (teclado).

<button onclick="salvar()">Salvar</button>

- Botão que dispara a função salvar() quando clicado.
- O clique é uma **entrada por mouse**.

<h4>Histórico de Temperaturas</h4>

<ul id="listaHistorico">

- Título da seção de **saída de dados**.
 - ul (lista não ordenada) será preenchida dinamicamente com o histórico de temperaturas.
-

◆ 2. Bloco JavaScript (Lógica da Aplicação)

<script>

Início do bloco de **script JavaScript**. Toda a lógica está dentro dele.

▶ Função salvar(): Captura, processa e armazena a entrada

```
function salvar() {
```

Cria a função principal do sistema. Executada ao clicar no botão "Salvar".

◆ [Entrada] Captura o valor digitado

```
const input = document.getElementById('inputTemp');  
const temp = parseFloat(input.value);
```

- input: constante que armazena o elemento HTML <input>.
 - temp: converte o valor digitado para **número decimal** (parseFloat).
 - Componentes de **entrada**: teclado + elemento input.
-

◆ [Validação] Verifica se o valor é válido

```
if (isNaN(temp)) {  
  alert('Digite um número válido');  
  return;  
}
```

- isNaN(temp): verifica se o valor **não é um número**.
 - Se for inválido, mostra uma **mensagem de alerta** e encerra a função.
-

◆ [Processamento] Define status com base na temperatura

```
const status = temp > 25 ? 'Alerta' : 'OK';
```

- **Operador ternário**: se temp > 25, define "Alerta", senão "OK".
 - Constante status representa o **resultado do processamento** (interpretação do dado).
-

◆ [Armazenamento] Recupera e atualiza histórico local

```
const historico = JSON.parse(localStorage.getItem('historicoTemp') || '[]');
```

- localStorage.getItem(...): tenta buscar dados salvos.
- Se não houver nada salvo, usa [] (lista vazia).

- `JSON.parse(...)`: transforma o texto salvo em **lista de objetos**.
-

◆ [Processamento + Armazenamento] Adiciona nova entrada

```
historico.push({ temp, status, data: new Date().toLocaleString() });
```

- `historico` é uma lista de objetos.
- Cada objeto possui:
 - `temp`: temperatura digitada
 - `status`: alerta ou OK
 - `data`: data/hora atual, como string

```
localStorage.setItem('historicoTemp', JSON.stringify(historico));
```

- Converte a lista de volta para texto JSON.
 - Salva no **armazenamento local do navegador** (`localStorage`).
 - Componente de **armazenamento**: equivalente a HD/SSD, mas no navegador.
-

◆ [Saída] Atualiza visualmente a lista de histórico

```
mostrarHistorico();
```

- Chama a função que **exibe os dados na tela** (saída).

```
input.value = "";
```

- Limpa o campo de entrada para nova digitação.
-

▶ Função `mostrarHistorico()`: exibe lista de temperaturas salvas

```
function mostrarHistorico() {
```

Essa função monta visualmente o histórico na tela, percorrendo os dados salvos.

```
const lista = document.getElementById('listaHistorico');  
lista.innerHTML = "";
```

- Referencia o elemento ``, e limpa seu conteúdo atual.

```
const historico = JSON.parse(localStorage.getItem('historicoTemp') || '[]');
```

- Recupera os dados salvos no `localStorage`.

```
historico.forEach(item => {  
  const li = document.createElement('li');  
  li.textContent = `${item.data}: ${item.temp}°C → ${item.status}`;  
  lista.appendChild(li);  
});
```

- Para cada item no histórico:
 - Cria um elemento
 - Define o texto com data, temperatura e status
 - Adiciona o item à lista na tela

Essa é a **saída de dados** apresentada ao usuário via monitor.

Exibir histórico automaticamente ao carregar a página

`mostrarHistorico();`

Ao final do script, a função `mostrarHistorico()` é chamada para mostrar qualquer dado salvo anteriormente (mesmo após fechar o navegador).

Resumo Visual das Responsabilidades

| Elemento | Papel no Ciclo | Código Envolvido |
|---------------|----------------------------|--|
| Entrada | Coletar dado do usuário | <code><input></code> , <code>document.getElementById()</code> , <code>input.value</code> |
| Processamento | Decisão lógica e validação | <code>if, status = temp > 25 ? ...</code> |
| Armazenamento | Guardar dados localmente | <code>localStorage.setItem()</code> , <code>JSON.stringify()</code> |
| Saída | Mostrar dados ao usuário | <code>document.createElement()</code> , <code>appendChild()</code> |

Conclusão







Este exemplo simples implementa **um ciclo completo de informação**, com os mesmos conceitos que sistemas maiores utilizam:

- Entrada (interface + periféricos)
 - Processamento (lógica de negócios)
 - Armazenamento (banco de dados ou `localStorage`)
 - Saída (interface de apresentação)
-

Relação entre o Código e os Componentes de Hardware

| Componente | O que faz no código | Explicação didática | Exemplos reais no computador |
|---------------|---|--|--|
| Entrada | <code><input></code> para digitar a temperatura | O usuário digita com o teclado a temperatura (dado de entrada) | Teclado, mouse (para clicar em "Salvar") |
| Processamento | <code>const status = temp > 25 ? 'Alerta' : 'OK';</code> | O navegador (CPU) interpreta os dados e toma decisões, como verificar se a temperatura é alta | CPU executa esse cálculo |
| Armazenamento | <code>localStorage.setItem(...)</code> e <code>localStorage.getItem(...)</code> | O dado digitado é salvo no navegador , em um espaço chamado localStorage | HD, SSD ou memória do navegador |
| Saída | <code>lista.appendChild(...)</code> mostra dados na tela | O navegador exibe na tela o histórico de temperaturas com base nos dados armazenados | Monitor mostra a lista |

Ciclo Explicado com Etapas Visuais

- Entrada** 
O usuário digita uma temperatura no campo e clica em **Salvar**.
 Isso usa **entrada de dados** via teclado e mouse.
- Processamento** 
O navegador (JavaScript) verifica:
 - A entrada é válida (é um número)?
 - A temperatura é maior que 25°C?
 Se sim, o **status é "Alerta"**, senão "OK".
- Armazenamento** 
O código pega a temperatura, a data e o status e **grava no armazenamento local** (como se fosse um mini-HD do navegador, chamado `localStorage`).
- Saída** 
Depois de salvar, o código **atualiza a tela**, exibindo o histórico com data, temperatura e status — **usando o monitor** como saída.

Exemplo Prático


Se o usuário digita 30 e clica em **Salvar**:

- **Entrada:** 30 digitado com teclado
- **Processamento:** $30 > 25 \rightarrow \text{status} = \text{"Alerta"}$
- **Armazenamento:** salva `{ temp: 30, status: "Alerta", data: "..."}`
- **Saída:** aparece na tela algo como:

16/06/2025, 16:10: 30°C → Alerta

Curiosidades Didáticas

- O localStorage **não é volátil**, ou seja, os dados **continuam lá mesmo depois de fechar o navegador**. (diferente da RAM!)
 - Essa estrutura imita um **sistema completo de informação**: entrada → processamento → armazenamento → saída.
-

 **Desafio: Reutilize o código de medição de temperatura para criar um sistema de monitoramento de batimentos cardíacos!**

Instruções:

1. **Altere o título e o placeholder do campo de entrada** para indicar que agora o dado é BPM (batimentos por minuto).
2. **Altere o nome da variável temp para bpm** e adapte todas as ocorrências.
3. **Modifique a lógica de avaliação do status**, de forma que:
 - `bpm < 60` → 'Baixa Frequência'
 - `bpm >= 60 && bpm <= 100` → 'Normal'
 - `bpm > 100` → 'Alta Frequência'
4. **Atualize os rótulos do HTML** para refletir o novo propósito.
5. **Salve o histórico com os novos dados** e exiba-os como:
"16/06/2025, 16:30: 72 BPM → Normal"
- 6.

Dica:

Você só precisa **modificar textos, nomes de variáveis e a lógica condicional**. O restante do código pode ser mantido!

```
// Avaliação dos batimentos cardíacos
let status = "";
if (bpm < 60) {
  status = 'Baixa Frequência';
} else if (bpm <= 100) {
  status = 'Normal';
} else {
  status = 'Alta Frequência';
}
```