**JAVASCRIPT => NIVEL 1**

**✅ ETAPA 1 – Fundamentos do JavaScript**

**1. Introdução**

* O JavaScript é usado para adicionar **interatividade** às páginas web.
* Ele roda no **navegador** e, hoje em dia, também no servidor (com Node.js).

**Primeiro código JavaScript (no navegador):**

*html*

---

<!DOCTYPE html>

<html>

<head><title>Primeiro JS</title></head>

<body>

<h1>Olá, Mundo!</h1>

<script>

alert ("Olá, Mundo!");

</script>

</body>

</html>

➡ **Explicação:**  
<script> é onde colocamos o código JavaScript.  
alert("Olá, Mundo!"); exibe uma caixa de alerta.

**2. Variáveis e Tipos de Dados**

* var, let, const (mais usado: let e const)
* Tipos: string, number, boolean, null, undefined

**Exemplo:**

*javascript*

---

let nome = "Ana"; // string

let idade = 25; // number

let ativo = true; // boolean

➡ **Dica:** Use let para variáveis que mudam e const para valores fixos.

**Exercício:**  
Crie variáveis para **nome**, **idade** e **cidade**, e exiba no console:

*javascript*

---

let nome = "João";

let idade = 30;

let cidade = "São Paulo";

console.log("Nome:", nome);

console.log("Idade:", idade);

console.log("Cidade:", cidade);

**3. Operadores**

* **Aritméticos:** +, -, \*, /, %
* **Comparação:** ==, ===, !=, <, >
* **Lógicos:** &&, ||, !

**Exemplo:**

*javascript*

---

let a = 10;

let b = 5;

console.log(a + b); // 15

console.log(a > b); // true

console.log(a === "10"); // false (=== compara valor e tipo)

**🔹 Aritméticos**

São usados para realizar operações matemáticas entre números.

| **Operador** | **O que faz** | **Exemplo** | **Resultado** |
| --- | --- | --- | --- |
| + | Soma | 5 + 3 | 8 |
| - | Subtração | 5 - 3 | 2 |
| \* | Multiplicação | 5 \* 3 | 15 |
| / | Divisão | 10 / 2 | 5 |
| % | Resto da divisão (módulo) | 10 % 3 | 1 (resto da divisão de 10 por 3) |

**Exemplo prático:**

*javascript*

---

let a = 10;

let b = 3;

console.log(a + b); // 13

console.log(a % b); // 1

**🔹 Comparação**

São usados para comparar valores e retornar true ou false.

| **Operador** | **O que faz** | **Exemplo** | **Resultado** |
| --- | --- | --- | --- |
| == | Igualdade (com conversão de tipo) | 5 == "5" | true (5 e "5" são considerados iguais) |
| === | Igualdade estrita (sem conversão) | 5 === "5" | false (tipos diferentes) |
| != | Diferente (com conversão) | 5 != "6" | true |
| < | Menor que | 3 < 5 | true |
| > | Maior que | 10 > 7 | true |

**Dica:** Sempre prefira usar === (igualdade estrita) para evitar erros inesperados por causa da conversão automática de tipos.

**Exemplo prático:**

*javascript*

---

console.log(5 == "5"); // true

console.log(5 === "5"); // false

console.log(10 > 7); // true

**🔹 Lógicos**

Usados para combinar ou negar expressões booleanas (true ou false).

| **Operador** | **O que faz** | **Exemplo** | **Resultado** |
| --- | --- | --- | --- |
| && | E lógico (AND) - verdadeiro se todos forem verdadeiros | true && false | false |
| ` | Ou lógico (OR) - verdadeiro se pelo menos um for verdadeiro | ` | Ou lógico (OR) - verdadeiro se pelo menos um for verdadeiro |
| ! | Negação lógica - inverte o valor booleano | !true | false |

**Exemplo prático:**

*javascript*

---

let idade = 20;

let temCarteira = true;

if (idade >= 18 && temCarteira) {

console.log("Pode dirigir");

} else {

console.log("Não pode dirigir");

}

console.log(!temCarteira); // false

**Resumo simples**

| **Tipo** | **Símbolos** | **Função** |
| --- | --- | --- |
| Aritméticos | + - \* / % | Operações matemáticas básicas |
| Comparação | == === != < > | Comparar valores, retornando true ou false |
| Lógicos | `&& |  |

**4. Estruturas de Controle**

* **If / Else**

Serve para tomar decisões no código.

* if: executa um bloco se a condição for verdadeira.
* else: executa outro bloco se a condição for falsa.

**Exemplo:**

*javascript*

---

let idade = 18;

if (idade >= 18) {

console.log("Maior de idade");

} else {

console.log("Menor de idade");

}

✔ Se idade >= 18, imprime Maior de idade  
✔ Caso contrário, imprime Menor de idade

* **Switch**

Usado para comparar um valor com várias opções (cases).  
Melhor que vários if/else seguidos.

**Exemplo:**

*Javascript*

---

let cor = "azul";

switch(cor) {

case "vermelho":

console.log("Cor vermelha");

break;

case "azul":

console.log("Cor azul");

break;

default:

console.log("Outra cor");

}

✔ Se cor for "azul", executa o case azul.  
✔ break impede que os outros cases rodem.  
✔ default é executado se nenhum case for verdadeiro.

**✅ 5. Laços de Repetição**

Servem para executar um bloco de código várias vezes.

* **for**

Quando sabemos o número de repetições.

*javascript*

---

for(let i = 1; i <= 5; i++) {

console.log("Número:", i);

}

✔ Começa em i = 1  
✔ Roda enquanto i <= 5  
✔ Incrementa i++ a cada loop

* **while**

Executa enquanto a condição for verdadeira.

*javascript*

---

let i = 1;

while(i <= 5) {

console.log("Número:", i);

i++;

}

* **do...while**

Executa pelo menos uma vez, mesmo se a condição for falsa.

*javascript*

---

let i = 1;

do {

console.log("Número:", i);

i++;

} while(i <= 5);

**✅ ETAPA 2 – Funções e Arrays**

**6. Funções**

Permitem organizar código e reutilizá-lo.

* **Função tradicional:**

*javascript*

---

function soma(a, b) {

return a + b;

}

console.log(soma(2, 3)); // 5

* **Arrow Function**

**✅ O que é uma Arrow Function?**

A Arrow Function é uma forma moderna e mais curta de escrever funções em JavaScript, introduzida no ES6 (ECMAScript 2015).

Ela serve para simplificar a sintaxe das funções e, em alguns casos, ajuda com o comportamento do this (explico mais abaixo).

***Sintaxe básica***

*javascript*

---

const nomeDaFuncao = (parametros) => { /\* código \*/ }

**Regras importantes:**

* Se a função tiver apenas 1 parâmetro, os parênteses podem ser omitidos.
* Se a função tiver apenas 1 linha de retorno, as chaves {} e a palavra return podem ser omitidas.
* É ideal para funções simples.

✅ Exemplo explicado

*javascript*

---

const dobro = (x) => x \* 2;

console.log(dobro(4)); // 8

Como funciona passo a passo:

1. const dobro → cria uma variável com a função atribuída.
2. (x) → parâmetro da função (número que queremos dobrar).
3. => → indica que é uma arrow function.
4. x \* 2 → corpo da função (retorna o valor multiplicado por 2).
5. console.log(dobro(4)) → chama a função passando 4, que retorna 8.

✅ Esse código é equivalente a:

*javascript*

---

function dobro(x) {

return x \* 2;

}

console.log(dobro(4)); // 8

✅ Quando usar Arrow Function?

* Para funções curtas e simples.
* Em callbacks (como .map(), .filter(), .forEach()).
* Quando não precisar do this tradicional (pois arrow functions não têm this próprio).

✅ Exemplo com mais parâmetros

*javascript*

---

const soma = (a, b) => a + b;

console.log(soma(3, 5)); // 8

✅ Exemplo com várias linhas

Se precisar de várias linhas, use chaves e return:

*javascript*

---

const calcular = (a, b) => {

const resultado = a \* b;

return resultado;

};

console.log(calcular(3, 4)); // 12

⚠ Diferença importante:

Arrow Functions não criam um this próprio.  
Isso é útil em eventos e objetos, mas pode causar confusão se você esperar o this funcionar como numa função normal.

**a. Regras da Arrow Function**

As Arrow Functions são uma forma **mais curta e moderna** de escrever funções. Elas seguem estas regras:

✔ **Sintaxe básica:**

*javascript*

---

const nome = (param1, param2) => expressão;

✔ **Se tiver um único parâmetro, pode omitir os parênteses:**

*javascript*

---

const dobro = x => x \* 2;

✔ **Se o corpo tiver apenas uma expressão, pode omitir {} e return:**

javascript

---

const soma = (a, b) => a + b;

✔ **Se tiver várias linhas, use {} e return:**

*javascript*

---

const calcular = (a, b) => {

const resultado = a \* b;

return resultado;

};

**b. Diferenças para funções normais**

| **Característica** | **Função Tradicional** | **Arrow Function** |
| --- | --- | --- |
| Sintaxe | Mais longa | Mais curta |
| this | Tem seu próprio this | NÃO tem this próprio (usa o do contexto onde foi criada) |
| arguments | Disponível | NÃO existe em arrow function |
| Uso ideal | Funções com escopo próprio | Callbacks, funções simples |

**Sobre o this**

* **Função normal:** this depende de **como** a função é chamada.
* **Arrow Function:** this é **léxico**, ou seja, pega o valor do escopo onde foi criada.

Exemplo:

*javascript*

---

function tradicional() {

console.log(this);

}

const arrow = () => {

console.log(this);

};

tradicional(); // `this` = objeto global ou undefined (modo estrito)

arrow(); // `this` = escopo externo (não cria um novo)

**c. Exemplos com Arrays (map, filter, reduce)**

Esses métodos são perfeitos para usar Arrow Functions porque ficam muito mais legíveis.

**map() → percorre e transforma valores**

Exemplo: dobrar todos os números

*javascript*

---

const numeros = [1, 2, 3, 4];

const dobrados = numeros.map(n => n \* 2);

console.log(dobrados); // [2, 4, 6, 8]

**filter() → filtra valores com base em uma condição**

Exemplo: pegar apenas números maiores que 2

*javascript*

---

const numeros = [1, 2, 3, 4];

const maioresQueDois = numeros.filter(n => n > 2);

console.log(maioresQueDois); // [3, 4]

**reduce() → reduz o array a um único valor**

Exemplo: somar todos os números

*javascript*

---

const numeros = [1, 2, 3, 4];

const soma = numeros.reduce((acumulador, n) => acumulador + n, 0);

console.log(soma); // 10

**✅ Resumo**

* **Arrow Functions** são **mais curtas**, não têm this nem arguments.
* Excelentes para **callbacks** e funções simples.
* Combinam perfeitamente com **map, filter e reduce**.

**7. Arrays e Métodos**

Um array é uma lista de valores.

*javascript*

---

let frutas = ["maçã", "banana", "uva"];

frutas.push("laranja"); // adiciona no final

console.log(frutas.length); // 4

✔ push(): adiciona elemento  
✔ length: mostra quantidade de itens

Outros métodos importantes:

* pop() → remove último item
* shift() → remove primeiro item
* includes() → verifica se existe
* forEach() → percorre todos os elementos

**✅ ETAPA 3 – Objetos e DOM**

**8. Objetos**

Estrutura para armazenar dados em pares chave: valor.

*javascript*

---

let pessoa = {

nome: "Maria",

idade: 28,

falar: function() {

console.log("Olá, meu nome é " + this.nome);

}

};

pessoa.falar(); // "Olá, meu nome é Maria"

**9. Manipulando o DOM**

DOM = estrutura da página. Podemos alterar elementos via JavaScript.

*html*

---

<h1 id="titulo">Título</h1>

<button onclick="mudarTexto()">Clique</button>

<script>

function mudarTexto() {

document.getElementById("titulo").innerText = "Texto alterado!";

}

</script>

✔ getElementById() → pega elemento pelo ID  
✔ innerText → altera texto interno

**✅ ETAPA 4 – Intermediário**

**10. Eventos**

Eventos são ações do usuário, como clique, digitar, etc.

*html*

---

<input type="text" id="nome" placeholder="Digite seu nome">

<button id="btn">Enviar</button>

<script>

document.getElementById("btn").addEventListener("click", function() {

let nome = document.getElementById("nome").value;

alert("Olá, " + nome);

});

</script>

**11. JSON e LocalStorage**

JSON = formato para salvar dados  
LocalStorage = salva dados no navegador (mesmo fechando ele).

*javascript*

---

let usuario = { nome: "Ana", idade: 30 };

localStorage.setItem("usuario", JSON.stringify(usuario)); // salva

let dados = JSON.parse(localStorage.getItem("usuario")); // pega

console.log(dados.nome); // Ana

**12. Fetch API (Requisições HTTP)**

Para buscar dados em APIs:

*javascript*

---

fetch("https://jsonplaceholder.typicode.com/posts/1")

.then(resposta => resposta.json())

.then(dados => console.log(dados))

.catch(erro => console.log("Erro:", erro));

✅ Próximos Passos

* Praticar com pequenos projetos, como:
  + Relógio digital
  + Calculadora
  + Lista de tarefas (To-Do List)

**JAVASCRIPT NIVEL 2:**

**✅ 1. ES6+**

ES6 é a abreviação para **ECMAScript 6**, a versão de 2015 do JavaScript.  
Ele trouxe várias melhorias e novos recursos que usamos até hoje.  
O símbolo **"+"** significa que estamos incluindo **as versões seguintes** (ES7, ES8, etc.).

**Principais novidades do ES6:**

* **let e const** (variáveis)
* **Arrow Functions** (()=>{})
* **Template Strings** (uso de crase `Olá ${nome}`)
* **Destructuring** (extrair valores de objetos e arrays)
* **Módulos** (import e export)
* **Classes**
* **Promises** (para trabalhar com código assíncrono)

**✅ 2. Promises**

Uma **Promise** é um objeto que representa um valor que pode estar disponível agora, no futuro ou nunca.  
Ela é usada para **trabalhar com operações assíncronas** (como buscar dados em uma API).

**Exemplo com Promise:**

*javascript*

---

let promessa = new Promise((resolve, reject) => {

let sucesso = true;

if(sucesso) {

resolve("Operação bem-sucedida!");

} else {

reject("Erro na operação!");

}

});

promessa.then(resultado => {

console.log(resultado);

}).catch(erro => {

console.log(erro);

});

➡ **Por que usar?**  
Porque tarefas como **requisições a servidores** demoram, e você não quer travar o código esperando.

**✅ 3. Async/Await**

É uma **forma mais simples e legível** de lidar com Promises.

**Exemplo:**

*javascript*

---

async function buscarDados() {

try {

let resposta = await fetch("https://jsonplaceholder.typicode.com/posts/1");

let dados = await resposta.json();

console.log(dados);

} catch (erro) {

console.log("Erro:", erro);

}

}

buscarDados();

➡ await faz o código **esperar a Promise terminar**, mas sem travar a aplicação.

**✅ 4. Módulos**

Permitem **organizar código em arquivos separados** e reutilizá-los com import e export.

**arquivo1.js**

*javascript*

---

export function soma(a, b) {

return a + b;

}

**arquivo2.js**

*javascript*

---

import { soma } from './arquivo1.js';

console.log(soma(2, 3)); // 5

➡ Muito usado em projetos grandes e frameworks como React, Angular, Vue.

**✅ 5. Classes**

Antes, JavaScript era baseado em protótipos. Agora, com ES6, temos sintaxe de **Classes**, como em outras linguagens.

**Exemplo:**

*javascript*

---

class Pessoa {

constructor(nome, idade) {

this.nome = nome;

this.idade = idade;

}

apresentar() {

console.log(`Olá, meu nome é ${this.nome} e tenho ${this.idade} anos.`);

}

}

const p1 = new Pessoa("Ana", 28);

p1.apresentar();

➡ Classes facilitam a criação de **objetos** e o uso de **Programação Orientada a Objetos (POO)** no JS.

🔥 **Resumo prático:**

* **ES6+** = Melhorias modernas do JS
* **Promises** = Lidar com tarefas assíncronas
* **Async/Await** = Forma simplificada de trabalhar com Promises
* **Módulos** = Dividir código em partes organizadas
* **Classes** = Criar objetos com estrutura clara

**EXERCICIO:**

**✅ Estrutura dos arquivos**

bash

---

/es6\_projetos

├── exemplo1\_arrow\_template.js

├── exemplo2\_promise.js

├── exemplo3\_async.js

├── modulo1.js

├── modulo2.js

├── exemplo5\_classe.js

└── projeto\_app.js

**1. exemplo1\_arrow\_template.js**

*javascript*

---

// Exemplo 1: Arrow Function e Template String

const nome = "João";

const idade = 30;

const apresentar = () => `Meu nome é ${nome} e tenho ${idade} anos.`;

console.log(apresentar());

**2. exemplo2\_promise.js**

*javascript*

---

// Exemplo 2: Promise simples

const promessa = new Promise((resolve, reject) => {

setTimeout(() => {

resolve("Dados carregados com sucesso!");

}, 2000);

});

promessa

.then(msg => console.log(msg))

.catch(err => console.log(err));

**3. exemplo3\_async.js**

javascript

---

// Exemplo 3: Async/Await com Fetch API

async function buscarDados() {

try {

const resposta = await fetch("https://jsonplaceholder.typicode.com/users");

const dados = await resposta.json();

console.log("Usuários:", dados);

} catch (erro) {

console.log("Erro ao buscar dados:", erro);

}

}

buscarDados();

**4. modulo1.js**

*javascript*

---

// modulo1.js

export function soma(a, b) {

return a + b;

}

export function multiplica(a, b) {

return a \* b;

}

**5. modulo2.js**

*javascript*

---

// modulo2.js

import { soma, multiplica } from './modulo1.js';

console.log("Soma:", soma(5, 3));

console.log("Multiplicação:", multiplica(5, 3));

**Observação:** Para usar módulos no navegador, adicione no HTML:

*html*

---

<script type="module" src="modulo2.js"></script>

**6. exemplo5\_classe.js**

*javascript*

---

// Exemplo 5: Classe

class Carro {

constructor(modelo) {

this.modelo = modelo;

this.ligado = false;

}

ligar() {

this.ligado = true;

console.log(`${this.modelo} está ligado.`);

}

desligar() {

this.ligado = false;

console.log(`${this.modelo} está desligado.`);

}

}

const meuCarro = new Carro("Fusca");

meuCarro.ligar();

meuCarro.desligar();

**7. projeto\_app.js**

(Consumo de API com **classe + async/await**)

*javascript*

---

// Projeto Prático: Mini App com API e Classes

class UsuarioAPI {

constructor(apiUrl) {

this.apiUrl = apiUrl;

}

async buscarUsuarios() {

try {

const resposta = await fetch(this.apiUrl);

const usuarios = await resposta.json();

console.log("Lista de Usuários:");

usuarios.forEach(u => console.log(`- ${u.name}`));

} catch (erro) {

console.error("Erro ao buscar usuários:", erro);

}

}

}

const api = new UsuarioAPI("https://jsonplaceholder.typicode.com/users");

api.buscarUsuarios();

✅ Agora você pode:

* Criar uma pasta chamada **es6\_projetos**
* Criar os arquivos com os nomes acima
* Copiar e colar o código correspondente

PROJETO:  
  
 ✅ **Projeto: App de Usuários com Busca e API**

* Usa **Classes**, **Async/Await**, **Fetch API**
* Usa **Módulos** para organizar o código
* Manipula o **DOM**
* Permite **buscar usuários pelo nome**

**Estrutura do Projeto**

bash

---

/app-usuarios

├── index.html

├── style.css

├── main.js (arquivo principal)

├── api.js (módulo para requisições)

└── ui.js (módulo para manipulação do DOM)

**1. index.html**

*html*

---

<!DOCTYPE html>

<html lang="pt-br">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>App de Usuários</title>

<link rel="stylesheet" href="style.css">

</head>

<body>

<div class="container">

<h1>Lista de Usuários</h1>

<input type="text" id="busca" placeholder="Buscar por nome...">

<ul id="lista"></ul>

</div>

<!-- Importar script principal como módulo -->

<script type="module" src="main.js"></script>

</body>

</html>

**2. style.css**

*css*

---

body {

font-family: Arial, sans-serif;

background: #f4f4f4;

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

margin: 0;

}

.container {

width: 400px;

background: #fff;

padding: 20px;

border-radius: 10px;

box-shadow: 0 0 10px rgba(0,0,0,0.1);

}

h1 {

text-align: center;

}

input {

width: 100%;

padding: 10px;

margin-bottom: 15px;

border: 1px solid #ccc;

border-radius: 5px;

}

ul {

list-style: none;

padding: 0;

}

li {

padding: 10px;

background: #eee;

margin-bottom: 5px;

border-radius: 5px;

}

**3. api.js (módulo de requisição)**

*javascript*

---

// api.js

export default class UsuarioAPI {

constructor(apiUrl) {

this.apiUrl = apiUrl;

}

async buscarUsuarios() {

try {

const resposta = await fetch(this.apiUrl);

return await resposta.json();

} catch (erro) {

console.error("Erro ao buscar usuários:", erro);

return [];

}

}

}

**4. ui.js (módulo de interface)**

*javascript*

---

// ui.js

export default class UI {

constructor(listaElement) {

this.listaElement = listaElement;

}

exibirUsuarios(usuarios) {

this.listaElement.innerHTML = "";

usuarios.forEach(usuario => {

const li = document.createElement("li");

li.textContent = usuario.name;

this.listaElement.appendChild(li);

});

}

}

**5. main.js (arquivo principal)**

*javascript*

---

// main.js

import UsuarioAPI from './api.js';

import UI from './ui.js';

const api = new UsuarioAPI("https://jsonplaceholder.typicode.com/users");

const listaElement = document.getElementById("lista");

const ui = new UI(listaElement);

const inputBusca = document.getElementById("busca");

let todosUsuarios = [];

async function inicializar() {

todosUsuarios = await api.buscarUsuarios();

ui.exibirUsuarios(todosUsuarios);

}

inputBusca.addEventListener("input", () => {

const termo = inputBusca.value.toLowerCase();

const filtrados = todosUsuarios.filter(u => u.name.toLowerCase().includes(termo));

ui.exibirUsuarios(filtrados);

});

inicializar();

**✅ Como usar**

1. Crie a pasta **app-usuarios**
2. Adicione os arquivos acima
3. Abra index.html no navegador
4. O app:
   * Carrega usuários de uma API pública
   * Exibe lista na tela
   * Permite **filtrar por nome em tempo real**

🔥 Esse projeto **integra todos os conceitos modernos do JavaScript**:

* **Classes** → UsuarioAPI e UI
* **Async/Await** → busca de dados com Fetch
* **Módulos ES6** → import e export
* **DOM** → exibição dinâmica
* **Busca dinâmica** → evento input

**Guia Prático: App de Usuários com ES6+**

**✅ Objetivo**

Criar um aplicativo que:

* Carrega **usuários de uma API pública**
* Exibe os usuários dinamicamente no navegador
* Permite **buscar por nome em tempo real**

**Conceitos aplicados:**  
✔ **Classes**  
✔ **Async/Await**  
✔ **Fetch API**  
✔ **Módulos (import/export)**  
✔ **Manipulação do DOM**  
✔ **Eventos**

**✅ Estrutura do Projeto**

bash

---

/app-usuarios

├── index.html

├── style.css

├── main.js

├── api.js

└── ui.js

**✅ Passo 1: HTML**

Crie o arquivo **index.html**:

*html*

---

<!DOCTYPE html>

<html lang="pt-br">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>App de Usuários</title>

<link rel="stylesheet" href="style.css">

</head>

<body>

<div class="container">

<h1>Lista de Usuários</h1>

<input type="text" id="busca" placeholder="Buscar por nome...">

<ul id="lista"></ul>

</div>

<!-- Importar script principal como módulo -->

<script type="module" src="main.js"></script>

</body>

</html>

**✅ Passo 2: CSS**

Crie o arquivo **style.css**:

*css*

---

body {

font-family: Arial, sans-serif;

background: #f4f4f4;

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

margin: 0;

}

.container {

width: 400px;

background: #fff;

padding: 20px;

border-radius: 10px;

box-shadow: 0 0 10px rgba(0,0,0,0.1);

}

h1 {

text-align: center;

}

input {

width: 100%;

padding: 10px;

margin-bottom: 15px;

border: 1px solid #ccc;

border-radius: 5px;

}

ul {

list-style: none;

padding: 0;

}

li {

padding: 10px;

background: #eee;

margin-bottom: 5px;

border-radius: 5px;

}

**✅ Passo 3: api.js (Módulo para API)**

*javascript*

---

// api.js

export default class UsuarioAPI {

constructor(apiUrl) {

this.apiUrl = apiUrl;

}

async buscarUsuarios() {

try {

const resposta = await fetch(this.apiUrl);

return await resposta.json();

} catch (erro) {

console.error("Erro ao buscar usuários:", erro);

return [];

}

}

}

**✅ Passo 4: ui.js (Módulo para interface)**

*javascript*

---

// ui.js

export default class UI {

constructor(listaElement) {

this.listaElement = listaElement;

}

exibirUsuarios(usuarios) {

this.listaElement.innerHTML = "";

usuarios.forEach(usuario => {

const li = document.createElement("li");

li.textContent = usuario.name;

this.listaElement.appendChild(li);

});

}

}

**✅ Passo 5: main.js (Arquivo principal)**

*javascript*

---

// main.js

import UsuarioAPI from './api.js';

import UI from './ui.js';

const api = new UsuarioAPI("https://jsonplaceholder.typicode.com/users");

const listaElement = document.getElementById("lista");

const ui = new UI(listaElement);

const inputBusca = document.getElementById("busca");

let todosUsuarios = [];

async function inicializar() {

todosUsuarios = await api.buscarUsuarios();

ui.exibirUsuarios(todosUsuarios);

}

inputBusca.addEventListener("input", () => {

const termo = inputBusca.value.toLowerCase();

const filtrados = todosUsuarios.filter(u => u.name.toLowerCase().includes(termo));

ui.exibirUsuarios(filtrados);

});

inicializar();

**✅ Como funciona**

* **api.js** → Classe para buscar dados da API usando **async/await**
* **ui.js** → Classe para atualizar o DOM com a lista de usuários
* **main.js** → Integra os módulos, carrega usuários e aplica **filtro em tempo real**

**✅ Execução**

1. Crie os arquivos conforme a estrutura.
2. Abra o **index.html** no navegador.
3. Veja a lista carregando da API e use o campo de busca para filtrar.

**✅ Exercícios Extras**

1. Adicione um botão **Recarregar** para atualizar a lista.
2. Mostre também o **email** de cada usuário na lista.
3. Crie uma funcionalidade para **ordenar os usuários por nome**.
4. Ao clicar em um nome, exiba **detalhes do usuário em um modal**.
5. Transforme a busca para também filtrar por **email**.

🔥 Este projeto é um **ótimo exemplo prático de ES6+**, unindo **API, classes, módulos e manipulação do DOM**.