



PROGRAMAÇÃO FRONT END

ANDRÉ ABDALA NOEL

ACESSE AQUI ESTE
MATERIAL DIGITAL!

EXPEDIENTE

Coordenador(a) de Conteúdo

Silvio César de Castro

Projeto Gráfico e Capa

Arthur Cantareli Silva

Editoração

Lilian Andreia Hasse

Design Educacional

Aguinaldo José Lorca Ventura Júnior

Revisão Textual

Ariane Andrade Fabreti

Ilustração

Bruno Cesar Pardinho Figueiredo

Eduardo Aparecido Alves

Geison Ferreira da Silva

Fotos

Shutterstock e Envato

FICHA CATALOGRÁFICA

N964 Núcleo de Educação a Distância. **NOEL**, André Abdala

Programação Front End / André Abdala Noel. - Florianópolis, SC:
Arqué, 2025.

232 p.

ISBN papel 978-65-279-1070-1

ISBN digital 978-65-279-1069-5

1. Programação 2. Front end 3. EaD. I. Título.

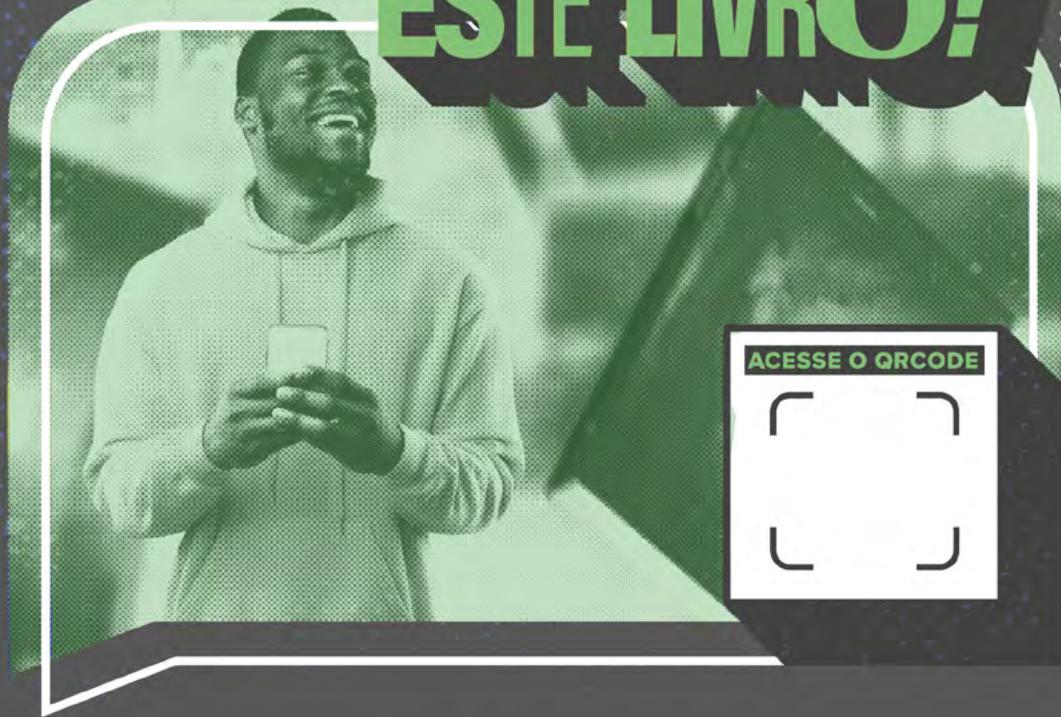
CDD - 005.12

Bibliotecária: Leila Regina do Nascimento - CRB- 9/1722.

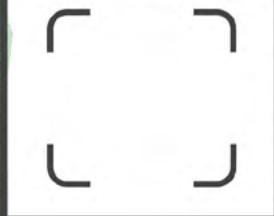
Ficha catalográfica elaborada de acordo com os dados fornecidos pelo(a) autor(a).

Impresso por:

AVALIE ESTE LIVRO!



ACESSE O QR CODE



CRIAR MOMENTOS DE APRENDIZAGENS
INESQUECÍVEIS É O NOSSO OBJETIVO E POR ISSO,
GOSTARIAMOS DE SABER COMO FOI SUA EXPERIÊNCIA.

Conta para nós! leva menos de 2 minutos. Vamos lá?!

DIGITE O CÓDIGO

02511836

Aa

RESPOnda A
PESQUISA

... ?

...



RECURSOS DE IMERSÃO



PENSANDO JUNTOS

Este item corresponde a uma proposta de reflexão que pode ser apresentada por meio de uma frase, um trecho breve ou uma pergunta.



APROFUNDANDO

Utilizado para temas, assuntos ou conceitos avançados, levando ao aprofundamento do que está sendo trabalhado naquele momento do texto.



EU INDICO

Utilizado para agregar um conteúdo externo.



ZOOM NO CONHECIMENTO

Utilizado para desmistificar pontos que possam gerar confusão sobre o tema. Após o texto trazer a explicação, essa interlocução pode trazer pontos adicionais que contribuam para que o estudante não fique com dúvidas sobre o tema.

PRODUTOS AUDIOVISUAIS

Os elementos abaixo possuem recursos audiovisuais. Recursos de mídia disponíveis no conteúdo digital do ambiente virtual de aprendizagem.



PLAY NO CONHECIMENTO

Professores especialistas e convidados, ampliando as discussões sobre os temas por meio de fantásticos podcasts.



EM FOCO

Utilizado para aprofundar o conhecimento em conteúdos relevantes utilizando uma linguagem audiovisual.



INDICAÇÃO DE FILME

Uma dose extra de conhecimento é sempre bem-vinda. Aqui você terá indicações de filmes que se conectam com o tema do conteúdo.



INDICAÇÃO DE LIVRO

Uma dose extra de conhecimento é sempre bem-vinda. Aqui você terá indicações de livros que agregarão muito na sua vida profissional.



CAMINHOS DE APRENDIZAGEM

7

UNIDADE 1

INTRODUÇÃO À PROGRAMAÇÃO FRONT-END	8
--	---

27

UNIDADE 2

CRIANDO PÁGINAS COM HTML E CSS	28
--	----

APROFUNDANDO O LAYOUT	52
---------------------------------	----

75

UNIDADE 3

DANDO VIDA AOS SISTEMAS COM JAVASCRIPT.	76
---	----

CRIANDO SISTEMAS PARA SEREM UTILIZADOS	102
--	-----

129

UNIDADE 4

SISTEMAS QUE CONVERSAM ENTRE SI	130
---	-----

ACESSIBILIDADE PARA TODOS.	158
------------------------------------	-----

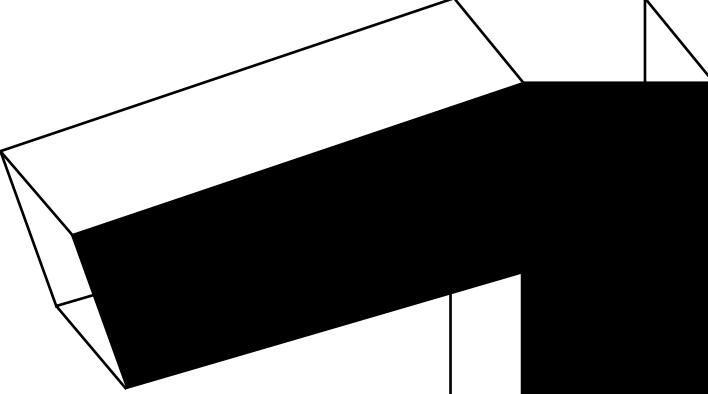
185

UNIDADE 5

OTIMIZAÇÃO DE DESEMPENHO PARA WEBSITES RÁPIDOS.	186
---	-----

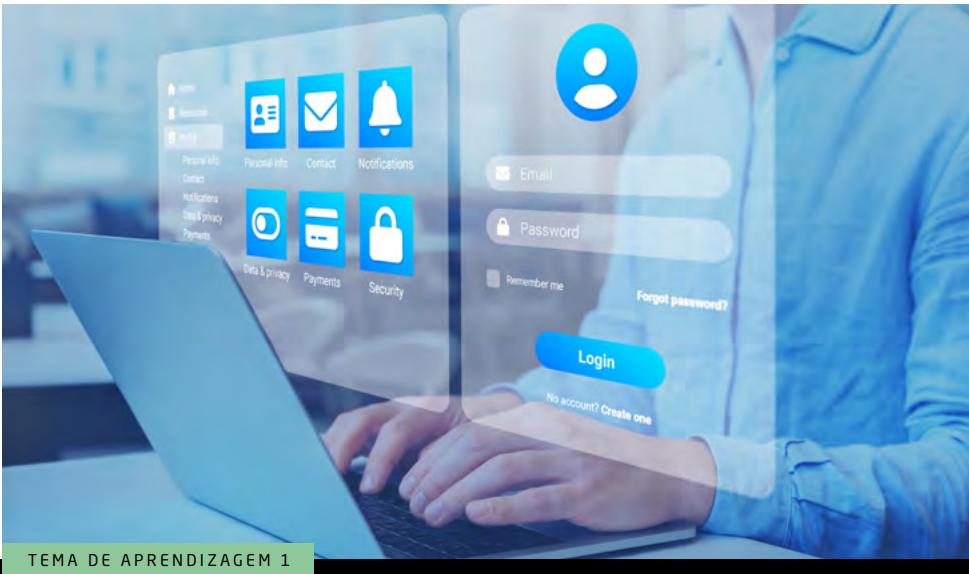
TESTES E DEPURAÇÃO EM DESENVOLVIMENTO FRONT-END	208
---	-----





*uni
dade*





TEMA DE APRENDIZAGEM 1

INTRODUÇÃO À PROGRAMAÇÃO FRONT-END

MINHAS METAS

- Entender o que é desenvolver para web.
- Apresentar o desenvolvimento front-end e sua importância.
- Conhecer as linguagens utilizadas e os propósitos de cada uma.
- Discutir o funcionamento da arquitetura cliente-servidor.
- Diferenciar programação front-end e design.
- Preparar o ambiente para iniciar o desenvolvimento.
- Conversar sobre a diferença entre utilizar ou não *frameworks* de desenvolvimento.

INICIE SUA JORNADA

Olá, estudante em busca de novos conhecimentos. Tudo bem com você? Procure um lugar relaxado, mas com uma boa dose de concentração, e iniciaremos uma nova jornada, mergulhando no conhecimento da programação front-end.

Antes de começar, refletiremos sobre o seu dia. Pense nas suas atividades diárias e responda: o quanto dessas atividades envolve ou depende da internet? Vivemos em uma época em que a maioria das nossas atividades passa por dispositivos e aplicativos conectados, nossos dados são trocados para lá e para cá a todo momento, podemos inclusive passar por situações difíceis, se a bateria do celular acabar ou se não tivermos sinal para uma conexão.

Portanto, vemos que, a cada dia, precisamos estar mais presentes na web, assim como muitos potenciais clientes (ou empregadores) precisam de soluções que funcionem online para movimentar o dia a dia de milhares de pessoas. O que precisamos para isso? Entenderemos como a web funciona e qual é o papel de cada parte do desenvolvimento web.

PLAY NO CONHECIMENTO

Em nosso episódio de podcast, ouça as diferentes possibilidades que o desenvolvimento front-end nos proporciona e a importância de uma boa interação com o usuário. **Recursos de mídia disponíveis no conteúdo digital do ambiente virtual de aprendizagem.**

VAMOS RECORDAR?

A programação, em geral, é realizada a partir de uma sequência de instruções bem definidas e não ambíguas. Quando falamos de desenvolvimento web, principalmente o desenvolvimento front-end, estamos falando de programas criados para serem executados em um navegador web, que funciona como um interpretador de código. Por isso, aproveitaremos o conhecimento prévio que podemos ter obtido com outras linguagens de programação e ajustaremos à sintaxe do Javascript a questão de declaração de variáveis, estruturas de controle, definição de funções e muito mais. Para entender melhor como funciona um interpretador de código, assista ao seguinte vídeo: <https://www.youtube.com/watch?v=XjoYSTMgjMw>.

DESENVOLVA SEU POTENCIAL

Dizem que quando algo “cai na rede”, não tem mais volta. Esse algo que cai na rede e chega em instantes no mundo todo, pode viralizar ou não. Essa situação talvez seja ruim, dependendo do conteúdo, mas talvez seja algo muito bom. Imagine um sistema que você fez viralizando, alcançando o mundo inteiro graças à internet.

Nos anos 1990, computadores munidos com o sistema operacional Windows 3.1 costumavam carregar, entre outras coisas, alguns pequenos programas escritos em QBasic, escritos para o MS-DOS 5. Um deles era o jogo *Gorillas* para dois jogadores, no qual cada jogador era um gorila em cima de um prédio, que devia jogar bananas para acertar o outro, a partir da perspectiva de ângulo e velocidade.

A ideia desse jogo deu base para inúmeras variações, como o *Worms*. Gostaria, porém, de destacar uma empresa que, após mais de 50 tentativas de criação de jogos e a quase desistência, viralizou com seu jogo *Angry Birds*, ao aproveitar as características recentes de dispositivos móveis e as possibilidades da web para difundir pelo mundo todo o game e conquistar jogadores que viraram fãs, pois começaram a consumir os mais diferentes tipos de produtos que carregavam a marca.

Antigamente, os softwares eram vendidos em caixas, a partir de mídias como disquetes, CD-ROM ou DVD. Hoje, é possível fazer, literalmente, no computador do quarto de casa, o lançamento de um sistema que pode rodar o mundo todo e estar disponível instantaneamente na web.

VOU PROGRAMAR PARA WEB. MAS O QUE É A WEB?

“Quando cheguei aqui, tudo era mato!”, disse qualquer pessoa de alguma geração antes da sua sobre qualquer coisa que, hoje, seja mais simples. É bom entender, porém, de onde viemos para saber aonde vamos.

A internet, a famosa “rede mundial de computadores”, teve seu início na década de 1960, mas a web (*www - world wide web*) surgiu apenas em 1989, com lançamento público em 1991.

Mas não são a mesma coisa? Basicamente, a internet é a rede que interconecta tudo, possibilitando a comunicação entre os diferentes dispositivos, enquanto a web roda sobre a internet, criando uma camada para troca de informações via hipertexto, um tipo de documento que permite texto, mídias (imagens, vídeos, áudios) e *hyperlinks*, o qual ficou conhecido como site ou página de internet.

APROFUNDANDO

Em 1969 foi realizada a primeira conexão de internet que, na época, se chamaava **Arpanet**. Foram conectados dois computadores através do uso de modems, entre a Universidade de Stanford, em Massachusetts, na costa leste dos Estados Unidos, e a Universidade da Califórnia, em Los Angeles, na costa oeste. A rede teve rápido crescimento até que, em 1982, a Arpanet, que já era a maior dentre as redes do tipo, resolveu aderir ao Protocolo Internet (*Internet Protocol - IP*), sendo seguida pelas demais redes, e tornou-se conhecida pelo nome **internet**. Em 1989, Tim Berners-Lee foi o programador responsável por desenvolver o protocolo HTTP, que foi deu origem à web, e o disponibilizou como domínio público, para que todos tivessem acesso. No ano seguinte, a rede Arpanet foi desligada (Costa Bisneto, 2003).

Basicamente, a inovação da web surgiu com base em documentos interconectados via links, onde um documento aponta para outros, formando uma grande teia de documentos, daí o nome “web”, que significa “teia”, em inglês.

Apesar de parecer muito simples, foi isso que permitiu o surgimento da maioria dos serviços os quais usamos hoje, como Google, Facebook e inúmeros outros.

ARQUITETURA CLIENTE-SERVIDOR

Quando falamos em programação web, estamos falando da programação que utiliza essa base de **hipertexto**, portanto, dados que trafegam sobre o protocolo **HTTP (HyperText Transfer Protocol)**.

De cada lado do protocolo, existem dois principais atores: o cliente e o servidor. O **cliente**, também chamado de usuário, é o lado de quem está utilizando o sistema, é a pessoa usando o WhatsApp, o YouTube ou qualquer outro serviço via web. Na outra ponta, está o **servidor**: ele é quem escuta o protocolo e espera requisições para responder com o conteúdo desejado.

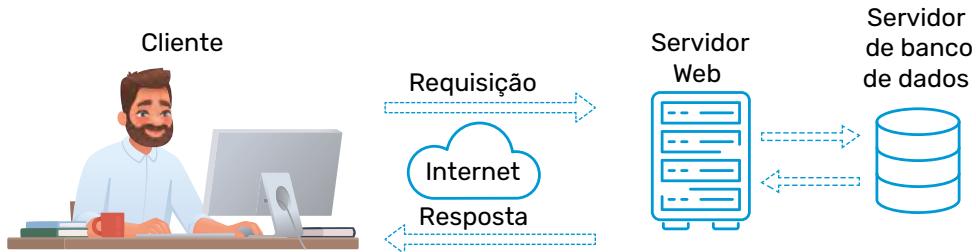


Figura 1 - Modelo de arquitetura cliente-servidor / Fonte: o autor.

Descrição da Imagem: diagrama de funcionamento da arquitetura-servidor, com o usuário representando o cliente, enquanto troca informações com o servidor web por meio de requisições e respostas via internet. Nesse ínterim, o servidor web também troca informações com o servidor de banco de dados. Fim da descrição.

Na Figura 1, temos uma representação de como funciona o modelo de arquitetura cliente-servidor. No lado do cliente, há um usuário acessando os sistemas a partir de um navegador web. O navegador é um **interpretador** que renderiza o documento de hipertexto, com seus estilos, mídias e tudo mais. Também é nele que será executada a parte da programação **front-end**.

O usuário ou o próprio sistema que roda no front-end envia requisições via internet para algum servidor web. O servidor é um processo que fica em execução, escutando o tempo todo, aguardando requisições. Ao receber uma requisição, o servidor a processa e retorna uma resposta com o conteúdo requisitado. A parte executada no servidor é responsabilidade da programação **back-end**.

Para encontrar o servidor que responderá à requisição, o front-end faz as solicitações a partir de um endereço que diz onde está a máquina a qual está executando o processo do servidor. Esse processo é feito a partir de um sistema de endereçamento distribuído, chamado **DNS** (*Domain Name System*), que traduz endereços textuais, por exemplo, www.google.com, para endereços IP, que são sequências de números bem mais difíceis de lembrar ou de associar a sites e empresas (Kurose, 2013).

Ainda no lado do back-end, o servidor web pode interagir com sistemas de bancos de dados, consultando ou alterando as informações lá contidas, por meio de chamadas ao sistema de gerenciamento de bancos de dados.

Para facilitar a comunicação do front-end com o back-end, é comum que as requisições sejam realizadas através das **API**, que são “um conjunto de características e regras existentes em uma aplicação que possibilitam interações com essa através de um software - ao contrário de uma interface de usuário humana” (MDN Contributors, 2023a, on-line).

É comum que as requisições sejam realizadas através das API



APIs são conjuntos prontos de blocos de construção de código que permitem que um desenvolvedor implemente programas que seriam difíceis ou impossíveis de implementar. Eles fazem o mesmo para a programação que os kits de móveis prontos para a construção de casas - é muito mais fácil pegar os painéis prontos e parafusá-los para formar uma estante de livros do que para elaborar o design, sair e encontrar a madeira, cortar todos os painéis no tamanho e formato certos, encontrar os parafusos de tamanho correto e depois montá-los para formar uma estante de livros (MDN Contributors, 2024, on-line).

Em geral, o servidor fornece uma API, onde existem as regras de como as informações serão solicitadas e retornadas. O front-end faz as requisições, recebe as respostas com apenas as informações necessárias e monta a interface para o usuário, a fim de exibir as informações, gerenciando, assim, as interações com o usuário final.



FRONT-END É PROGRAMAÇÃO OU DESIGN?

O principal trabalho do desenvolvimento front-end é o de fornecer uma interface para o usuário interagir com o sistema, possibilitando a entrada e saída de dados de forma intuitiva, agradável e coerente.

Então, quer dizer que o trabalho do front-end é fazer o design? Embora seja uma tarefa bem focada em interfaces e interação com o usuário, desenvolver para o front-end não é o mesmo que design, não é desenhar telas. Envolve também bastante programação e transformar algo desenhado por um designer em uma interface programada para ser dinâmica.

O desenvolvimento front-end envolve, basicamente, três linguagens principais: HTML, CSS e Javascript.

HTML (*HYPertext Markup Language*, ou LINGUAGEM DE MARCAÇÃO DE HIPERTEXTO)

Como o nome diz, é uma linguagem de marcação e não de programação. É um subconjunto do XML, utilizando tags para criar documentos dinâmicos que possuem elementos como texto, mídias e links.

CSS (*Cascading Styles Sheet*, ou FOLHA DE ESTILOS EM CASCATA)

Uma linguagem de estilização, na qual são criadas as regras de formatação de nossos sites, ou seja, dos documentos HTML.

JAVASCRIPT

Linguagem de programação interpretada pelo navegador, o que permite tornar as páginas dinâmicas, com a possibilidade de manipulação dos elementos HTML e dos estilos CSS.

Ainda que o desenvolvimento front-end atual envolva diversos *frameworks*, ferramentas e geradores de código, basicamente, os códigos são convertidos para HTML, CSS e Javascript, que são as linguagens cujos navegadores entendem. A combinação dessas três linguagens fornece o pilar fundamental para o desenvolvimento front-end.

Cada linguagem dedica-se a uma parte específica do desenvolvimento. Se compararmos com a construção de uma casa, por exemplo, seria como se o HTML fosse usado para subir as paredes, a estrutura em geral, com o CSS, faríamos os acabamentos, pinturas e embelezamento, enquanto o Javascript envolveria as partes elétrica e hidráulica, que permitem uma utilização mais agradável, conveniente e dinâmica, para ligar equipamentos, tomar banho etc.



Um dos usos mais importantes de JavaScript é para a criação e modificação dinâmica de documentos HTML. JavaScript define uma hierarquia de objetos que casa com um modelo hierárquico de um documento HTML, definido pelo Document Object Model (DOM). Elementos de um documento HTML são acessados por meio desses objetos, fornecendo a base para o controle dinâmico dos elementos dos documentos (Sebesta, 2011, p. 121).

O desenvolvimento em Javascript fornece dinamicidade, permitindo que os elementos sejam modificados em tempo de execução, o que torna as páginas web diferentes de simples documentos. Esse processo possibilita inúmeras oportunidades de desenvolvimento de aplicações.

Por ser uma linguagem de programação, o Javascript utiliza o que você talvez já tenha aprendido em outras linguagens, como lógica de programação, estruturas de controle, estruturas de dados etc. Inclusive, a sintaxe de Javascript baseia-se na sintaxe da linguagem C, assim como outras linguagens, como Java e PHP, facilitando o entendimento, caso você já tenha familiaridade com algumas delas.

O QUE PRECISO PARA DESENVOLVER EM FRONT-END?

Visto que o desenvolvimento front-end gira em torno de três linguagens principais, o que precisamos ter para esse desenvolvimento? Como o processamento das informações é realizado no navegador, a parte do interpretador já existe em qualquer computador ou smartphone. Além disso, precisamos apenas de um editor para escrever os códigos, e existem diversos editores disponíveis e gratuitos.

Durante o processo, costumamos agregar algumas ferramentas e extensões para facilitar o trabalho, mas não precisa se preocupar com elas agora, pois, à medida que a necessidade aparecer, você poderá instalar novas ferramentas.

Sobre o editor de códigos, é possível utilizar qualquer editor que edite textos puros, porém é recomendado o uso de algum editor específico para programação, com vários auxílios para proporcionar um desenvolvimento mais produtivo. Seguem algumas opções:

VS CODE

Atualmente, o editor mais popular. Desenvolvido pela Microsoft, possui código aberto bem como inúmeras facilidades e extensões, incluindo um suporte ao Git, servidor web para testes e suporte a diversas linguagens de programação.

SUBLIME TEXT

Um editor de códigos leve e multiplataforma, muito utilizado por desenvolvedores de diferentes linguagens.

ATOM

Editor de código aberto, como o VS Code. Desenvolvido com HTML, CSS e Javascript, possibilita a instalação de extensões desenvolvidas em Node.js.

BRACKETS

Desenvolvido pela Adobe e disponibilizado em código aberto, é focado em desenvolvimento web, principalmente front-end.

NOTE PAD++

Um editor simples e poderoso, de código aberto, com alta possibilidade de personalização.

O principal impacto de um editor ou de outro está no momento da escrita do código, em relação à produtividade e à facilidade. Você pode, porém, escrever seus sistemas em qualquer editor, pois o código não depende de nada específico de nenhum ambiente de desenvolvimento.

Após a criação do código, os navegadores web modernos possuem em si diversas ferramentas para inspecionar o código e identificar possíveis problemas, funcionando também como um ambiente de desenvolvimento.

No decorrer do aprendizado, você também aprenderá a lidar com essas ferramentas do navegador.

Framework ou não framework? Eis a questão!

O desenvolvimento front-end envolve, muitas vezes, a utilização de *frameworks* de desenvolvimento. **Frameworks** são conjuntos de códigos e ferramentas para auxiliar o desenvolvimento e reaproveitar trechos que são comuns a sistemas diferentes, trazendo benefícios como “[...] agilidade, produtividade, padrões de projeto de forma simples e descomplicada e maior qualidade ao produto final” (Ferrão; Macedo; Kreutz, 2018).



No dia a dia, é recomendado o uso de um *framework* pelas vantagens citadas e, provavelmente, você deve ser orientado pela sua empresa a utilizar as ferramentas que já utilizam, porém, para melhor aprendizado, focaremos no desenvolvimento sem a utilização de *frameworks*, por um conjunto de motivos: primeiro, entender os fundamentos do desenvolvimento web é muito importante. Mesmo que as ferramentas mudem, a base é a mesma, e conhecer a linguagem de forma “pura” possibilita maior domínio sobre o código que é escrito.

Além disso, existem diversos bons *frameworks* de desenvolvimento para front-end (diz a lenda que, a cada minuto, surgem mais cinco), portanto, o conhecimento mais específico em determinada ferramenta é algo que fica defasado de forma muito mais rápida do que o conhecimento na linguagem. Conhecer a linguagem básica, em nosso caso, Javascript, nos permite entender o próprio funcionamento do que é feito no *framework* e facilita a troca de ferramentas.

Existem *frameworks* e bibliotecas para a parte visual, a fim de simplificar a estilização via CSS, que valem um pouco do seu tempo de pesquisa, porque são muito utilizados em empresas. Podemos destacar os pré-processadores de CSS mais usados: Sass e Less, também a biblioteca Bootstrap, e o *framework* Tailwind, os quais agilizam muito o trabalho de colocar um sistema bonito no ar.

Em relação ao Javascript, existem também várias opções de *frameworks*, tanto no front-end quanto no back-end, porém, como nosso foco está no front-end, comentaremos apenas esses. Vale tirar um tempo para estudar o funcionamento de ferramentas como React.js, Vue.js, Angular e Next.js., pois são, atualmente, muito utilizadas por empresas, mas faça isso após os estudos deste material, porque entender a base pode impulsionar muito o seu trabalho.

Tenha em mente que essas ferramentas estão sempre se atualizando e trazendo novos recursos para o desenvolvimento mais ágil e o melhor aproveitamento dos recursos disponíveis nos dispositivos, por isso um material direcionado a uma ferramenta deteriora-se muito mais rapidamente do que um material que ensina os fundamentos de uma tecnologia. A base da linguagem e da programação mantém-se, e o seu conhecimento mais profundo, estudante, permite também que você tenha mais facilidade de se atualizar no decorrer da carreira.

**Entender os
fundamentos do
desenvolvimento
web é muito
importante**

**EM FOCO**

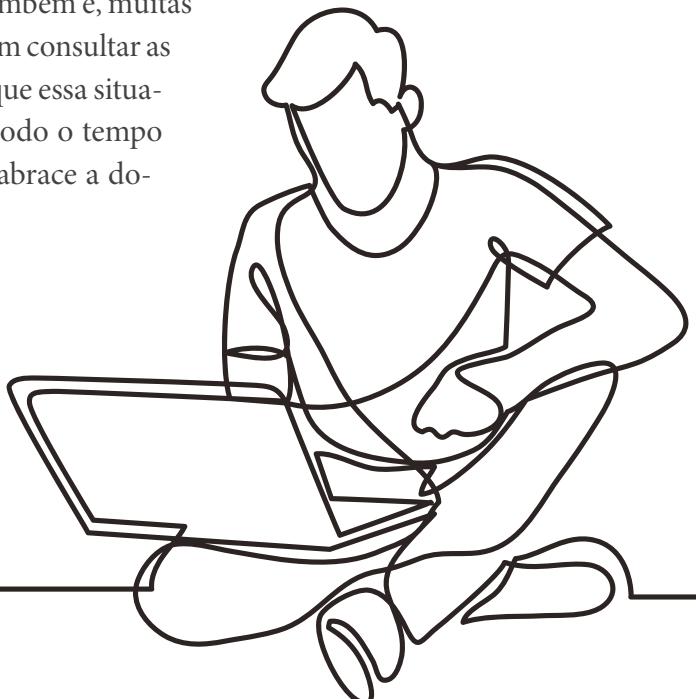
Assista à videoaula para mais informações e conhecimentos sobre este tema de aprendizagem. **Recursos de mídia disponíveis no conteúdo digital do ambiente virtual de aprendizagem.**

NOVOS DESAFIOS

Entender o funcionamento do desenvolvimento web, tanto front-end quanto back-end, faz muita diferença para o dia a dia nessa profissão. Como são partes que se comunicam constantemente, é ótimo que uma área entenda o que é esperado de uma e de outra.

Tenha em mente que todo aprendizado de programação demanda prática. Quando vemos algum código pronto ou alguém ensinando um código, tudo faz muito sentido e parece óbvio, porém, quando precisamos escrever sozinhos, percebemos o que retivemos do conhecimento e o que precisamos consultar novamente.

Não que seja um problema, é inclusive uma prática comum consultar referências. Uma pessoa experiente em programação também é, muitas vezes, uma pessoa experiente em consultar as referências de linguagens, porque essa situação nos acompanha durante todo o tempo de carreira. Em linhas gerais, abrace a documentação e seja feliz!



VAMOS PRATICAR

1. "JavaScript é uma linguagem de programação que permite implementar funcionalidades mais complexas em páginas web. Sempre que uma página web faz mais do que apenas mostrar informações estáticas para você - ela mostra em tempo real conteúdos atualizados, mapas interativos, animações gráficas em 2D/3D, vídeos etc. - você pode apostar que o Javascript provavelmente está envolvido" (MDN Contributors, 2023b, on-line).

Qual das seguintes tecnologias é essencial para o desenvolvimento front-end?

- a) PHP.
 - b) SQL.
 - c) Java.
 - d) HTML.
 - e) Python.
2. "Um uso muito comum do JavaScript é modificar dinamicamente HTML e CSS para atualizar uma interface do usuário, por meio da API do Document Object Model" (MDN Contributors, 2024, on-line).

A respeito do desenvolvimento web, avalie as afirmações, a seguir:

- I - A linguagem HTML serve para definir a estrutura do documento, mas não a programação.
- II - A linguagem Javascript permite manipular os elementos definidos no documento HTML.
- III - A linguagem CSS não é uma linguagem de programação, mas permite estilizar os elementos do documento.

É correto o que se afirma em:

- a) I, apenas.
- b) III, apenas.
- c) I e II, apenas.
- d) II e III, apenas.
- e) I, II e III.

VAMOS PRATICAR

3. "HTML (Linguagem de Marcação de HiperTexto) é o bloco de construção mais básico da web. Define o significado e a estrutura do conteúdo da web. Outras tecnologias além do HTML geralmente são usadas para descrever a aparência/apresentação (CSS) ou a funcionalidade/comportamento (JavaScript) de uma página da web" (MDN Contributors, 2023c, on-line).

Por que é vantajoso aprender desenvolvimento front-end sem o uso, inicialmente, de *frameworks*?

- a) Facilita a transição para back-end.
- b) Permite o foco exclusivo em design.
- c) Elimina a necessidade de aprender Javascript.
- d) Oferece compreensão superficial das tecnologias.
- e) Proporciona compreensão mais profunda e controle sobre o código.

REFERÊNCIAS

COSTA BISNETO, P. L. O. **A história da internet.** [S. l.: s. n.]: 2003. Disponível em: <http://www.pedroom.com.br/portal/vitae/download/cientificos/03%20A%20Historia%2oda%20Internet.pdf>. Acesso em: 8 out. 2024.

FERRÃO, I. G.; MACEDO, D. J. de; KREUTZ, D. Investigação do impacto de frameworks de desenvolvimento de software na segurança de sistemas web. In: SBSeg - SIMPÓSIO BRASILEIRO DE SEGURANÇA DA INFORMAÇÃO E DE SISTEMAS COMPUTACIONAIS, 18., Natal, 2018. **Anais** [...]. Natal: UFRN, 2018.

KUROSE, J. F. **Redes de computadores e a internet:** uma abordagem top-down. 6. ed. São Paulo: Pearson Education do Brasil, 2013.

MDN CONTRIBUTORS. Aprendendo desenvolvimento web: O que é JavaScript? **MDN Web Docs**, [s. l.], 29 jul. 2024. Disponível em: https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/First_steps/What_is_JavaScript. Acesso em: 8 out. 2024.

MDN CONTRIBUTORS. Glossário do MDN Web Docs: API. **MDN Web Docs**, [s. l.], 19 nov. 2023a. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Glossary/API>. Acesso em: 8 out. 2024.

MDN CONTRIBUTORS. Aprendendo desenvolvimento web: JavaScript **MDN Web Docs**, [s. l.], 3 ago. 2023b. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript>. Acesso em: 8 out. 2024.

MDN CONTRIBUTORS. HTML: Linguagem de Marcação de Hipertexto. **MDN Web Docs**, [s. l.], 1 out. 2023c. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTML>. Acesso em: 8 out. 2024.

SEBESTA, R. W. **Conceitos de linguagens de programação**. 9. ed. Porto Alegre: Bookman, 2011.

CONFIRA SUAS RESPOSTAS

1. Alternativa D.

HTML é uma tecnologia fundamental para o desenvolvimento front-end, pois define a estrutura e o conteúdo das páginas web.

A alternativa A está incorreta, pois PHP é utilizado no back-end.

A alternativa B está incorreta, pois SQL é utilizada no banco de dados.

A alternativa C está incorreta, pois Java é utilizada no back-end.

A alternativa E está incorreta, pois Python é utilizado no back-end.

2) Alternativa E.

A afirmativa I está correta, pois a linguagem HTML serve para definir a estrutura do documento, mas não a programação.

A afirmativa II está correta, pois a linguagem Javascript permite manipular os elementos definidos no documento HTML.

A afirmativa III está correta, pois a CSS não é uma linguagem de programação, mas permite estilizar os elementos do documento.

2. Alternativa E.

Aprender desenvolvimento front-end sem *frameworks* permite aos desenvolvedores entenderem profundamente como as tecnologias fundamentais (HTML, CSS, Javascript) funcionam e como elas se integram.

A alternativa A está incorreta, pois a transição para back-end não é necessariamente facilitada pelo aprendizado de front-end sem *frameworks*.

A alternativa B está incorreta, pois o aprendizado sem *frameworks* envolve a compreensão profunda das tecnologias fundamentais (HTML, CSS e JavaScript) que vão além do design e incluem a lógica e a funcionalidade das interfaces.

A alternativa C está incorreta, pois Javascript é uma tecnologia fundamental para o desenvolvimento front-end.

A alternativa D está incorreta, pois aprender sem *frameworks* promove justamente a compreensão mais profunda das tecnologias, afinal, eles abstraem muitos detalhes.

CONFIRA SUAS RESPOSTAS

3. Alternativa E.

Aprender desenvolvimento front-end sem *frameworks* permite aos desenvolvedores entenderem profundamente como as tecnologias fundamentais (HTML, CSS, Javascript) funcionam e como elas se integram.

A alternativa A está incorreta, pois a transição para back-end não é necessariamente facilitada pelo aprendizado de front-end sem *frameworks*.

A alternativa B está incorreta, pois o aprendizado sem *frameworks* envolve a compreensão profunda das tecnologias fundamentais (HTML, CSS e JavaScript) que vão além do design e incluem a lógica e a funcionalidade das interfaces.

A alternativa C está incorreta, pois Javascript é uma tecnologia fundamental para o desenvolvimento front-end.

A alternativa D está incorreta, pois aprender sem *frameworks* promove justamente a compreensão mais profunda das tecnologias, afinal, eles abstraem muitos detalhes.







TEMA DE APRENDIZAGEM 2

CRIANDO PÁGINAS COM HTML E CSS

MINHAS METAS

- Conhecer a linguagem HTML.
- Entender o funcionamento das principais *tags*.
- Criar o primeiro documento estruturado.
- Apresentar, através do CSS, a estilização.
- Aprender as principais propriedades de estilos.
- Entender o modelo de caixas.
- Estilizar uma página HTML.



INICIE SUA JORNADA

No momento que você inicia o desenvolvimento web, abre-se um universo de possibilidades. Imagine, porém, que você tem uma ótima ideia de sistema, desenvolve uma solução web e essa solução fica disponível entre diversas outras. O que diferencia o seu sistema de outros, para ser considerado pelo usuário final?

Diversos são os fatores que podem impactar a escolha e o uso de um software, no entanto a usabilidade e a correta apresentação do conteúdo, geralmente, impactam positiva ou negativamente. Além disso, uma criação bem estruturada permite uma boa apresentação não apenas aos usuários, mas também para outro ator importante no sucesso de uma solução: os indexadores de busca. Afinal, no meio de tanta informação, mais do que nunca precisamos que a nossa informação fique bem indexada e disponível para ser encontrada.

Você pode, então, colocar valor em seu sistema através de uma boa usabilidade, criando uma interface amigável e bonita, para conquistar os usuários. Você também tem a opção de trabalhar em temas específicos, alinhando seu sistema a determinada tendência, ou a um nicho, onde, dentre as demais opções, haja aumento do interesse em seu sistema.

Pensando como usuário, o que lhe atrairia em utilizar um sistema? Ou ainda, quais são as situações que lhe afastariam de um? Essas podem ser questões importantes para definir como você criará e estruturará seu software.



PLAY NO CONHECIMENTO

Para uma boa programação web, é bom lançar bons fundamentos em nosso projeto, para que toda a estrutura se comporte bem, como esperado. Ouça esse episódio do podcast, no qual conversamos sobre esses fundamentos e a importância do bom uso de HTML semântico. **Recursos de mídia disponíveis no conteúdo digital do ambiente virtual de aprendizagem.**

VAMOS RECORDAR?

Desenvolver um sistema web envolve trabalhar com conceitos como a arquitetura cliente-servidor, por isso é bom entender o funcionamento de redes e de tráfego de informações. Você não precisa entender a fundo o funcionamento de redes para trabalhar com programação web, mas, como a parte do tráfego é o maior ponto de lentidão nos sistemas, esse entendimento pode lhe ajudar a projetar sistemas melhores e evitar essa lentidão. Para saber ainda mais como a internet funciona, assista ao vídeo, a seguir: <https://www.youtube.com/watch?v=HNQDoqJoTC4>.

DESENVOLVA SEU POTENCIAL

O desenvolvimento web envolve o desenvolvimento back-end, que é executado no servidor, e o front-end, que é executado no lado do usuário, no navegador. Assim, o navegador web (ou “browser”, do inglês), funciona como um **interpretador**, responsável por transformar tudo o que é descrito nas linguagens **HTML**, **CSS** e **Javascript** em páginas bonitas e dinâmicas para interação com os usuários.

Portanto, em nosso desenvolvimento front-end, seguiremos com essa tríade de linguagens, na qual a linguagem HTML é o responsável pela definição de um documento bem estruturado, a linguagem CSS, pela estilização dos sistemas, e a linguagem Javascript, pela parte dinâmica de um sistema web, proporcionando uma interação mais fluida e requisições assíncronas de informações em outros sistemas.

ESTRUTURANDO PÁGINAS EM HTML

Independentemente do tipo de sistema e de onde será executado, a parte mais importante está nos dados. Nenhum sistema é encomendado ou comprado apenas por ter sido feito em uma linguagem de programação ou em outra, o importante é como ele trata os dados e se resolve os problemas os quais se propõe resolver. Dessa forma, uma exibição correta e bem estruturada de dados pode fazer muita diferença no sucesso dos sistemas.

Para estruturar as informações, a **linguagem HTML** (*HyperText Markup Language* - linguagem de marcação de hipertexto) é uma linguagem de marcação que utiliza *tags* para definir as estruturas. A partir das *tags*, cada elemento é identificado, e o navegador entende como deve apresentar cada parte da informação.

Estrutura básica de um documento

Um **documento HTML** (também conhecido como **página web**) é um documento de **hipertexto**, ou seja, um formato de texto formatado, com possibilidade de uso de imagens, vídeos e diversos outros elementos, principalmente os **links**, que conectam páginas às outras (MDN Contributors, 2023, on-line).

Para entender a estrutura básica e as principais *tags*, a partir de um código HTML simples, descreveremos o que faz cada *tag*, em seguida ,veremos o resultado no navegador.

```
1  <!DOCTYPE html>
2  <html lang="pt-br">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-
width, initial-scale=1.0">
6      <title>Página de Exemplo</title>
7  </head>
8  <body>
9      <h1>Minha primeira página</h1>
10     <p>Exemplo de parágrafo.</p>
11     <p>Um outro parágrafo.</p>
12     <h2>Subtítulo</h2>
13     <p>Mais outro parágrafo.</p>
14 </body>
15 </html>
```

Todo documento HTML é dividido em duas partes: **head** (cabeçalho) e **body** (corpo). Podemos ver a parte do cabeçalho iniciando na linha 3 e encerrando, ou “fechando” na linha 7, no documento apresentado. Já o corpo da página inicia na linha 8 e fecha-se na linha 14. Além disso, todo o conteúdo do documento fica entre as tags `<html>` e `</html>`, das linhas 2 e 15, respectivamente.

Antes de prosseguir, entenderemos melhor o funcionamento desses elementos tão importantes. Uma **tag** (“etiqueta”, em português) é uma marcação para indicar a que o conteúdo se refere. As **tags** podem vir em pares, nas quais há uma **tag** de abertura e uma de fechamento, como vimos em `<html>` e `</html>`, ou podem ser **tags** que não necessitam de fechamento, como a **tag** `<meta>`, que não vem em par.

É possível especificar valores para **propriedades** (ou **atributos**) em **tags**, como vemos na linha 2, que contém: `<html lang="pt-br">`. A **tag** “html” possui uma propriedade “lang”, que possui um valor “pt-br”, o que indica para o navegador que o nosso documento possui um idioma definido como “português brasileiro”, para fins de exibição e de indexação.

Na área de cabeçalho do documento (**tag head**), definimos algumas informações que não ficarão visíveis para o usuário, são informações para os navegadores sobre o próprio documento. Existem as metatags, que permitem indicar configurações do site ao navegador, como a codificação de caracteres na linha 4, ou configurações como o comportamento da renderização da tela, na linha 5. Também no cabeçalho do documento, usamos a **tag title**, onde colocamos o título do site, o qual aparecerá na barra de título do navegador e na aba do site.

Toda a parte do cabeçalho fica meio escondida para o usuário, então, agora, entenderemos o corpo (**body**), que é onde organizaremos as informações visíveis.

Na linha 9, utilizamos a **tag h1**, uma **tag** de título, para indicar que é o título da página ou do artigo onde estiver colocada. Na linha 12, usamos a **tag h2**, para indicar um título num nível mais baixo, ou subtítulo. Em HTML, existem 6 níveis diferentes de títulos: h1, h2, h3, h4, h5 e h6.

Nas linhas 10, 11 e 13 usamos a **tag p**, que é a **tag** de parágrafo, onde colocamos o texto comum de nossa página. Dentro da **tag p**, podemos ainda destacar trechos do texto com ``, que é um texto “mais forte”, equivalente ao “negrito”, ou `` para “enfatizar” algo, equivalente ao “íntilico”.



HTML semântico

Não são apenas elementos visuais que compõem nossos sistemas. Ao desenvolver, precisamos ter em mente o seguinte: o nosso sistema também será acessado por pessoas que precisarão de leitores de telas e, também, por “robozinhos” de indexação, através dos mecanismos de buscas, como Google, Bing e demais outros.

Pensando nisso, quanto mais bem organizadas as informações, melhor consumida será a nossa página e maior será a taxa de sucesso. A versão 5 da linguagem HTML, ou, simplesmente, HTML5, introduziu diversas *tags* que ficaram conhecidas como “tags semânticas”. Elas possibilitam organizar os nossos conteúdos de forma que a *tag* já indica o que quer dizer aquela parte.

Antes de conhecer essas *tags*, entenderemos o funcionamento visual de *tags* em geral. Para começar, veremos que os elementos HTML podem ter comportamento “em bloco” ou “em linha”.



Elementos bloco ocupam todo o espaço horizontal disponível e iniciam uma nova linha no documento. Novos elementos irão começar na próxima linha livre [...] Elementos em linha ocupam apenas o espaço necessário e não iniciam uma nova linha. São chamados elementos em linha justamente por aparecer na mesma linha que outros elementos, caso seja possível (Elementos [...], c2024, on-line).

Assim, temos uma *tag* de bloco genérica, que é a *tag* **<div>**, muito utilizada para ajustar os elementos e um site. Também há uma *tag* em linha genérica, que é a *tag* ****, a qual serve para marcar trechos de texto e, dessa forma, aplicar algum estilo ou comportamento. Essas *tags* não trazem nenhum significado intrínseco, utilizar *tags* semânticas pode trazer o mesmo efeito visual, porém estamos interessados na semântica envolvida.

Então, conhiceremos e utilizaremos as *tags* certas para as coisas certas. Com o intuito de rápida compreensão, usaremos um exemplo um pouco comum de organização de uma página web, como o que aparece na Figura 1.



Figura 1 - Estrutura básica de uma página web / Fonte: o autor.

Descrição da Imagem: a figura ilustra o layout de uma página web semântica, por meio de elementos HTML5. Ela apresenta um cabeçalho (header), seguido por uma barra de navegação (nav), uma área principal de conteúdo (main) com dois artigos (article), uma barra lateral (aside) à direita e, por fim, um rodapé (footer). Esse esquema organiza o conteúdo de forma clara e acessível. Fim da descrição.

É possível variar as disposições dos elementos ou a presença deles, porém, geralmente, uma página web tem um cabeçalho, um menu de navegação, uma área principal, um rodapé, artigos e alguns outros elementos comuns.

Tais divisões do site podem ser feitas por blocos genéricos, como as tags “div”, mas, ao utilizar as tags indicadas, direcionamos o interesse dos leitores automatizados para os lugares corretos. As tags semânticas principais, adicionadas pelo HTML5, são:

<ARTICLE>

Define um conteúdo independente, autocontido.

<ASIDE>

Define um conteúdo “à parte”, um conteúdo anexo ao conteúdo principal.

<DETAILS>

Define detalhes adicionais, que podem ser exibidos ou ocultados.

<FIGCAPTION>

Define uma legenda para o elemento <figure>.

<FIGURE>

Especifica um conteúdo visual autocontido, como ilustrações, diagramas, fotos etc.

<FOOTER>

Define um rodapé para o documento ou para uma seção.

<HEADER>

Define um cabeçalho visual para o documento ou para uma seção.

<MAIN>

Define o conteúdo principal de um documento.

<MARK>

Define texto destacado.

<NAV>

Define links de navegação.

<SECTION>

Define uma seção em um documento.

<SUMMARY>

Define um título visível para um elemento <details>.

<TIME>

Define uma data ou um horário.

Assim, organizamos as informações semanticamente, independentemente de como elas serão exibidas no site. Decisões sobre como exibir, se o menu será horizontal ou vertical, se haverá uma, duas ou três colunas no site etc. serão realizadas na estilização, ou seja, deixaremos a cargo do CSS, que veremos, a seguir.

ESTILIZAÇÃO DE PÁGINAS COM CSS

Depois de estruturar o nosso documento, precisamos deixá-lo apresentável, com boas leitura e usabilidade. Para isso, utilizaremos a linguagem de estilização **CSS** (*Cascading Styles Sheet* - folha de estilos em cascata).

Para começar, precisamos saber que as definições de estilização em CSS sempre acontecem na forma de pares, com “propriedade” e “valor”. Existem muitas propriedades disponíveis, por isso não espere decorar todas elas. Desenvolvemos consultando as referências da linguagem para descobrir ou relembrar as propriedades. Por exemplo, podemos definir que um texto tenha suas letras na cor vermelha a partir da seguinte definição:

```
color: red
```

Simples assim. Mas onde escrevemos essas definições?

Três formas de definir estilos

É possível escrever as definições de estilos CSS em três diferentes formas:

EM LINHA

As definições de estilo são escritas dentro do atributo "style" da *tag* onde o formato será aplicado. Não é recomendado o uso, pois dificulta o reuso e a manutenção.

NO CABEÇALHO

Dentro da *tag* <head>, podemos abrir uma *tag* <style> e escrever nossas definições de estilização dentro dela. Possui a vantagem de unificar as declarações, facilitando a leitura e a manutenção do código.

EM ARQUIVO EXTERNO

É a forma preferencial, através da criação de um arquivo com extensão .css que contém todas as definições de estilização. Também traz a vantagem de agrupar as definições para facilitar a manutenção, com a vantagem extra de facilitar muito o reuso por arquivos diferentes, funcionando como uma biblioteca de estilização.

Na primeira forma (definição **em linha**), definimos a estilização dentro da *tag*. Veja o código, a seguir.

```
<p style="color: red; font-size: 12px">Primeiro parágrafo</p>
<p>Segundo parágrafo</p>
<p>Terceiro parágrafo</p>
```

No código, o primeiro parágrafo traz definições de formatação. As definições são feitas dentro do atributo “style” e podem ser separadas por ponto e vírgula.

A estilização fica limitada ao local onde ela é definida, os demais parágrafos continuam com a formatação padrão. Não é uma forma errada, mas não é incentivada, pois mistura a estilização com o conteúdo, além de dificultar tanto o reuso quanto a manutenção.

As outras formas de definição utilizam o conceito de “seletores” para definir a estilização dos elementos.

Entendendo os seletores CSS

Como a definição de estilos em linha é realizada dentro da *tag*, não há a necessidade de indicar em que lugar essa definição será válida, pois ela funcionará no local onde foi inserida. Agora, para usar definições em outros locais, precisamos uma forma de indicar onde os nossos estilos deverão ser aplicados.

Para isso, CSS utiliza o conceito de **seletores**, que indicam em quais elementos de um documento HTML as definições serão aplicadas. O CSS busca no documento um conjunto de todos os elementos que combinam com o seletor e aplica os estilos a todos esses elementos. Caso nenhum elemento seja encontrado, é retornado um conjunto vazio, não ocasionando nenhum erro.

```
seletor {  
    propriedade1: valor1;  
    propriedade2: valor2;  
}
```

Aqui, observamos a estrutura de uma definição de propriedades CSS, na qual o seletor indica onde será aplicado o conjunto de propriedades definidas entre as chaves. Esse formato possibilita que as definições sejam reaproveitadas em diferentes lugares da página, ou ainda, de outras páginas, evitando o retrabalho (Alves, 2021).

Vejamos um exemplo de como utilizar esse formato de definição, para formatar os parágrafos de um site.

```
p {  
    color: red;  
    font-size: 14px;  
    margin-bottom: 5px;  
}
```

A partir desse exemplo, entenderemos que estamos estilizando todos os elementos os quais usam a tag `<p>` de nosso site, com a cor da fonte vermelha, tamanho 14px, e uma margem de 5px abaixo de cada parágrafo.

Quando precisamos definir uma propriedade relativa a um tamanho, seja de um elemento, seja de um espaçamento, utilizamos, junto a um valor, uma **unidade de medida**, que pode ser uma unidade de **tamanho fixo** (px, pt, cm, mm, in) ou de **tamanho relativo** (% , em, rem, vh, vw). As unidades de tamanho relativo são mais recomendadas, por se ajustarem mais facilmente a diferentes tamanhos de telas, mas há possibilidade de querermos alguns tamanhos fixos para elementos, dependendo de nosso layout.

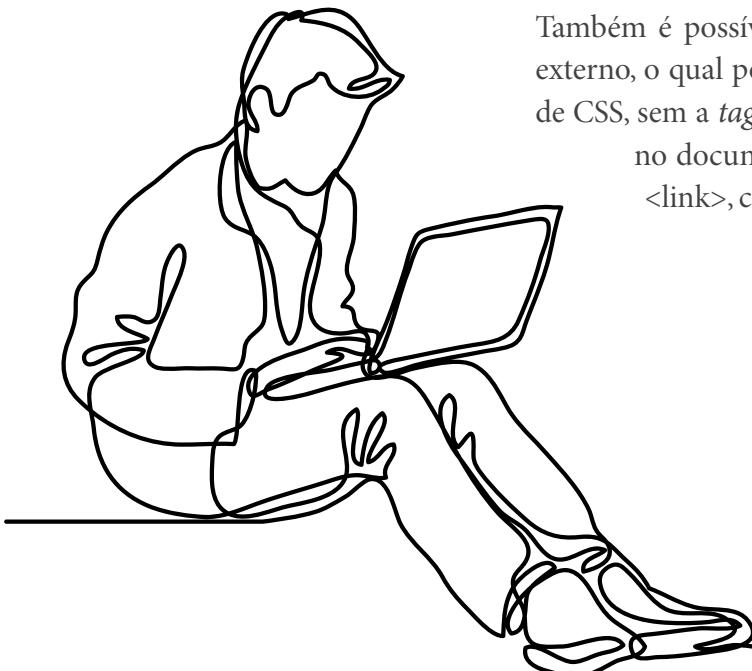


A unidade px é a unidade mágica das CSS. Ela não é relacionada ao tamanho da fonte atual nem a unidades absolutas. A unidade px é definida para ser pequena, mas visível, e tal como uma linha de 1px de largura, pode ser apresentada com 'serrilhamento' (sem anti-aliasing). O quanto nítida, pequena e visível será, depende do dispositivo e da maneira como ele está sendo usado: você o está segurando perto de seus olhos, como em um celular, no comprimento dos braços, como num monitor de computador ou alguma coisa entre um e outro, como se fosse um livro? O px, portanto, não é definido como uma medida constante, mas algo que depende do tipo de dispositivo e o uso típico deste (BOS, 2021, on-line).

Portanto, para definir as propriedades por meio de seletores, como vimos anteriormente, podemos colocar essas definições dentro de uma tag `<style>` no cabeçalho, como no exemplo, a seguir.

```
<!doctype html>
<html>
<head>
<title>Página de exemplo</title>
<style>
p {
    color: red;
    font-size: 14px;
    margin-bottom: 5px;
}
</style>
</head>
<body>
...
</body>
</html>
```

Também é possível colocar em um arquivo externo, o qual possuirá apenas as definições de CSS, sem a tag `<style>`, e será referenciado no documento HTML através da tag `<link>`, como no exemplo, a seguir.



```
<!doctype html>
<html>
<head>
<title>Página de exemplo</title>
<link rel="stylesheet" href="estilos.css">
</head>
<body>
...
</body>
</html>
```

Em ambos os casos, a definição das propriedades será igual, mas o último exemplo, colocar o CSS num arquivo externo, possibilita reaproveitar a estilização em qualquer documento HTML que utilizar essa mesma tag `<link>` apontando para nosso arquivo CSS.

Diferentes tipos de seletores

São muitas as propriedades existentes no CSS, por isso não nos prenderemos muito a elas neste material. Serão descobertas propriedades novas pelo caminho, mas recomendo que você sempre acompanhe uma referência da linguagem, como as referências da **Mozilla Developer Network** (developer.mozilla.org) ou a **W3Schools** ([w3schools.com](https://www.w3schools.com/css/)).

Antes, precisamos conversar sobre as diferentes possibilidades de seletores que temos. Não podemos apenas selecionar pelas *tags*, mas fazer várias combinações de seletores e até mesmo usar pseudoseletores.

Além das *tags*, temos a opção de selecionar os elementos por suas classes. Em HTML, as **classes** são definidas através do atributo **class**, que recebe um nome para identificar um conjunto de elementos. As classes podem ser definidas para qualquer elemento HTML, como vemos na tag, a seguir.

```
<a href="index.html" class="link-especial">Home</a>
```

Nesse exemplo, nosso link recebe uma classe com o nome “link-especial” (você pode escolher qualquer nome para classe), o que significa que os links os quais têm essa classe receberão uma formatação especial.

Na definição do CSS, as classes são identificadas por um ponto (.) no início do nome.

```
.link-especial {  
    color: gold;  
}
```

Por essa definição, todos os elementos que possuírem a classe “link-especial” terão seu texto escrito em letras douradas (“gold”). Uma classe pode ter diversos elementos, assim como um elemento pode ter várias classes. No HTML, definimos mais de uma classe para um elemento, por meio do uso de espaço.

```
<p class="importante citacao">Olá, mundo!</p>
```

Nesse exemplo, o parágrafo possui, ao mesmo tempo, as classes “importante” e “citacao”, tendo, então, as propriedades de ambas as classes aplicadas ao parágrafo.

Também podemos colocar um “id” (identificador) em um elemento, ou seja, um nome que o identifica de forma única em um elemento. Não se deve usar o mesmo id para diferentes elementos.

```
<p id="paragrafo1">Eu sou um parágrafo</p>
```

Com um id, podemos formatar diretamente o elemento, utilizando seu id no identificador, por meio de um símbolo de sustenido (#) antes do nome.

```
#paragrafo1 {
    color: blue;
}
```

Assim, definimos que aquele parágrafo, especificamente, terá a cor de fonte azul, porém ainda há muito mais poder nos seletores do que parece. Podemos fazer várias combinações de seletores para controlar bem o que selecionar.

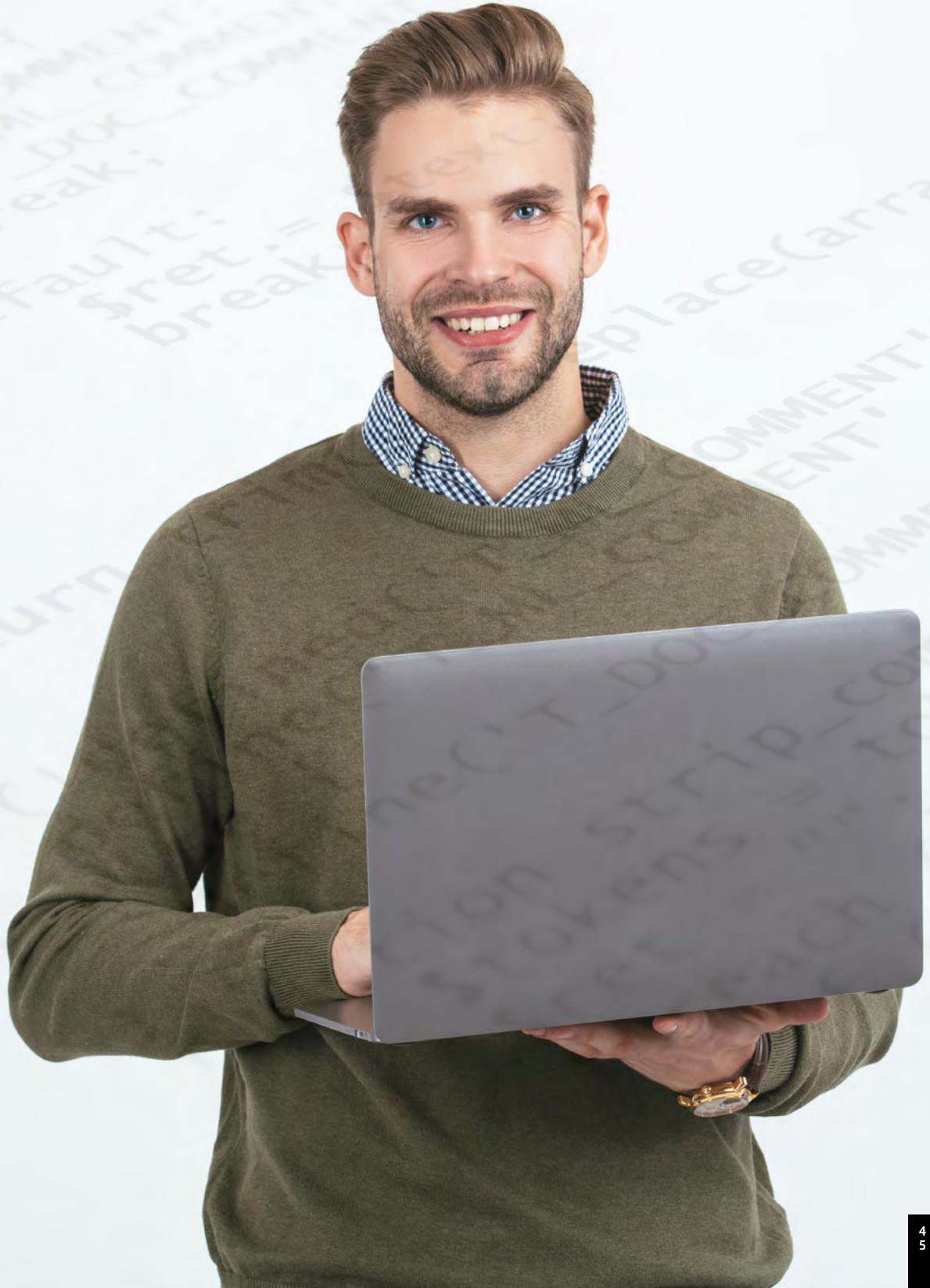
Lembrando que CSS significa “folhas de estilo em cascata”. O que isso significa?

Vimos que o HTML cria uma hierarquia de elementos, onde uma tag pode estar definida dentro de outra, o que chamamos de “descendente” ou “elemento filho” (ou algumas variações disso). Aproveitando essa hierarquia, diversas propriedades são passadas diretamente (ou “herdadas”) pelos elementos descendentes, como acontece com a propriedade “color”, mas existem outras propriedades que não são herdadas, como a propriedade “border”.

Além disso, é possível definir nossos seletores em termos dessa hierarquia. Por exemplo, se quisermos definir uma estilização para todos os elementos que possuem a classe “destaque” dentro de parágrafos que estiverem dentro de artigos, faremos algo como o código, a seguir.

```
article p .destaque {
    background-color: yellow;
}
```

Os espaços na definição do seletor, na primeira linha, definem a ideia de hierarquia. Sugiro que você teste isso em sua máquina, ou em algum sistema on-line, para explorar esse comportamento que ajuda muito no desenvolvimento.





EU INDICO

Existem diversas possibilidades de uso de seletores combinados, incluindo a possibilidade de selecionar elementos pelo conteúdo, por atributos das tags, por posicionamento, entre outras. Como esse seria um conteúdo bem amplo para este material, veja uma ótima tabela com (quase) todas as possibilidades apresentadas pelo W3 Schools: https://www.w3schools.com/cssref/css_selectors.php.

Assim como HTML, CSS é uma linguagem que vem evoluindo bastante com o tempo, pois novas propriedades e possibilidades são adicionadas frequentemente. Deve-se conhecer bem as referências da linguagem que costumam mostrar quais navegadores dão suportes para determinada propriedade e como funcionam.

Precedência de definições CSS

Para definir as propriedades por meio de seletores em CSS, um grupo de elementos é selecionado e sobre ele é aplicado um conjunto de propriedades. Porém, o que acontece caso um elemento esteja selecionado por dois conjuntos diferentes de seletores, com regras conflitantes?

Isso é extremamente comum de acontecer. Imagine que você defina que todos os elementos parágrafos (`<p>`) terão como cor da fonte o cinza escuro, mas você define que a classe “destaque” terá, como cor da fonte, o vermelho escuro. Um dos parágrafos foi definido como segue:

```
<p class="destaque">Eu sou um parágrafo</p>
```

Esse parágrafo aparecerá como cinza ou como vermelho? Você pode se perguntar: “qual regra foi definida primeiro?” Já adianto que, nesse caso, isso não faz diferença.

A linguagem CSS define a **precedência de estilização**, ou seja, quais regras possuem prioridade, baseado em um conceito chamado **especificidade**. Quanto mais específica for a definição, maior a precedência. Além da especificidade, o CSS leva em conta, também, a herança e o efeito cascata (Souza, 2017, on-line).

Assim, quando definimos a estilização pelo id, este é mais específico do que as classes, e as classes são mais específicas do que os elementos. Em nosso exemplo, o parágrafo seria vermelho porque foi definido de forma mais específica. Em caso de duas definições de mesma especificidade, aí sim faria diferença qual foi escrita primeiro, porque seria sobreescrita pela última definição.

Em partes, o CSS funciona bastante por tentativas e erros. Se tiver dúvidas de algo, teste. Vale a pena.

 **EM FOCO**

Assista à videoaula para mais informações e conhecimentos sobre este tema de aprendizagem. Recursos de mídia disponíveis no conteúdo digital do ambiente virtual de aprendizagem.

NOVOS DESAFIOS

O começo dos estudos de desenvolvimento web pode parecer um pouco lento e detalhado, mas é importante para a sua carreira conhecer bem os fundamentos do desenvolvimento, sabendo estruturar os documentos e estilizar os elementos.

Um sistema bem desenvolvido, bem estruturado e com usabilidade bem trabalhada em sua apresentação pode fazer muita diferença no sucesso das aplicações. Portanto, dedique certo tempo a aprender e a testar as diferentes propriedades de CSS para os elementos.

No decorrer da carreira de desenvolvimento front-end, você utilizará vários *frameworks* e bibliotecas que fazem boa parte do serviço, porém um profissional que sabe resolver detalhes os quais o *framework* não cobre ou faz errado, podem ser um diferencial em sua carreira.

Além disso, o aprendizado de uma ferramenta nova sempre é mais tranquilo quando você entende o que está funcionando nos bastidores. Treine o seu conhecimento, coloque em prática e continue sempre se atualizando para ter uma carreira bem completa.

VAMOS PRATICAR

1. O que o HTML5 trouxe de novo são as novas possibilidades para que o Javascript controle os elementos criados no código. O HTML5 será o novo padrão da linguagem de marcação de hipertextos (HTML) (Bonatti, 2014).

Qual dos seguintes elementos HTML é considerado semântico?

- a) .
- b) <i>.
- c) <div>.
- d) <main>.
- e) .

2. "HTML semântico refere-se ao uso de *tags* HTML para reforçar o significado do conteúdo que elas contêm, ao invés de apenas definir sua aparência. Utilizar HTML semântico ajuda não apenas os desenvolvedores a entender a estrutura e o conteúdo de uma página, mas também os motores de busca e tecnologias assistivas, como leitores de tela, a interpretar o conteúdo da web de forma mais eficaz" (HTML semântico [...], 2024, on-line).

Considerando o HTML semântico, é correto dizer que:

- a) A tag é um elemento semântico que deve ser usado para trechos de texto com significado especial.
- b) A tag é um elemento semântico que define texto em negrito para indicar ênfase.
- c) A tag <div> tem um significado semântico claro, sendo ideal para estruturar o conteúdo.
- d) A tag <header> é usada para definir cabeçalhos de página ou de seções.
- e) A tag <link> é utilizada para indicar um link clicável que leva a navegação para outra página.

VAMOS PRATICAR

3. O uso desses elementos semânticos tem finalidade semântica, ou seja, serve para identificar os conteúdos, e não para efeitos de construção de layout. Softwares específicos, por exemplo, leitores de tela ou outros agentes de usuário, poderão acessar, com facilidade, as seções específicas do documento definidas pelos elementos semânticos se os conteúdos estiverem bem definidos.

Ao criar a estrutura semântica de uma página, qual dos seguintes conjuntos de elementos está organizado corretamente?

- a) <header>, <nav>, <section>, <footer>.
- b) <header>, <footer>, <main>, <section>.
- c) <nav>, <header>, <section>, <main>.
- d) <section>, <header>, <footer>, <main>.
- e) <footer>, <header>, <main>, <section>.

REFERÊNCIAS

- ALVES, W. P. **HTML & CSS**: Aprenda como construir páginas web. Rio de Janeiro: Grupo GEN, 2021. *E-book*. Disponível em: <https://integrada.minhabiblioteca.com.br/#/books/9786558110187/>. Acesso em: 8 out. 2024.
- BONATTI, D. **Desenvolvimento de jogos em HTML5**. Rio de Janeiro: Brasport, 2014. *E-book*. Disponível em: <https://plataforma.bvirtual.com.br>. Acesso em: 9 out. 2024.
- BOS, B. Folhas de estilo web. Dicas & truques CSS. **W3C**, *ls. lJ*, 6 jan. 2021. Disponível em: https://www.w3.org/Style/Examples/007/units.pt_BR.html. Acesso em: 8 out. 2024.
- ELEMENTOS Bloco e Em Linha - HTML e CSS na Prática. **Guilherme Müller**, *ls. lJ*, c2024. Disponível em: <https://guilhermemuller.com.br/ead/html-css-na-pratica/elementos-bloco-em-linha>. Acesso em: 8 out. 2024.
- HTML SEMÂNTICO: Entendendo e utilizando para estruturas Web mais significativas. **Sujeito Programador**, *ls. lJ*, 29 mar. 2024. Disponível em: <https://sujeitoprogramador.com/html-seman-tico/>. Acesso em: 9 out. 2024.
- MDN CONTRIBUTORS. HTML: Linguagem de Marcação de Hipertexto. **MDN Web Docs**, *ls. lJ*, 1 out. 2023. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTML>. Acesso em: 8 out. 2024.
- SOUZA, E. G. de. Entendendo a precedência de estilo em CSS: Especificidade, Herança e Efeito Cascata. **Medium**, *ls. lJ*, 15 fev. 2017. Disponível em: <https://medium.com/emanuelg-blog/entendendo-a-preced%C3%A3ncia-de-estilo-em-css-especificidade-heran%C3%A7a-e-efeito-cascata-a437c4929173>. Acesso em: 8 out. 2024.
- TERUEL, E. C. **HTML 5**: Guia prático. Rio de Janeiro: Grupo GEN, 2013. *E-book*. Disponível em: <https://integrada.minhabiblioteca.com.br/#/books/9788536519296/>. Acesso em: 9 out. 2024.

CONFIRA SUAS RESPOSTAS

1. Alternativa D.

O elemento `<main>` é semântico, pois define o conteúdo principal do documento, sendo relevante para SEO e acessibilidade.

A alternativa A está incorreta, pois o elemento `` não traz conteúdo semântico, por isso ficou defasado e não deve ser usado.

A alternativa B está incorreta, pois o elemento `<i>` não traz conteúdo semântico, por isso ficou defasado e não deve ser usado.

A alternativa C está incorreta, pois o elemento `<div>` é um bloco genérico, sem conteúdo semântico.

A alternativa E está incorreta, pois o elemento `` é um elemento genérico, sem conteúdo semântico.

2. Alternativa D.

A tag `<header>` é usada para definir cabeçalhos de página ou de seções.

A alternativa A está incorreta, pois a tag `` é genérica, não semântica.

A alternativa B está incorreta, pois a tag `` está em desuso, substituída pela tag ``, que tem melhor significado.

A alternativa C está incorreta, pois a tag `<div>` é genérica, não semântica.

A alternativa E está incorreta, pois a tag `<a>` é quem faz essa função.

3. Alternativa A.

A sequência correta organiza os elementos semânticos de uma maneira que reflete a estrutura típica de um documento HTML, começando com cabeçalho, seguido pela navegação, conteúdo e, finalmente, rodapé.

A alternativa B está incorreta, pois o elemento `<footer>` deve vir ao final.

A alternativa C está incorreta, pois o elemento `<header>` deveria vir antes do `<nav>`.

A alternativa D está incorreta, pois o elemento `<main>` está posicionado após o rodapé.

A alternativa E está incorreta, pois o elemento `<footer>` não deve vir no início.



TEMA DE APRENDIZAGEM 3

APROFUNDANDO O LAYOUT

MINHAS METAS

- Entender o box model.
- Conhecer o flexbox.
- Entender os eixos de contêineres.
- Definir o alinhamento interno do contêiner.
- Personalizar o alinhamento de itens.
- Controlar a flexibilidade de itens.
- Aprender a criar layout responsivo.

INICIE SUA JORNADA

Certa vez, um desenvolvedor criou um sistema inovador que ressolveria o problema de diversas pessoas. Então, ele criou, publicou, divulgou etc., mas não teve muito sucesso. Meses depois, um concorrente entrou no mercado, arrebatou os corações e virou sucesso absoluto em sua área.

Essa é uma história genérica, mas inspirada em diversas histórias reais. Às vezes, um software é bom (ou suficiente) para resolver um problema, mas não possui apelo visual ou boa usabilidade e perde o mercado para alguma solução que pode até ser pior, porém agrada mais, além de ser mais prazerosa de usar.

A criação do design de um sistema não pode ficar apenas no básico, não dá para apenas “resolver o problema”, até porque a usabilidade também é parte do problema. Quer saber se o seu sistema está com boa usabilidade? O primeiro passo é passá-lo a alguém que não é da área de desenvolvimento ou da área de domínio do software, para testar. Entregue para sua mãe, sua tia, alguém que não tenha participado da criação, e veja se o sistema está intuitivo, se está fácil de ser utilizado.

Você pode aproveitar a sua própria experiência, também. Lembra de algum programa que já baixou, utilizou muito pouco e o desinstalou, por que era muito ruim de usar? O que lhe incomodou naquele sistema? O que você pode evitar para que não façam o mesmo com o seu?

Aprender o básico para estruturar o layout é importante. Agora, iremos um pouquinho além, com um olhar mais apurado aos detalhes. Veremos como controlar o layout para desenhar na tela o que está em nossa imaginação.



PLAY NO CONHECIMENTO

Quer saber como isso tudo se aplica à “vida real”? Ouça esse episódio do podcast, no qual há um pouco mais desse conteúdo, com exemplos de como as empresas estão utilizando esses conceitos em seus sistemas. **Recursos de mídia disponíveis no conteúdo digital do ambiente virtual de aprendizagem.**

VAMOS RECORDAR?

Conhecer a base de HTML e CSS é importante para criar layouts mais aprofundados. Caso você precise aprender ou relembrar o funcionamento de CSS, seus seletores e propriedades, assista ao vídeo, a seguir: <https://www.youtube.com/watch?v=TyqWhaSntcU>.

DESENVOLVA SEU POTENCIAL

Desenvolver software é uma arte que envolve muito raciocínio lógico, mas também muita criatividade. Criamos soluções para diferentes problemas e precisamos desenvolver soluções criativas para fazer com que o computador, que é algo puramente lógico, exato e repetitivo, entenda problemas complexos de pessoas que, por sua vez, não são tão exatas, muitas vezes, elas aproximam-se mais do caos.

Agora, exploraremos um pouco a nossa criatividade visual. Como podemos apresentar os sistemas de uma forma mais bonita, mais elegante? No dia a dia do desenvolvimento, faz muita diferença trabalhar com uma equipe de design que já tenha experiência em montar interfaces de boa usabilidade. Se for esse o caso, mesmo assim, você precisa entender como transformar aquele design em código e como comportar tudo em seu sistema.

ENTENDENDO O MODELO DE CAIXA (BOX MODEL)

Para criar um sistema web, começamos pela escrita de nosso documento em HTML, em seguida, passamos às definições de estilos em CSS. Para isso, entendemos os conceitos de *tags*, atributos, seletores, classes, propriedades etc. Agora, faremos em propriedades importantes para entender o comportamento visual de nossos elementos.

Todo elemento em um documento possui definições de propriedades conhecidas como **modelo de caixa**, comumente chamadas pelo nome em inglês, box model. Esse modelo define como deve ser calculado e apresentado o elemento.

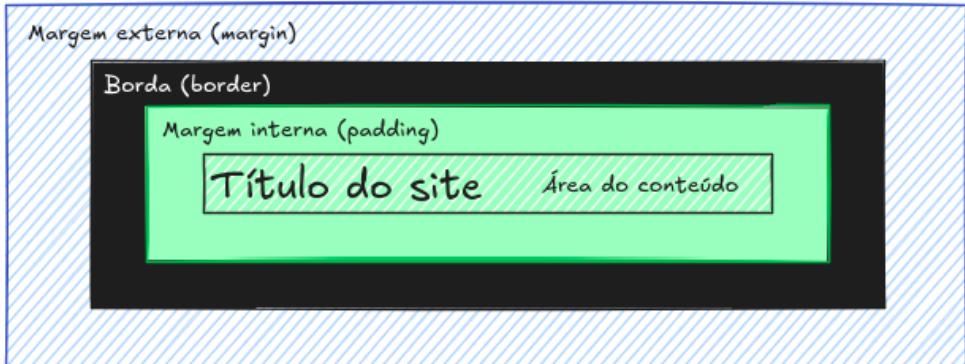


Figura 1 - O modelo de caixa (box model) de um elemento HTML / Fonte: o autor.

Descrição da Imagem: diagrama colorido que ilustra a estrutura de layout de uma página da web, com destaque de diferentes camadas de formatação. No centro, há uma área nomeada "Área do conteúdo" com o texto "Título do site" em destaque, circundada por uma área chamada "Margem interna (padding)" em verde claro. Envolvendo essa seção, há uma borda denominada "Borda (border)" em verde escuro e, finalmente, uma área chamada "Margem externa (margin)", indicada por um padrão quadriculado cinza claro. Esse diagrama serve para explicar visualmente a separação e função das margens internas e externas, bordas e área de conteúdo no design web. Fim da descrição.

Podemos observar, na Figura 1, um elemento qualquer HTML. Por padrão, os elementos possuem um formato retangular (uma caixa) e quatro camadas: a área do conteúdo, a **margem interna** (padding), a **borda** e a **margem externa** (MDN Contributors, 2023, on-line).

Cada uma dessas partes possui seu próprio tamanho, o que impacta no tamanho final de um elemento. Por exemplo, se tivermos um elemento como uma *div*, com largura de 300px, com uma borda de 2px, margem interna de 10px e margem externa de 20px, lembrando que as margens e as bordas circulam o elemento, calculamos o espaço necessário para essa *div* da seguinte forma:

$$\text{Espaço horizontal necessário} = 300 \text{ (largura)} + 4 \text{ (bordas)} + 20 \text{ (paddings)} + 40 \text{ (margens)} = 364\text{px}$$

O elemento em si terá 324px (conteúdo + margem interna + borda) e ainda precisará de mais 40px de margens.

Para ter mais controle do dimensionamento de nossos elementos, podemos definir se a largura que definimos na propriedade “width” já contabilizará ou não a margem interna e a borda. Para isso, definiremos a propriedade “**box-sizing**”, do CSS, que tem duas formas possíveis: “**content-box**”, essa forma padrão que vimos, somando todas as dimensões, ou “**border-box**”, a qual fixa a largura do documento, diminuindo a área do conteúdo, caso haja margem interna ou borda.

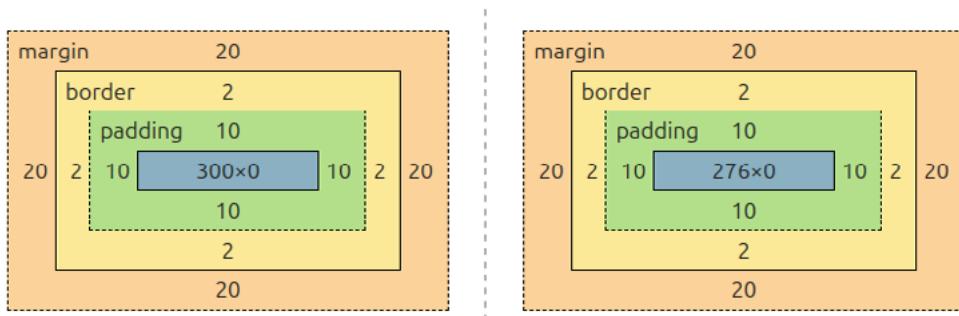


Figura 2 - Modelo de caixa de nosso exemplo, utilizando box-sizing como content-box à esquerda, e como border-box à direita / Fonte: o autor.

Descrição da Imagem: duas representações do modelo de caixa (box model) em CSS, comparando dois elementos com diferentes configurações de largura. Em ambas as caixas, são destacados os espaços de margem, borda e preenchimento. À esquerda, a largura total do conteúdo é de 300 pixels, enquanto à direita, a largura do conteúdo é de 276 pixels. Cada caixa inclui uma margem de 20 pixels, uma borda de 2 pixels, e um preenchimento (padding) de 10 pixels ao redor do conteúdo. A imagem ilustra visualmente como o modelo de caixa afeta o tamanho e o espaçamento de elementos em CSS. Fim da descrição.

Pela ferramenta de inspeção do navegador, podemos observar as propriedades CSS aplicadas a cada elemento. Os termos talvez mudem de navegador a navegador, mas, basicamente, clicamos no elemento desejado com o botão direito e procuramos a opção “Inspecionar” no menu suspenso.

Na Figura 2, podemos observar o modelo de caixa de nosso elemento mencionado, funcionando com as duas formas de “box-sizing”. Para a nossa *div* de exemplo, foi utilizado o seguinte código em CSS:

```
.container {  
    width: 300px;  
    border: 2px solid black;  
    padding: 10px;  
    margin: 20px;  
}
```

Assim, a área ocupada no site muda de acordo com o comportamento do modelo de caixa. Apesar da forma “content-box” ser o comportamento padrão, esperado pelos navegadores, o formado “border-box” pode ser mais previsível, o que faz com que muitos desenvolvedores o adicionem como padrão (Bell, 2021, on-line).

CSS Reset

Para ter um controle maior do layout de um site e do *box model* dos elementos, é preciso ter em mente que todos os elementos têm margens externas, internas, bordas e área de conteúdo, ainda que não sejam declaradas explicitamente. Quando não declaramos uma propriedade, ela receberá o valor padrão, que é gerenciado pelos navegadores. Porém, como os navegadores podem ter uma diferença ou outra de implementação, talvez os tamanhos de margens, fontes e alguns outros detalhes gerem efeitos indesejados para o seu layout, por isso é comum fazer uma definição padrão a todos os elementos.

Ao utilizar seletores no CSS, é possível usar o coringa (*) para selecionar todos os elementos. Uma técnica comum, ao definir estilos, é chamada de **CSS reset**, que consiste em colocar valores iniciais para todos os elementos de uma página, com intuito de sobreescrver as definições utilizadas pelos navegadores.

Em geral, define-se para zero tamanhos de margens internas e externas, mas pode-se definir, também, zero para as bordas, tipos de fontes e comportamento do “box-sizing” para todos os elementos de uma vez. Veremos um exemplo simples na Figura 3.

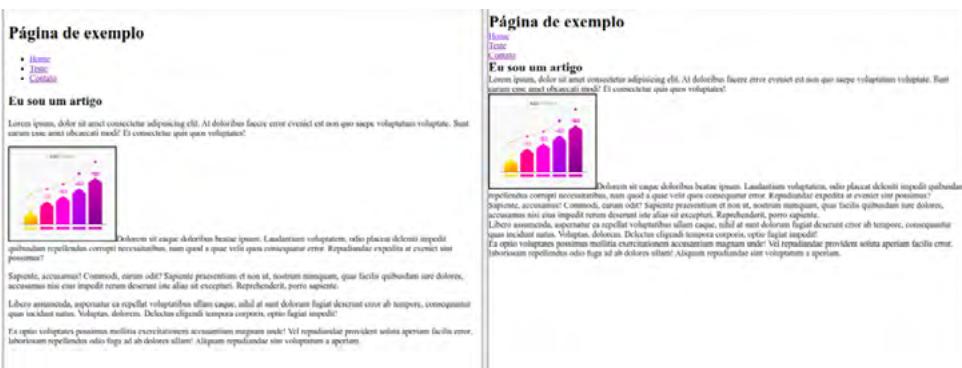


Figura 3 - Um exemplo de página com as definições de margem do navegador à esquerda e com CSS reset à direita / Fonte: o autor.

Descrição da Imagem: duas versões da mesma página web com conteúdo idêntico, lado a lado, para ilustrar os efeitos de um reset de CSS. À esquerda, a página apresenta o estilo padrão dos navegadores, com espaçamento maior ao redor dos elementos e links sublinhados em cor azul e roxa. À direita, a página parece mais compacta, sem margens e espaçamentos padrão, o que sugere a aplicação de um reset de CSS, o que remove estilos iniciais, deixando o layout mais básico e uniformizado. Ambas as páginas têm um título, um menu de links, um artigo com parágrafos de texto e uma imagem. Fim da descrição.

É possível observar, na imagem, como os elementos aparecem mais “coladinhos” no exemplo da direita, onde foi definido o CSS reset. Você pode pensar: “o exemplo da esquerda está mais bonito, qual a vantagem de juntar tudo?”. A vantagem está no controle que você tem, definindo suas próprias margens, bordas e demais definições. No exemplo apresentado, no início do arquivo CSS, foi declarado o seguinte trecho:

```
* {
    margin: 0;
    padding: 0;
}
```

Como dito antes, o asterisco (*) funciona como um “coringa” que define as propriedades para todos os elementos de uma página. Você pode definir outras propriedades, de acordo com o que se adequar melhor ao seu design.

ALINHANDO OS ELEMENTOS COM FLEXBOX



Não é segredo que o pessoal da programação tem certa dificuldade com a construção de layouts. Às vezes, a pessoa resolve problemas muito difíceis na área de lógica ou matemática, mas não se dá bem alinhando um botão na tela.

Definir o posicionamento e o alinhamento de elementos em CSS também era uma tarefa árdua, repleta de gambiarras, antes do surgimento do **flexbox**, um modelo do CSS que permite os componentes serem empilhados vertical e horizontalmente (Peretta, 2019, on-line).

Agora que já entendemos o funcionamento do *box model*, podemos entender melhor a disposição dos elementos em uma página. Por padrão, todos os elementos são exibidos como “bloco” ou “elemento em linha”. Um bloco, por padrão, possui como largura 100% do espaço que estiver disponível, assim, não conseguimos colocar blocos lado a lado sem um pouco de magia (leia-se CSS).

O modelo flexbox introduz um conceito muito importante, que é o **contêiner**, ou container, em inglês, que é o mais comum em literaturas as quais abordam esse modelo). Qualquer elemento pode se comportar como um **flex container**, ou seja, contêiner de flexbox, para isso, basta que o elemento tenha a propriedade CSS “**display: flex**”.

Em linhas gerais, um contêiner é um elemento que engloba **itens**, os quais podem ser qualquer tipo de elemento. Dessa forma, temos propriedades especiais em flexbox para contêineres e para itens.

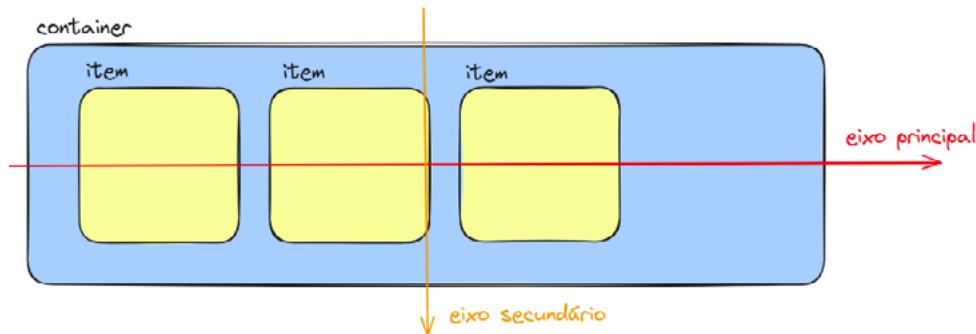


Figura 4 - Um contêiner flexbox com três itens / Fonte: o autor.

Descrição da Imagem: mostra um contêiner flexbox com três itens dispostos horizontalmente dentro dele, ilustrando o conceito de eixos nesse modelo. O eixo principal, destacado em vermelho, vai da esquerda para a direita, indicando a disposição horizontal dos itens. O eixo secundário, destacado em laranja, vai de cima para baixo, representando a direção vertical em que os itens podem ser alinhados. O contêiner é azul, e os itens são quadrados amarelos, demonstrando a organização visual em um layout flexbox.

Algo essencial para entender o funcionamento do flexbox: ele trabalha com dois eixos, o principal e o secundário. Não se usam os termos “vertical” e “horizontal”, porque você pode trocar a direção dos eixos por meio de uma propriedade que veremos, a seguir.

EU INDICO

É importante ressaltar que não faremos uma cobertura extensiva de todas as propriedades do flexbox. No site W3Schools, existe uma área de demonstração interativa onde há uma demonstração de todas as propriedades do CSS, flexbox ou não, cujo entendimento fica facilitado pela visualização das propriedades em ação. Acesse: https://www.w3schools.com/cssref/playdemo.php?filename=playcss_flex-direction

Veremos como funcionam as propriedades flexbox, iniciando pelos contêineres, que fazem uma formatação geral. Depois, passaremos para os itens, onde faremos personalizações mais específicas.

Propriedades para contêineres

Para começar, podemos definir a direção de nosso eixo principal através da propriedade “flex-direction”, definindo uma das quatro opções: “row” (linha, a opção padrão), “column” (coluna), “row-reverse” (linha reversa) e “column-reverse” (coluna reversa). Como uma imagem vale mais do que mil palavras, veja o funcionamento na Figura 5.

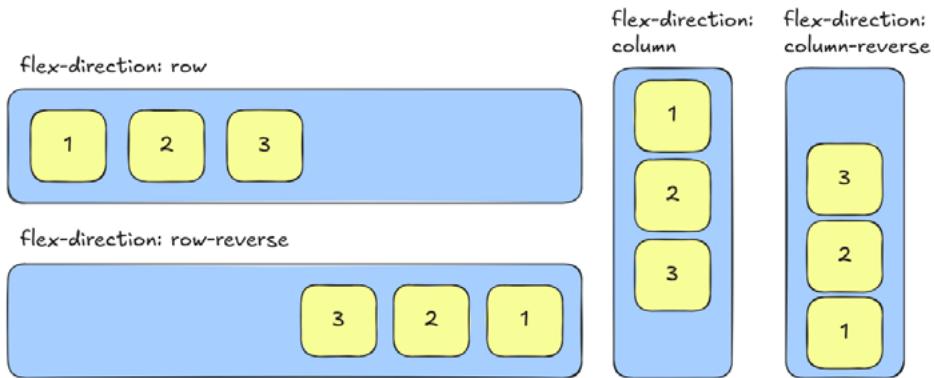


Figura 5 – Definição do eixo vertical do contêiner / Fonte: o autor.

Descrição da Imagem: ilustração de quatro conjuntos de caixas retangulares numeradas de 1 a 3, demonstrando diferentes propriedades de “flex-direction” no CSS, para design de páginas web. O primeiro conjunto, à esquerda, classificado como “flex-direction: row”, mostra as caixas em uma linha horizontal, indicando uma disposição padrão em linha. O segundo conjunto, no canto superior direito, classificado como “flex-direction: column”, exibe as caixas em uma coluna vertical, mostrando uma sequência de cima para baixo. O terceiro conjunto, embaixo do segundo, identificado como “flex-direction: column-reverse”, mostra as caixas empilhadas verticalmente em ordem inversa, de baixo para cima. Embaixo do primeiro conjunto, há outra linha de caixas, identificada como “flex-direction: row-reverse”, onde a sequência de números vai da direita para a esquerda, oposta à ordem padrão. Essa imagem serve como um auxílio visual educacional para entender como diferentes propriedades de flex-direction afetam a orientação do layout no CSS. Fim da descrição.

Assim, o eixo principal pode estar em linha, em coluna, e pode ter a ordem reversa, permitindo mudar toda a ordem dos elementos, sem a necessidade de modificar o documento HTML.

Depois, podemos trabalhar o alinhamento dos itens dentro do contêiner de forma bem direta e simplificada, o que facilita muito a disposição dos elementos em um layout.

Trataremos de duas propriedades importantes aqui: “justify-content”, propriedade que alinha os itens no eixo principal, e “align-items”, que alinha os itens pelo eixo secundário. Note a grafia de “items” com “m”, porque as definições estão em inglês.

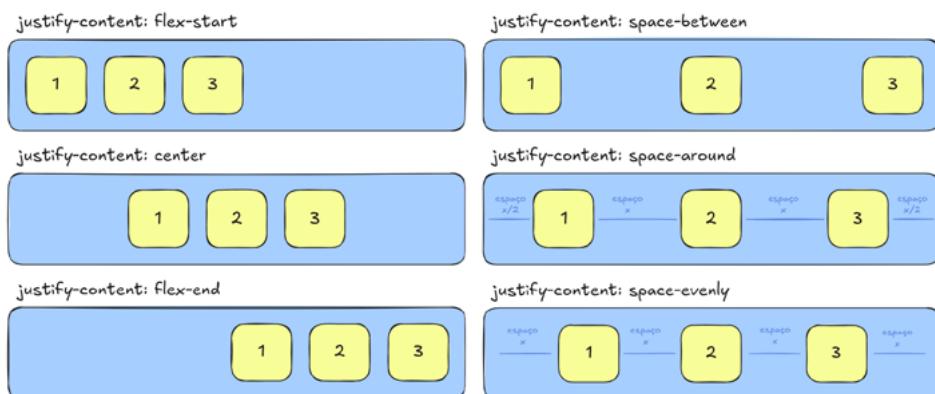


Figura 6 - Alinhamento no eixo principal pela propriedade “justify-content” / Fonte: o autor.

Descrição da Imagem: ilustração de seis painéis, cada um mostrando uma maneira diferente de distribuir três retângulos amarelos, os quais são numerados de 1 a 3, cada um dentro de um contêiner azul. Cada painel representa um valor diferente da propriedade “justify-content” no flexbox do CSS, que é um modo de layout no CSS3. Os valores mostrados são “flex-start”, “space-between”, “center”, “space-around”, “flex-end” e “space-evenly”. Esses valores afetam o alinhamento e o espaçamento dos itens ao longo do eixo principal do contêiner. Fim da descrição.

Podemos alinhar os itens no eixo principal através da propriedade “justify-content”, como visto na Figura 6, a partir dos seguintes valores:

FLEX-START

Alinha os itens no início do contêiner.

CENTER

Alinha os itens no centro do contêiner.

FLEX-END

Alinha os itens no final do contêiner.

SPACE-BETWEEN

Coloca o primeiro elemento no início do contêiner, o último item no final do contêiner, espalha os demais e coloca o mesmo espaçamento entre eles.

SPACE-AROUND

Similar ao "space-between", porém coloca o mesmo espaçamento antes e depois de cada item, inclusive do primeiro e do último.

SPACE-EVENLY

Similar ao "space-around", mas com o mesmo espaçamento antes do primeiro, depois do último e entre os elementos.

Depois, podemos alinhar os itens pelo eixo secundário com a propriedade “align-items”, similar ao alinhamento do eixo principal, porém com uma pequena diferença nas opções, como veremos a seguir.

FLEX-START

Alinha os itens no início do eixo secundário.

CENTER

Alinha os itens no centro do eixo secundário.

FLEX-END

Alinha os itens no final do eixo secundário.

STRETCH

Estica os itens para preencher todo o espaço.

BASELINE

Alinha os itens pela base do conteúdo.

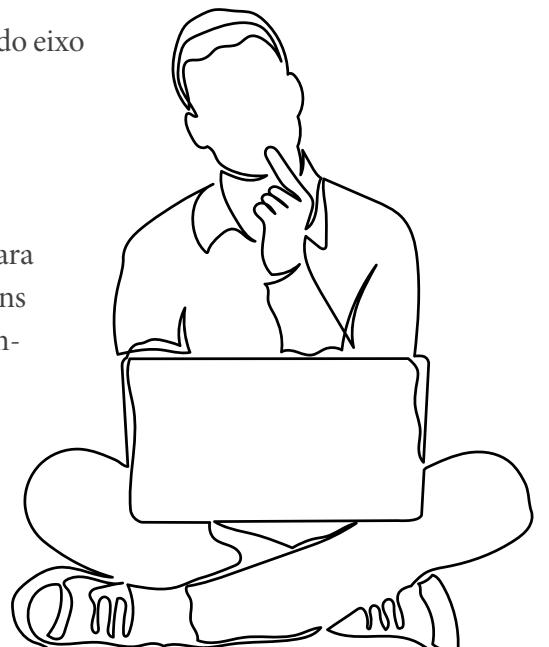
Agora, quando alguém chegar até você dizendo que “dev *não sabe centralizar uma div*”, você pode responder com o seguinte código:

```
div {  
    display: flex;  
    justify-content: center;  
    align-items: center;  
}
```

O código apresentado alinha a div no centro do eixo principal e no centro do eixo secundário.

Propriedades para itens

Enquanto as definições de propriedades para contêineres permitem manipular todos os itens de uma só vez, podemos personalizar o comportamento de itens específicos a partir de propriedades flexbox.



Primeiro, mudaremos o alinhamento de um item específico no eixo secundário com a propriedade “**align-self**”, a qual aceita os mesmos valores que “align-items”. Além disso, é possível mudar a ordem de exibição dos itens a partir da propriedade “**order**” que, a princípio, tem valor zero. Valores menores (podem ser negativos) aparecem primeiro, seguindo a ordem do menor para o maior. Vejamos um exemplo de ambas as propriedades na Figura 7.

```
#item2 {
    order: 5;
    align-self: center;
}

#item4 {
    align-self: flex-end;
}
```

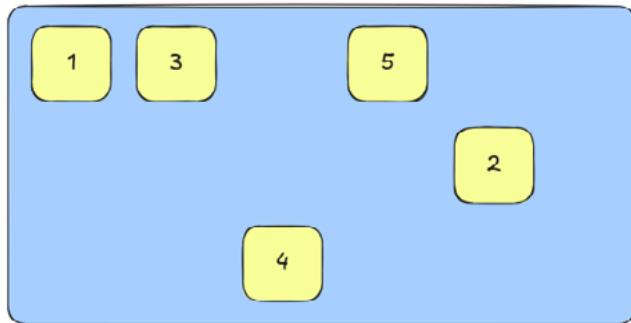


Figura 7 – Alinhamento personalizado de item e mudança de ordem / Fonte: o autor.

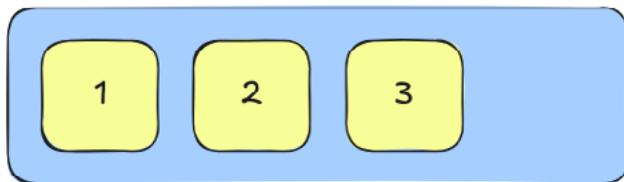
Descrição da Imagem: ilustração de um retângulo azul com cinco quadrados amarelos dentro, cada um numerado de 1 a 5. Há, também, um texto no lado esquerdo da imagem que parece ser um código CSS relacionado ao estilo de páginas web. O código especifica estilos para “#item2” e “#item4” com as propriedades “align-self: center;” e “align-self: flex-end;”, respectivamente. Essa imagem é relevante ao desenvolvimento web, especialmente para entender como o CSS flexbox alinha itens individuais dentro de um contêiner flexível. Fim da descrição.

Outro ponto importante é que o flexbox busca facilitar e automatizar a estilização de layouts, por isso o comportamento padrão é tentar ajustar os itens para se manterem dentro do layout da forma mais flexível possível. Veremos, de uma só vez, três propriedades que estão bastante entrelaçadas:

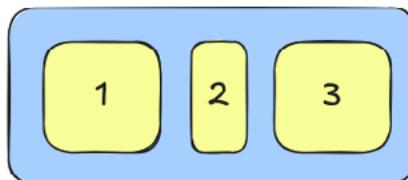
- **Flex-shrink:** a capacidade de um item encolher, caso o contêiner seja menor que o necessário (padrão: 1).
- **Flex-grow:** a capacidade de um elemento crescer, caso sobre espaço em um contêiner (padrão: 0).
- **Flex-basis:** o tamanho base de um elemento flexbox.

Vejamos, na Figura 8, o comportamento dessas propriedades.

Comportamento padrão



```
#item1, #item3 {  
    flex-shrink: 0;  
}
```



```
#item2 {  
    flex-grow: 1;  
}
```

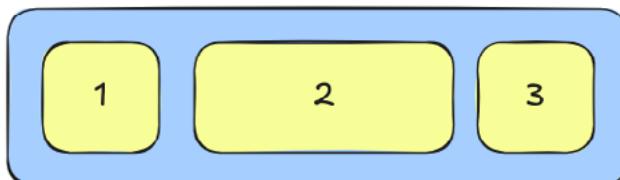


Figura 8 – Comportamento de esticar ou encolher elementos / Fonte: o autor.

Descrição da Imagem: representação visual das propriedades do CSS flexbox. Há três seções separadas, cada uma com um título e um conjunto de três caixas numeradas de 1 a 3. A primeira seção é intitulada "Comportamento padrão" e exibe as três caixas de tamanho igual, dentro de um contêiner azul. A segunda seção tem o título "#item3 { flex-shrink: 0; }", indicando que os itens 1 e 3 têm a propriedade flex-shrink definida como zero, assim, essas duas caixas mantêm seu tamanho, enquanto a caixa número 2 aparece menor. A terceira seção é rotulada "#item2 { flex-grow: 1; }", mostrando que o item 2 tem a propriedade flex-grow definida como 1, representado pela caixa número 2 sendo maior do que as caixas número 1 e número 3. Fim da descrição.

Vale a pena estudar mais a fundo as propriedades do flexbox, assim como a utilização do **CSS grid**, que seria uma forma similar ao flexbox para criar e controlar layouts, porém um pouco mais poderosa, para layouts mais complexos.

LAYOUTS RESPONSIVOS POR MEIO DO MEDIA QUERY

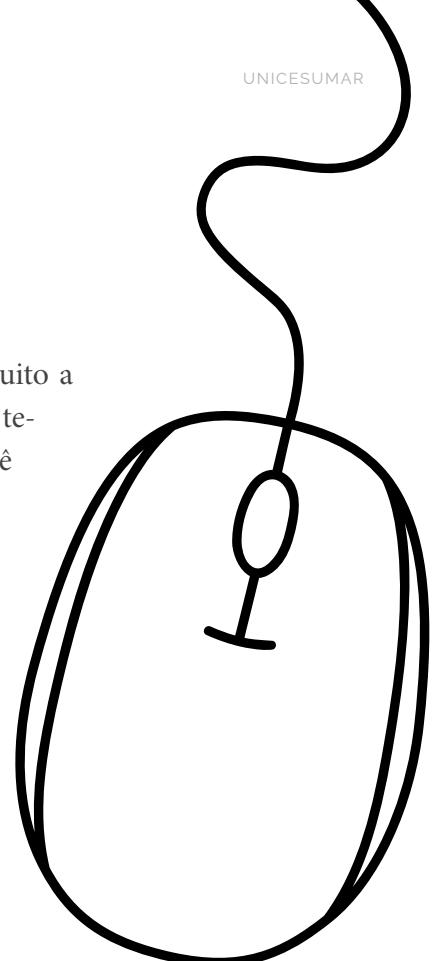
Utilizar o modelo flexbox é algo que facilita muito a adaptação de layouts para diferentes tamanhos de telas, porém não é suficiente, em alguns casos. Você pode controlar as propriedades de seu CSS de forma especial para diferentes mídias ou tamanhos de telas utilizando as diretivas @media, conhecidas como media queries (Consultas [...], 2021, on-line).

A ideia é desenvolver toda a definição de CSS normalmente, como seria feita, e caso alguma propriedade precise ser definida de forma diferente para uma diferente mídia, isso deve ser indicado explicitamente.

Para definir, utilizamos a palavra reservada **@media**, seguida da indicação de qual mídia queremos usar (podemos combinar parâmetros, como veremos, a seguir). Depois, as definições de propriedades CSS que desejamos definir. Vejamos através de exemplos.

```
@media print {  
    body {  
        background-color: transparent;  
    }  
}
```

A partir do código, definimos que, para a versão impressa da página, ou seja, a mídia “print”, a tag “body” não deve exibir a cor de fundo, a fim de economizar tinta na impressão.



```
@media screen and (max-width:800px) {  
    main {  
        flex-direction: column;  
    }  
}
```

O código apresentado define que, caso a mídia seja uma tela (monitor, smartphone etc.), e o tamanho máximo seja 800px, isto é, uma tela pequena, a parte principal do site, indicada pela tag “main”, deve mudar a direção do flexbox para coluna, a fim de melhor alocar o conteúdo.



EU INDICO

Caso queira aprofundar-se na criação de layouts em CSS e aprender a utilizar variáveis em CSS, enquanto desenvolve um sistema de modo escuro (*dark mode*) para seu site, assista ao vídeo, a seguir: <https://www.youtube.com/watch?v=gXMSSID5qKI>

Perceba que, dentro das media queries, é possível redefinir quais propriedades quisermos, utilizando a mesma sintaxe de seletores e propriedades CSS habituais. Essas propriedades serão renderizadas pelo navegador apenas se estiverem utilizando as mídias com os parâmetros especificados.

Assim, conseguimos criar layouts que se adaptam mais facilmente às diferentes realidades e diferentes telas. Isso é importante porque, hoje, o acesso móvel é algo muito comum, e a navegação por telas pequenas é, inclusive, mais frequente do que a navegação utilizando grandes monitores.



EM FOCO

Assista à videoaula para mais informações e conhecimentos sobre este tema de aprendizagem. Recursos de mídia disponíveis no conteúdo digital do ambiente virtual de aprendizagem.

NOVOS DESAFIOS

Desenvolver para web envolve criar interfaces amigáveis, com boa usabilidade e bom tempo de resposta, de forma a deixar a experiência do usuário algo leve e prazeroso. Dominar a construção de interfaces é essencial para alguém que deseja trabalhar com desenvolvimento front-end.

Além do conteúdo visto neste material, convém estudar um pouco a tipografia, que é o estudo das fontes utilizadas em uma publicação. Empregar fontes bonitas não é apenas embelezamento, transmite profissionalismo e melhora também a usabilidade. Você pode usar sistemas como o Google Fonts, que permite a incorporação de inúmeras fontes profissionais em seu software, sem pesar o seu serviço.

Também vale a pena estudar pré-processadores de CSS, que são sistemas como SASS e LESS. Eles possuem a capacidade de utilizar menos código para definir as propriedades CSS, criando algumas formas de reaproveitamento de código, além de realizar uma otimização nas definições de seus estilos.

Por último, um algo a mais para encantar e trazer uma boa experiência: você pode aproveitar recursos de animação e transições incorporadas no CSS3 e, assim, criar efeitos visuais que deixam a utilização mais suave e mais moderna.



Bons estudos
para você e sucesso!

VAMOS PRATICAR

1. Em uma página web, cada elemento é representado como um box retangular. Determinar o tamanho, propriedades, como sua cor, fundo, estilo das bordas, e a posição desses boxes é o objetivo do mecanismo de renderização (MDN Contributors, 2023, on-line).

No modelo de caixa (box model) do CSS, qual parte do elemento é responsável por definir a distância entre o conteúdo do elemento e suas bordas?

- a) Borda (border).
- b) Margem (margin).
- c) Conteúdo (content).
- d) Espaçamento (spacing).
- e) Preenchimento (padding).

2. “A criação de layouts responsivos na web é um dos desafios mais comuns e importantes para desenvolvedores front-end. Com a evolução das tecnologias CSS, Flexbox e Grid Layout emergiram como soluções poderosas e flexíveis para o design de interfaces complexas e responsivas” (Clemente, 2024, on-line).

A respeito da utilização do modelo flexbox, avalie as afirmações, a seguir:

- I - O flexbox permite alinhar itens horizontal e verticalmente.
- II - A propriedade flex-direction define a direção de fluxo dos itens no contêiner flexível.
- III - O flexbox substitui completamente o grid layout em projetos modernos.

É correto o que se afirma em:

- a) I, apenas.
- b) III, apenas.
- c) I e II, apenas.
- d) II e III, apenas.
- e) I, II e III.

VAMOS PRATICAR

3. “O Flexbox é um conceito do CSS3 que visa organizar os elementos de uma página HTML dentro de seus containers de forma dinâmica. Ou seja, independente das suas dimensões eles sempre manterão um layout flexível dentro do seu elemento pai, reorganizando-se e acordo com a necessidade” (Gaspar, 2013, on-line).

Qual das seguintes alternativas descreve corretamente a função da propriedade justify-content em um contêiner flexbox?

- a) Alinha itens na direção horizontal.
- b) Define a direção do fluxo dos itens.
- c) Define a ordem dos itens no contêiner.
- d) Controla a largura do contêiner flexbox.
- e) Alinha itens ao longo da direção principal do fluxo.

REFERÊNCIAS

BELL, A. Modelo de caixa. **web.dev.**, [s. l.], 29 mar. 2021. Disponível em: <https://web.dev/learn/css/box-model?hl=pt>. Acesso em: 10 out. 2024.

CLEMENTE, P. Flexbox e Grid Layout: Aprenda de maneira prática. **Blog da Rocketseat**, [s. l.], 24 mar. 2024. Disponível em: <https://blog.rocketseat.com.br/flexbox-e-grid-layout-aprenda-de-maneira-pratica/>. Acesso em: 10 out. 2024.

CONSULTAS de mídia. **web.dev.**, [s. l.], 3 nov. 2021. Disponível em: <https://web.dev/learn/design/media-queries?hl=pt-br>. Acesso em: 10 out. 2024.

GASPAR, F. CSS3 Flexbox: Funcionamento e Propriedades. **DevMedia**, Rio de Janeiro, 2013. Disponível em: <https://www.devmedia.com.br/css3-flexbox-funcionamento-e-propriedades/29532>. Acesso em: 10 out. 2024.

MDN CONTRIBUTORS. Box model - CSS. **MDN Web Docs**, [s. l.], 8 nov. 2023. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/CSS/CSS_box_model/Introduction_to_the_CSS_box_model. Acesso em: 10 out. 2024.

PERETTA, B. Iniciando com FlexBox. **Medium**, [s. l.], 3 fev. 2019. Disponível em: <https://medium.com/@peretta.breno/introdu%C3%A7%C3%A3o-ao-flexbox-9d715db30a5b>. Acesso em: 10 out. 2024.

CONFIRA SUAS RESPOSTAS

1. Alternativa E.

O preenchimento (padding) define o espaço entre o conteúdo do elemento e as suas bordas.
A alternativa A está incorreta, pois a borda fica entre a margem e o preenchimento.
A alternativa B está incorreta, pois margem (margin) é o espaço externo.
A alternativa C está incorreta, pois conteúdo é o conteúdo.
A alternativa D está incorreta, pois não existe a camada “espacamento” no box model.

2. Alternativa C.

A afirmação I está correta, pois o flexbox permite alinhar itens horizontal e verticalmente.
A afirmação II está correta, pois a propriedade flex-direction define a direção de fluxo dos itens no contêiner flexível.
A afirmação III está incorreta, pois o flexbox e o grid layout têm propósitos complementares, não sendo o flexbox um substituto completo do grid.

3. Alternativa E.

A propriedade justify-content alinha os itens ao longo da direção principal (horizontal ou vertical, dependendo da flex-direction).
A alternativa A está incorreta, pois o alinhamento é realizado no eixo principal, que nem sempre está na horizontal.
A alternativa B está incorreta, pois essa é a função da propriedade “flex-direction”.
A alternativa C está incorreta, pois essa é a função da propriedade “order”.
A alternativa D está incorreta, pois essa é a função da propriedade “flex-basis”.



unidade





DANDO VIDA AOS SISTEMAS COM JAVASCRIPT

MINHAS METAS

- Iniciar a criação de scripts integrados ao HTML.
- Inspecionar páginas no navegador.
- Conhecer as diferenças entre variáveis.
- Entender o conceito de DOM.
- Manipular os elementos do DOM.
- Estruturar dados em *arrays* e objetos.
- Criar funções em Javascript.



INICIE SUA JORNADA

Imagine uma aplicação de *e-commerce*, um jogo on-line ou até mesmo um simples formulário de contato. Todos esses sistemas precisam reagir às ações do usuário, como cliques, entradas de dados e comandos específicos.

Sem Javascript, essas ações não teriam resposta. A pergunta que se impõe é: como podemos criar sistemas que realmente se conectem com o usuário e entreguem uma experiência interativa e prática?

No contexto atual, a habilidade de criar sistemas interativos tornou-se essencial para os desenvolvedores. Javascript, além de ser uma das linguagens mais populares do mundo, também é o que dá vida a qualquer interface digital. Aprender Javascript vai além de dominar uma linguagem de programação, é entender como a web funciona e se comunica. Essa compreensão, muito mais do que um diferencial, é parte central do desenvolvimento profissional na área de tecnologia.

A melhor forma de aprender essa linguagem é pela experimentação. Por meio de pequenos scripts, você verá como um botão pode disparar uma mensagem, como um formulário pode ser validado antes mesmo de ser enviado, e uma imagem, animada ao toque do cursor. Essas experiências práticas são essenciais para que a teoria ganhe forma e torne-se algo concreto. Neste material, você terá a oportunidade de aplicar o que aprende de forma incremental, percebendo o impacto de cada linha de código no comportamento da sua aplicação.

Ao final, espera-se que você veja Javascript não apenas como uma ferramenta, mas como um modo de pensar e resolver problemas interativos. Com cada experiência e prática, seu olhar para as aplicações web se transformará. Pergunte-se sempre: “como isso melhora a experiência do usuário? Como meu código pode tornar o sistema mais fluido e dinâmico?” Essa reflexão contínua é o que permite transformar conhecimento técnico em habilidade prática, capacitando você a desenvolver sistemas realmente vivos.



PLAY NO CONHECIMENTO

Utilizar Javascript em um sistema web, além de essencial, é algo que pode trazer vida e agregar muito valor ao que você produz. A parte de interação com o usuário é capaz de definir a adoção ou não de um sistema pelos usuários. Para aprofundar-se nos motivos e casos de sucesso, ouça esse episódio de nosso podcast. **Recursos de mídia disponíveis no conteúdo digital do ambiente virtual de aprendizagem.**

VAMOS RECORDAR?

O desenvolvimento web é, antes de tudo, desenvolvimento. Por isso, o que aprendemos em qualquer linguagem de programação pode ser aproveitado em qualquer outra linguagem. Neste material, não abordaremos o básico de programação, como estruturas condicionais e estruturas de repetição. Para relembrar o conteúdo, assista a esse vídeo: <https://www.youtube.com/watch?v=X7-C3RAPOXA>

DESENVOLVA SEU POTENCIAL

Podemos escrever belas páginas web utilizando apenas HTML e CSS, porém serão páginas estáticas, “sem vida”. A partir da utilização de Javascript, é possível criar interações dinâmicas para melhorar a usabilidade e a apresentação de páginas web.

O conhecimento a ser adquirido neste tema é como barras de ouro, “que valem mais do que dinheiro”, como dizia Silvio Santos. Aproveitando, neste tema, será criado um jogo de trivia, estilo *Show do Milhão*, para aplicar os principais conceitos de Javascript. Ao final, você terá um jogo funcional e um bom entendimento das ferramentas essenciais da linguagem.

JAVASCRIPT: UMA LINGUAGEM CRIADA PARA O NAVEGADOR

O **Javascript** foi criado em 1995, para adicionar interatividade aos navegadores web, tornando páginas estáticas em experiências dinâmicas. Embora tenha evoluído muito desde então, inclusive chegando ao back-end, o propósito original dessa linguagem manteve-se: possibilitar a construção de sites que respondem às ações do usuário, como clicar em botões, preencher formulários, ou exibir informações em tempo real.

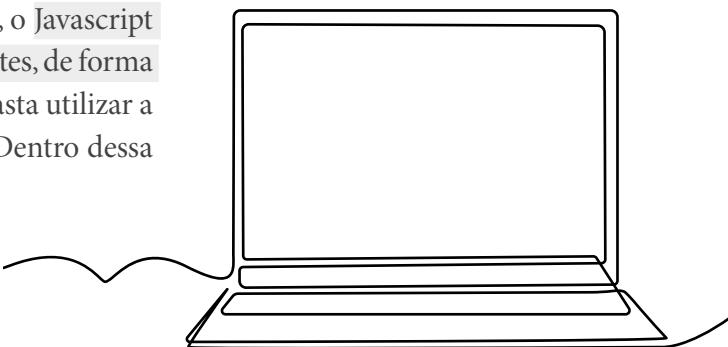
A linguagem Javascript foi originalmente batizada de **LiveScript**, sendo um projeto da **Netscape**. Em 1995, o projeto passou a ser desenvolvido em conjunto pela Netscape com a **Sun Microsystems**, que era criadora da linguagem **Java**, e o nome LiveScript foi modificado para Javascript (Sebesta, 2011).

No final dos anos 1990, a **ECMA** (*European Computer Manufacturers Association* – Associação Europeia de Fabricantes de Computadores) definiu um padrão para Javascript, o padrão **ECMA-262**. A partir de então, a linguagem passou a ser conhecida também como **ECMAScript**, principalmente quando se menciona o padrão da linguagem.



O padrão Javascript é ECMAScript. Desde 2012, todos os navegadores modernos possuem suporte total ao ECMAScript 5.1. Navegadores mais antigos suportam pelo menos ECMAScript 3. Em 17 de Junho de 2015, a ECMA International publicou a sexta versão do ECMAScript, que é oficialmente chamado de ECMAScript 2015, e foi inicialmente conhecido como ECMAScript 6 ou ES6. Desde então, as especificações do ECMAScript são lançadas anualmente (MDN Contributors, 2022, on-line).

Como toda linguagem de scripts, o Javascript pode ser escrito em pequenas partes, de forma embutida no HTML. Para isso, basta utilizar a tag `<script>` dentro do HTML. Dentro dessa tag, coloca-se código Javascript.



Um pouco de sintaxe

Não abordaremos o básico de programação, considerando que você já tenha passado por essa parte. Caso não tenha essa noção ainda, faça uma pausa nos estudos e veja a indicação no início do tema.

Javascript segue uma sintaxe similar à sintaxe da linguagem C, o que é comumente chamado de **C-style**, com utilização de chaves para marcar o início e o fim de bloco. Ao contrário da linguagem C, porém, é uma linguagem **fracamente tipada** e **dinamicamente tipada**, ou seja, ela permite operações com valores de diferentes tipos sem dar erros, e você não precisa declarar os tipos das variáveis, sendo que elas assumem o tipo a partir do valor atribuído.

Pela especificação, o Javascript possui dois tipos de dados (Groner, 2018):

- **Tipos primitivos:** aqueles tipos mais simples e predefinidos na linguagem. `null` (nulo), `undefined` (indefinido), `string` (texto), `number` (número), `boolean` (valor lógico, verdadeiro ou falso) e `symbol` (símbolo).
- **Tipos de dados derivados/objetos:** objetos Javascript, incluindo funções, `arrays` e expressões regulares.



Podemos escrever um pequeno trecho de Javascript dentro de um HTML, como no exemplo, a seguir:

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <title>Meu primeiro código</title>
</head>
<body>
    <script>
        document.body.style.backgroundColor = "blue";
    </script>
</body>
</html>
```

É possível fazer muita coisa diferente utilizando Javascript, principalmente manipulações de elementos do HTML. No exemplo apresentado, simplesmente foi modificada a cor de fundo do site para azul, manipulando a propriedade “style”, que define o CSS dos elementos. Nesse caso, foi aplicada a propriedade na área “body” do documento.

APROFUNDANDO

Como Javascript é uma linguagem fracamente tipada e permite trabalhar com tipos diferentes, há uma forma especial para testar se dois valores são mesmo iguais. O operador `==` testa se dois valores são iguais, porém ignora os tipos. Assim a expressão `1 == "1"` retorna verdadeiro, ainda que uma seja um número, e a outra, texto. Para comparar valor e tipo de dados, existe o operador `===`, dessa forma, `1 === "1"` retorna falso, porque os tipos não são iguais, apesar de ser o mesmo valor.

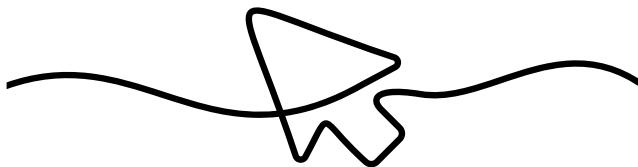
Já podemos, então, iniciar um jogo de trivia. Para isso, criaremos a estrutura HTML e CSS sem dar muitos detalhes aqui, a fim de economizar espaço.

 EU INDICO

Você pode obter o código inicial nesse endereço: <https://github.com/andre-noel/trivia-iniciante>

Inicialmente, temos apenas três arquivos em nosso projeto:

- **index.html**, com as definições da estrutura do nosso documento, definindo a área das perguntas, alternativas e botões.
- **styles.css**, com as definições de estilos, para que as informações já apareçam de forma agradável.
- **script.js**, que está, inicialmente, em branco, mas é onde a mágica acontecerá.



Adicionando Javascript ao HTML

Existem formas diferentes de adicionar seus scripts no HTML, mas sempre utilizando a tag `<script>`. É importante saber que o código Javascript é interpretado pelo navegador assim que lido, ou seja, **faz diferença** o lugar onde se adiciona a tag `<script>` no site. Caso você queira manipular um parágrafo em um site, o navegador precisa, primeiro, ter renderizado o parágrafo, para, depois, receber o código Javascript que tenta manipulá-lo.

Você pode adicionar o seu código diretamente dentro da tag `<script>`, como fizemos em nosso primeiro exemplo, ou pode usar um arquivo Javascript externo, como em nossos arquivos do projeto da trívia. Utilizar arquivos externos permite a reutilização fácil do código e a melhor separação tanto do conteúdo quanto do comportamento.

Para adicionar um arquivo externo, crie um arquivo apenas com código Javascript, e nomeie-o usando a extensão .js. Depois, você pode adicionar com o código:

```
<script src="arquivo.js"></script>
```

Lembrando que o código será interpretado no ponto do HTML onde for inserido. Por isso, é comum encontrar essa definição no final de sites, logo antes do fechamento da tag “body”.

```
...
<body>
    <h1>Show do Javascriptão</h1>
    <p>Você está certo disso?</p>
    <script src="arquivo.js"></script>
</body>
</html>
```

Para manter o código um pouco mais organizado, é possível adicionar o script no cabeçalho da página (dentro da tag `<head>`), com uma propriedade a qual indica que a renderização do Javascript deve ser “adiada”, ou seja, executada apenas depois do carregamento do documento HTML terminar.

```
<script src="arquivo.js" defer></script>
```

A propriedade “defer” indica que o navegador só deve interpretar o código do script após concluir a construção da estrutura do HTML. Isso é feito após a leitura do HTML, mesmo que não tenha ainda carregado conteúdos externos, como imagens e outros elementos.

UTILIZANDO AS FERRAMENTAS PARA DESENVOLVEDORES (*DEVTOOLS*)

Antes de escrever código, é importante conhecer as ferramentas que ajuda a entender e depurar o Javascript. As *DevTools*, ou ferramentas para desenvolvedores, são um conjunto de recursos disponíveis no navegador (em Chrome, Firefox etc.) que permitem testar comandos, inspecionar o DOM e monitorar o desempenho do código (adiante, entenderemos o que é o DOM).

A partir dessas ferramentas, você pode inspecionar o código HTML, os estilos CSS, visualizar o que está sendo trafegado pela rede, verificar dados armazenados no navegador em relação à sua navegação e utilizar um console Javascript, que é onde focaremos.

Para abrir o console Javascript, clique com o botão direito na página, selecione “Inspecionar”, ou pressione F12, e vá até a aba “Console”, na qual você pode testar comandos Javascript e ver mensagens de erro.

Usaremos o console para verificar se o Javascript está funcionando corretamente. Digite o código abaixo, no console:

```
console.log("Bem-vindo ao Show do Javascriptão!");
```

Essa linha exibirá a mensagem “Bem-vindo ao Show do Javascriptão!” no console, confirmando que o Javascript está ativo. Mais tarde, também utilizaremos o console para monitorar variáveis e verificar se o jogo está funcionando como esperado.



```
> console.log("Bem-vindo ao Show do JavaScriptão!");
  Bem-vindo ao Show do JavaScriptão! VM113:1
< undefined
```

Figura 1 - Execução de código no console do navegador / Fonte: o autor.

Descrição da Imagem: a primeira linha contém o código `console.log("Bem-vindo ao Show do Javascriptão!");`; A segunda linha contém a resposta da execução: Bem-vindo ao Show do Javascriptão! A terceira linha traz a palavra "undefined". Fim da descrição.

A Figura 1 exibe um trecho do console do navegador, onde podemos utilizá-lo como um interpretador de código interativo, digitando qualquer código que você queira testar, sem resultado permanente. Também é possível, de dentro do seu código, utilizar o comando `console.log()` para exibir mensagens que ficarão apenas ali, percebidas apenas quando se abre o console.

EU INDICO

O console do navegador não é a forma mais indicada de depuração (*debug*) de código, mas ele é muito útil durante o desenvolvimento. Se quiser aprender mais das funcionalidades do console, assista ao vídeo, a seguir: <https://www.youtube.com/watch?v=x5a4HyDMoNM>

Agora, estamos mais preparados para seguir adiante com a programação. Revisaremos conceitos que você pode reaproveitar de outras linguagens, mas com algumas particularidades em Javascript.

GUARDANDO DADOS EM VARIÁVEIS

A forma mais simples de guardar dados em programação, de forma temporária e na memória principal, é utilizando variáveis, o que precisamos fazer o tempo todo. Em uma linguagem fortemente tipada, como a linguagem C, você declara uma variável indicando o tipo de dado e o nome da variável. Como o Javascript é

uma linguagem dinamicamente tipada, não precisamos declarar o tipo de dado, mas temos palavras especiais para identificar que estamos declarando variáveis.

- **var**: usado nas versões mais antigas do Javascript. Não recomendado, pois pode causar confusões com escopo.
- **let**: ideal para variáveis que mudam de valor. Por exemplo, a pontuação do jogo.
- **const**: usado para variáveis que não mudam de valor, ou seja, constantes, como o número total de perguntas.

Em geral, a recomendação é utilizar “const” para todas as variáveis, caso perceba que ela precisará mudar de valor, você altera para “let”. Usar “const” garante imutabilidade ao evitar que o valor da variável seja reatribuído, tornando o código mais seguro e previsível. Não use “var” (não, não estamos falando de futebol).

```
let rodada = 0; // Qual é a rodada atual do jogo  
const totalPerguntas = 17; // Total de perguntas no jogo
```

A rodada atual é alterada a cada término de rodada, porém o total de perguntas nunca muda.

ENTENDENDO E MANIPULANDO O DOM

Para utilizar Javascript em um site, é preciso ter DOM (dizem as lendas que não se pode ensinar Javascript sem utilizar esse trocadilho ruim.)

O **DOM (Document Object Model)** representa a estrutura da página HTML, permitindo que o Javascript interaja e modifique elementos na página. Isso é importante para nosso jogo, pois usaremos o DOM para exibir perguntas, alternativas e verificar as respostas do usuário.

Entender o DOM não é difícil, conceitualmente. Na verdade, é comum as pessoas confundirem-se na hora de manipular, mas, basicamente, o DOM é uma árvore hierárquica, onde os elementos são organizados a partir de uma raiz, o “document”, um objeto especial que representa o documento HTML.

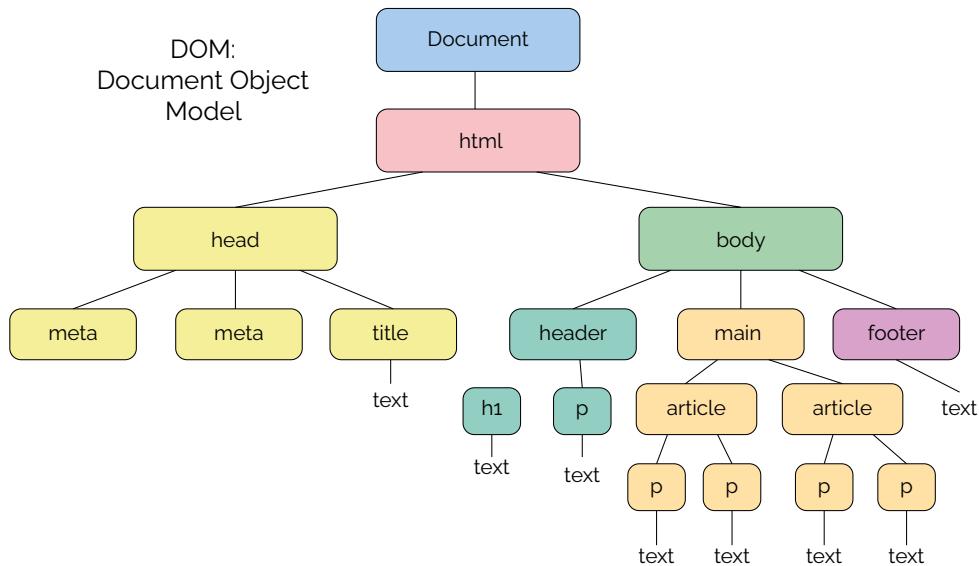


Figura 2 - Árvore do DOM de uma página HTML / Fonte: o autor.

Descrição da Imagem: elementos de uma página HTML dispostos em forma de uma árvore, similar a um organograma. No topo, o elemento “document”, diretamente abaixo “html”, que se divide em “head” e “body”. “head” divide-se em “meta”, “meta” e “title”, sendo que “title” tem um “text” abaixo. “body” divide-se em “header”, “main” e “footer”, “header” divide-se em “h1” e “p”, estas duas últimas possuem um “text” abaixo de cada. “main” divide-se em dois elementos “article”, cada elemento “article” divide-se em dois “p”, que possuem “text” abaixo. O elemento “footer” também possui “text” abaixo. Fim da descrição.

Todos os elementos HTML também são **nós (nodes)** no DOM, sendo que o texto puro é também tratado como um nó, mas não é um elemento do HTML.

Entendendo a estrutura, podemos selecionar os elementos de um HTML já montado e manipular suas propriedades.

Selecionando e alterando elementos no DOM

Usaremos métodos como `getElementById` e `querySelector` para selecionar elementos HTML que queremos modificar. Para entender melhor e ter uma referência, veremos as principais funções de captura de elementos do DOM (Menezes, 2023, on-line).

GETELEMENTBYID

Seleciona e retorna um elemento HTML, a partir do id, que é transmitido como parâmetro.

GETELEMENTSBYTAGNAME

Seleciona e retorna um conjunto de elementos no formato "HTML Collection", com todos os elementos definidos com a tag transmitida por parâmetro.

GETELEMENTSBYCLASSNAME

Seleciona e retorna um conjunto de elementos no formato "HTML Collection", com todos os elementos que contenham a classe transmitida por parâmetro.

QUERYSELECTOR

Seleciona e retorna um único elemento, o primeiro elemento, a partir de seletores CSS transmitidos por parâmetro.

QUERYSELECTORALL

Seleciona e retorna um conjunto de elementos no formato "NodeList" a partir de seletores CSS transmitidos por parâmetro.

Exibiremos a primeira pergunta do jogo, a partir da manipulação do DOM. Para melhor aprendizagem, você pode aplicar esse código no arquivo script.js do projeto baixado anteriormente.

```
const enunciado = document.getElementById("enunciado");
enunciado.innerHTML = "Quem descobriu o Brasil?";
```

No código anteriormente apresentado, na primeira linha, selecionamos o elemento HTML e o colocamos na variável “enunciado”. Na segunda linha, alteramos a propriedade “innerHTML”, a qual significa o conteúdo HTML que existe entre as *tags* de abertura e fechamento do elemento.

Seria possível fazer tudo em uma só linha, sem a utilização de variáveis, mas assim o código fica mais limpo, e podemos reutilizar a variável “enunciado” para mudar a pergunta outras vezes.

Quando manipulamos um elemento do DOM via Javascript, podemos alterar qualquer uma de suas propriedades, utilizando a sintaxe de objetos em Javascript. Por exemplo, alterar a propriedade “style” do enunciado usando a sintaxe “enunciado.style”.

ARRAYS E OBJETOS EM JAVASCRIPT

Arrays e objetos são estruturas fundamentais de dados em Javascript. No nosso jogo de trivia, usaremos *arrays* para armazenar as perguntas e objetos para representar cada pergunta com suas alternativas e respostas corretas.

Um *array* é uma lista de valores que podem se repetir, portanto, a posição do elemento no *array* faz diferença. Podemos acessar cada item do *array* por meio de um índice, que começa em zero. *Arrays* são, geralmente, definidos como valores entre colchetes.

```
const frutas = ["banana", "limão", "pera", "jaca"];
console.log(frutas[2]); // exibe no console a palavra
"pera"
```

Já **objetos** são estruturas que permitem armazenar pares de chave-valor, o que é ideal para representar uma pergunta e suas respostas. Objetos são definidos entre chaves, contendo os pares chave-valor separados por vírgulas.

```
{
    pergunta: "Qual o nome de batismo do Superman?",  

    alternativas: [ "Bruce Wayne", "Wally West", "Clark  

    Kent", "Speedy Gonzales"],  

    resposta: "Clark Kent"  

}
```

No objeto anteriormente apresentado, repare que temos três conjuntos de chave - valor. As chaves são definidas como texto, com a possibilidade de terem qualquer valor e serem definidas, também, entre aspas. Os valores podem ser de quaisquer tipos de dados, sendo que “pergunta” e “resposta” são do tipo texto (*string*), enquanto “alternativas” é do tipo *array*. Os valores podem ser até mesmo novos objetos ou *arrays* de objetos, o que é bastante comum.

Para acessar uma propriedade de um objeto, existem duas formas: por meio da sintaxe de ponto ou de colchetes. Vejamos a partir de um exemplo.

```
const novaPergunta = {};  

novaPergunta.pergunta = "Quantos números diferentes  

podem ser representados em um bit?";  

novaPergunta["resposta"] = 2;
```

Na primeira linha, criamos um objeto vazio. Na segunda linha, definimos a propriedade “pergunta” do objeto usando a sintaxe de ponto. Na terceira linha, definimos a propriedade “resposta” do objeto usando a sintaxe de colchetes, similar ao uso de *arrays*. Note que é preciso usar aspas, porque a sintaxe de colchetes pega o valor da *string*, mas também deixa mais dinâmico, permitindo, também, passar variáveis.



FUNÇÕES TRADICIONAIS E ARROW FUNCTIONS EM JAVASCRIPT

Funções permitem organizar o código, executando blocos específicos de instruções, sendo um bloco de código, geralmente, nomeado para ser reaproveitado. Em Javascript, podemos criar funções tradicionais ou *arrow functions*, introduzidas no ES6, que têm uma sintaxe mais curta.

As funções tradicionais são declaradas com a palavra reservada **function**. Para definir uma função, utiliza-se a sintaxe:

```
function nomeDaFunção (parâmetros) {  
    trecho de código;  
}
```

As *arrow functions* são funções com uma sintaxe simplificada que utilizam uma flecha (`=>`). São funções anônimas, mas às quais é possível atribuir um nome, definindo-as em variáveis. Por exemplo:

```
const nomeDaFunção = (parâmetros) => {  
    trecho de código;  
}
```

Em geral, funções tradicionais e *arrow functions* são exatamente a mesma coisa. Elas têm pequenas diferenças práticas que não farão diferença em nosso contexto neste material, e possuem algumas regras de simplificação:

- Caso a função tenha apenas uma linha de código com apenas um retorno de valores, as chaves e a palavra “return” podem ser omitidas.
- Se a função tiver apenas um parâmetro, os parênteses do parâmetro podem ser omitidos.
- Caso não haja parâmetros, os parênteses dos parâmetros podem ser omitidos e utiliza-se um “underscore” (`_`) no lugar do parâmetro.

Apesar de ser possível, o segundo e o terceiro caso não são muito recomendados para preservar a legibilidade do código. Veja, a seguir, algumas equivalências:

```
function soma (a, b) {
    return a + b;
}

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

function somaDois(numero) {
    return numero + 2;
}

const somaDoisArrow = numero => numero + 2;

function pi () {
    return 3.14;
}

const piArrow = _ => 3.14;
```

Juntando o que aprendemos, agora, é possível começarmos a escrever o código de nosso jogo, criando uma função que exibe as perguntas nos lugares certos. Coloque esse código em script.js.

```
const perguntas = [
    {
        enunciado: "Qual é o maior planeta do sistema solar?",
        alternativas: ["Júpiter", "Saturno", "Urano", "Netuno"],
        resposta: "Júpiter",
    },
    {
```

```
        enunciado: "Qual é o menor planeta do sistema
solar?",

        alternativas: ["Mercúrio", "Vênus", "Marte",
"Plutão"],

        resposta: "Mercúrio",

    },

];

const enunciado = document.
getElementById("enunciado");

const alternativa1 = document.querySelector("..
alternativa1");

const alternativa2 = document.querySelector("..
alternativa2");

const alternativa3 = document.querySelector("..
alternativa3");

const alternativa4 = document.querySelector("..
alternativa4");

function exibePergunta(numero) {

    enunciado.innerText = perguntas[numero].enunciado;

    alternativa1.innerText = perguntas[numero] .
alternativas[0];

    alternativa2.innerText = perguntas[numero] .
alternativas[1];

    alternativa3.innerText = perguntas[numero] .
alternativas[2];

    alternativa4.innerText = perguntas[numero] .
alternativas[3];

}

// Gera um número aleatório entre 0 e o total de
perguntas

const geraNumeroAleatorio = () => Math.floor(Math.
random() * perguntas.length);

exibePergunta(geraNumeroAleatorio());
```

Um jogo muito simples, apenas com duas perguntas, está feito. Cada vez que você entrar na página, verá uma pergunta sendo carregada aleatoriamente. Você pode aumentar a quantidade de perguntas e pensar em uma forma de verificar se a pessoa acertou a pergunta ou não.

Para deixar o jogo mais legal e mais encaminhado, há um arquivo `extras.js` no projeto, que conta com um pouco de comportamento, como selecionar as alternativas e o valor do prêmio a cada rodada. Adicione em seu HTML, incluindo mais uma linha de script, abaixo do script que você carregou:

```
<script src="extras.js" defer></script>
```

O que você aprende aqui, apesar de focado em um jogo, será útil em qualquer tipo de sistema, seja uma interface de fotos, seja um formulário de cadastro ou um gerenciamento de conteúdo de um site.

Bons estudos e que você consiga ganhar R\$ 1 milhão.



EM FOCO

Assista à videoaula para mais informações e conhecimentos sobre este tema de aprendizagem. **Recursos de mídia disponíveis no conteúdo digital do ambiente virtual de aprendizagem.**

NOVOS DESAFIOS

Chegar até aqui significa entender o poder transformador do Javascript na construção de sistemas web interativos e dinâmicos. Aprendemos a manipular elementos na página, controlando o comportamento e a resposta de sistemas que reagem a interações do usuário. No ambiente profissional, essas habilidades são altamente valorizadas, pois o Javascript é o núcleo de quase todas as interações no

navegador e nas aplicações de web modernas. Dominar essa linguagem e saber aplicá-la em um projeto prático, como fizemos com o jogo de trívia, é uma base sólida para enfrentar desafios reais no mercado de trabalho.

A capacidade de conectar HTML, CSS e Javascript em um único projeto também reflete o que é esperado em ambientes profissionais. No dia a dia de um desenvolvedor front-end, trabalhar com essas três camadas de uma página web — estrutura, estilo e comportamento — é uma rotina constante. Entender como cada camada se relaciona e aplicá-las juntas é essencial para desenvolver interfaces interativas, atraentes e, principalmente, intuitivas. Esse conhecimento permite que você se torne um desenvolvedor completo, capaz de produzir projetos funcionais do zero, o que é muito valorizado por empresas de tecnologia.

Além disso, o que estudamos, neste tema, de manipulação do DOM, uso de variáveis e criação de funções é um reflexo das operações que você encontrará em aplicações reais. No ambiente de trabalho, a habilidade de modificar elementos da página dinamicamente, ou seja, de personalizar a experiência do usuário de acordo com suas ações, é indispensável. Empresas procuram profissionais que não apenas dominem a técnica, mas que saibam aplicá-la de forma criativa para resolver problemas e criar experiências as quais agreguem valor ao produto.

Os conceitos, aqui, no entanto, são apenas o começo. O mercado exige atualização constante e o aprofundamento em temas mais avançados, como otimização de código, uso das API externas e, eventualmente, a introdução de *frameworks* como React ou Vue.js, que facilitam a construção de aplicações maiores e mais complexas. Por isso, é fundamental você continuar não apenas a explorar, mas também a praticar. Cada projeto, mesmo pequeno, será uma oportunidade de testar e fortalecer suas habilidades, preparando-lhe tanto para os projetos de maior escala quanto para os desafios mais sofisticados que enfrentará.

Por fim, é importante lembrar que o desenvolvimento de um bom profissional de front-end vai além do código. Envolve também trabalhar em equipe, compreender o impacto das soluções no usuário final e pensar estratégicamente as decisões de design e usabilidade. Ao construir essa base sólida e prosseguir com novos conhecimentos, você estará mais próximo de se tornar um profissional capacitado, versátil e apto a transformar o que aprendeu em soluções reais no ambiente profissional.

VAMOS PRATICAR

1. Considere o código, a seguir:

```
const usuarios = [  
    { id: 1, nome: "Alice", idade: 25, filhos: ["João", "Maria"] },  
    { id: 2, nome: "Bruno", idade: 30, filhos: [] },  
    { id: 3, nome: "Carla", idade: 22, filhos: ["Lucia"] }  
];
```

Considerando o código apresentado, avalie as seguintes afirmações:

- I - A variável "usuários" é um objeto.
- II - A variável "usuários" possui três objetos.
- III - A variável "usuários" é um array.
- IV - A propriedade "filhos" é um objeto dentro do objeto.

É correto o que se afirma em:

- a) I e IV, apenas.
- b) II e III, apenas.
- c) III e IV, apenas.
- d) I, II e III, apenas.
- e) II, III e IV, apenas.

2. Funções são estruturas fundamentais no desenvolvimento. Elas podem receber parâmetros, retornar valores e até mesmo ser atribuídas a variáveis. A flexibilidade no uso de funções permite desde simples cálculos até a implementação de complexos padrões de design. Compreender seu funcionamento e suas aplicações práticas é essencial para a construção de soluções eficientes e bem estruturadas.

A respeito de funções, em Javascript, avalie as afirmações, a seguir:

- I - *Arrow functions* são funções especiais criadas para serem executadas em arrays.
- II - Funções são trechos de códigos escritos para serem executados conforme a necessidade.
- III - As *arrow functions* são funções com uma sintaxe simplificada que utilizam uma flecha na definição.
- IV - Uma função é um trecho de código nomeado, pois não dá para utilizar funções se elas não tiverem um nome.

VAMOS PRATICAR

É correto o que se afirma em:

- a) I e IV, apenas.
 - b) II e III, apenas.
 - c) III e IV, apenas.
 - d) I, II e III, apenas.
 - e) II, III e IV, apenas.
3. Entender como selecionar, criar e alterar elementos no DOM é essencial para implementar funcionalidades, como validação de formulários, exibição de notificações ou atualizações dinâmicas de conteúdo.

Com base nas informações apresentadas, avalie as asserções, a seguir, e a relação proposta entre elas:

I - A manipulação do DOM é uma das principais funcionalidades de Javascript em páginas de web.

PORQUE

II - O DOM representa a estrutura de um documento HTML ou XML em forma de árvore.

A respeito dessas asserções, assinale a alternativa correta:

- a) As asserções I e II são verdadeiras, e a II é uma justificativa correta da I.
- b) As asserções I e II são verdadeiras, mas a II não é uma justificativa correta da I.
- c) A asserção I é uma proposição verdadeira, e a II é uma proposição falsa.
- d) A asserção I é uma proposição falsa, e a II é uma proposição verdadeira.
- e) As asserções I e II são falsas.

REFERÊNCIAS

GRONER, L. **Estruturas de dados e algoritmos com Javascript**. 2. ed. São Paulo: Novatec, 2018.

MDN CONTRIBUTORS. Javascript. **MDN Web Docs**, *ls. lJ*, 19 nov. 2022. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/Javascript>. Acesso em: 13 jan. 2025.

MENEZES, A de. 7 maneiras de selecionar elementos com Javascript sem querySelectorAll, querySelector e getElementById. **deMenezes**, *ls. lJ*, 26 jan. 2023. Disponível em: <https://demezes.dev/posts/7-alternativas-a-queryselectorall-queryselector-getelementbyid/>. Acesso em: 14 jan. 2025.

SEBESTA, R. W. **Conceitos de linguagens de programação**. 9. ed. Porto Alegre: Bookman, 2011.

CONFIRA SUAS RESPOSTAS

1. Alternativa B.

A afirmativa I está incorreta, pois a variável "usuários" é um *array* de objetos.

A afirmativa II está correta, pois a variável "usuários" possui três objetos.

A afirmativa III está correta, pois a variável "usuários" é um *array*.

A afirmativa IV está incorreta, pois a propriedade "filhos" é um *array* dentro do objeto.

2. Alternativa B.

A afirmativa I está incorreta, pois *arrow functions* são funções que podem ser usadas em qualquer contexto.

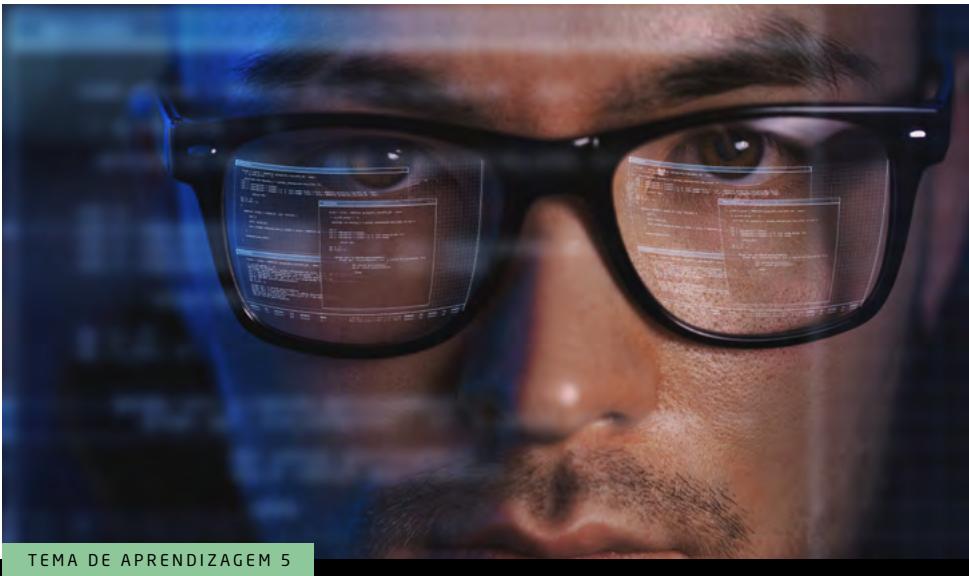
A afirmativa II está correta, pois funções são trechos de códigos escritos para serem executados conforme a necessidade.

A afirmativa III está correta, pois as *arrow functions* são funções com uma sintaxe simplificada que utilizam uma flecha na definição.

A afirmativa IV está incorreta, pois existem funções anônimas e elas são bastante utilizadas.

3. Alternativa A.

DOM é uma das principais funcionalidades para tornar-se um site dinâmico e permitir a manipulação de elementos, pois ele é uma representação do HTML ou XML em forma de árvore, permitindo a manipulação de qualquer parte do site.



TEMA DE APRENDIZAGEM 5

CRIANDO SISTEMAS PARA SEREM UTILIZADOS

MINHAS METAS

- Conhecer o Javascript moderno.
- Aprender funções de ordem superior.
- Utilizar filtros em *arrays*.
- Criar *arrays* novos com "map".
- Totalizar valores com "reduce".
- Lidar com armazenamento local no WebStorage.
- Desenvolver jogos para o navegador.



INICIE SUA JORNADA

Ao entrar no mundo do desenvolvimento web, um dos grandes desafios é entender como transformar conhecimento em projetos reais que atendam às necessidades dos usuários e tenham impacto no mundo digital.

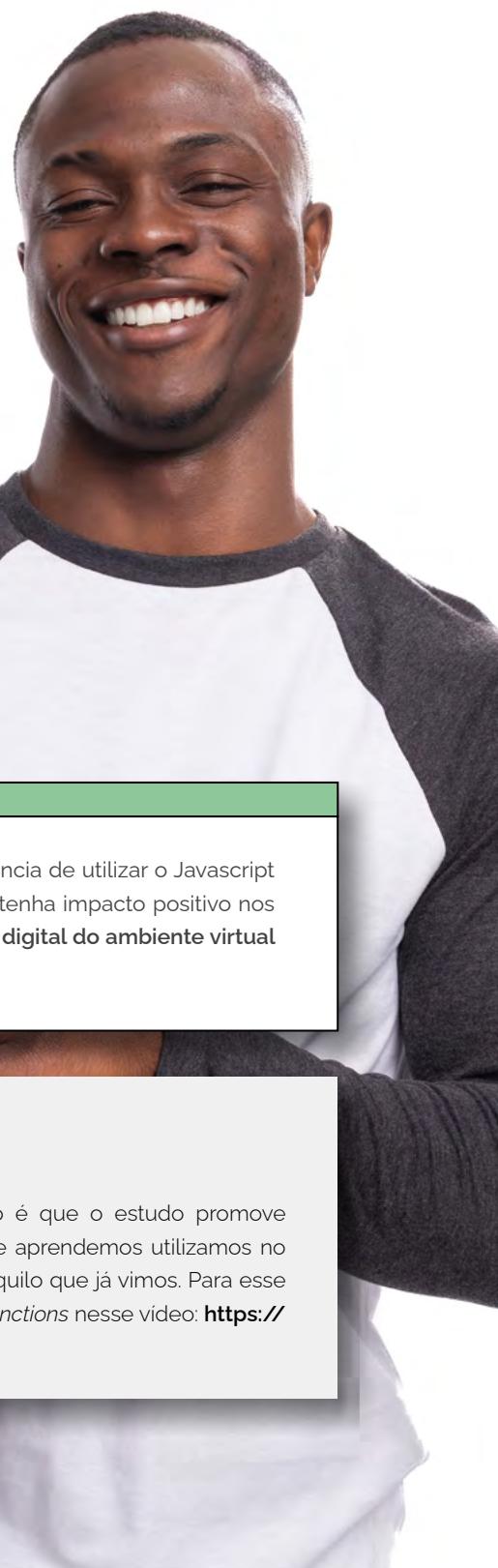
Agora, você irá aplicar Javascript para desenvolver sistemas interativos e funcionais. Focará em como criar um projeto que não apenas funcione bem tecnicamente, mas que também faça sentido para quem o usa.

Imagine essa situação: você está desenvolvendo uma aplicação para o seu primeiro cliente ou para um projeto importante de portfólio. Qual é o diferencial que a torna interessante e útil ao usuário? No mercado, criar uma página para apenas exibir informações não é mais suficiente. A verdadeira demanda é por sistemas interativos, nos quais cada ação do usuário tem uma resposta imediata e intuitiva, tornando a experiência enriquecedora. Pensar nisso desde o início é essencial para garantir que seu código seja funcional, mas principalmente, centrado no usuário.

Dominar o Javascript moderno e criar sistemas dinâmicos tem um propósito que vai além da programação em si: permite construir aplicações que façam sentido para as pessoas. Imagine quantos sites e aplicações você usa diariamente que memorizam suas preferências, ajustam-se ao seu comportamento e fornecem uma experiência personalizada. Cada uma dessas interações é guiada por código Javascript que armazena dados localmente, responde a eventos e manipula conteúdo conforme necessário. Ao entender e aplicar essas técnicas, você não só domina a ferramenta como também entende o impacto dela na vida das pessoas.

Este tema será a sua chance de experimentar. Você desenvolverá novas funcionalidades para o jogo de trívia, um exemplo prático e próximo de desafios reais que o mercado de trabalho pode trazer. Aqui, você brincará com o armazenamento local, explorará o poder das funções de ordem superior e entenderá como o Javascript moderno facilita o desenvolvimento. Cada linha de código que você escrever nesse exercício lhe aproximará de soluções reais bem como lhe preparará para criar sistemas interativos e independentes.

Ao final deste tema, reflita sobre como cada técnica contribui para o projeto e sob o que significa criar sistemas dinâmicos ao usuário final. A prática de armazenar dados e manipular *arrays* não é apenas uma habilidade técnica, é uma forma de entender o papel tanto da interatividade quanto da personalização nas aplicações. Conectar essa experiência ao seu processo de aprendizado e pensar em como essas técnicas podem ser aplicadas em projetos futuros é o passo que transforma conhecimento em expertise.



PLAY NO CONHECIMENTO

Ouça esse episódio de podcast que aborda a importância de utilizar o Javascript moderno e os seus recursos, para criar software que tenha impacto positivo nos usuários. **Recursos de mídia disponíveis no conteúdo digital do ambiente virtual de aprendizagem.**

VAMOS RECORDAR?

Algo interessante no aprendizado de programação é que o estudo promove sempre um aprendizado incremental, ou seja, o que aprendemos utilizamos no próximo assunto e, assim, aprendemos ainda mais aquilo que já vimos. Para esse conteúdo, vale a pena revisitar o conceito de *arrow functions* nesse vídeo: <https://www.youtube.com/watch?v=02236Epdh2Y>

DESENVOLVA SEU POTENCIAL

Aprofundaremos nosso conhecimento em desenvolvimento web, avançando para o Javascript moderno e explorando práticas que tornam o desenvolvimento mais eficiente e funcional.

Este tema trará ferramentas e técnicas para que nossos sistemas se tornem ainda mais dinâmicos e possam armazenar dados. Com essas novas habilidades, você será capaz de desenvolver sistemas interativos que não apenas capturam a atenção dos usuários, mas também proporcionam uma experiência personalizada.

O JAVASCRIPT MODERNO

Desde o seu surgimento, a linguagem Javascript sempre foi considerada, por alguns, uma linguagem limitada e sem muitas possibilidades além de pequenos efeitos e interações em páginas web.

Com o tempo, o Javascript passou por diversas atualizações que introduziram recursos poderosos e mais eficientes. No **ES6**, ou seja, no ECMAScript 2015, e nas versões seguintes, surgiram recursos como *arrow functions*, desestruturação de objetos e *arrays*, operador de espalhamento e módulos (Groner, 2018). Esses elementos não apenas modernizam o código, mas também o tornam mais legível e fácil de manter.

Com a utilização de elementos modernos, desenvolveremos um pedaço de um jogo para incorporar alguns desses recursos, iniciando pela desestruturação de *arrays* e objetos.

Uma aplicação relativamente simples, mas que ajuda a entender o funcionamento do desenvolvimento front-end, seria um jogo de perguntas e respostas, o famoso jogo de trívia.



EU INDICO

Para facilitar o desenvolvimento, parte dele já está desenvolvido, e o código pode ser obtido no repositório online. Acesse: <https://github.com/andre-noel/trivia>

Desestruturação de *arrays* e objetos

As alterações introduzidas a partir de ES6 visam diminuir a quantidade de código repetido e realizar melhor legibilidade do código (Pinho, 2018). Uma dessas funcionalidades é a desestruturação de *arrays* e objetos, para simplificar a sintaxe.

Para acessarmos elementos de *arrays*, usamos o nome do *array*, seguido de seu índice. Para acessar propriedades de um objeto, utilizamos o nome do objeto, seguido da propriedade, seja com a sintaxe de ponto, seja com a de colchetes.

Por exemplo, no jogo disponibilizado no repositório, há o conjunto de perguntas em um *array* da seguinte forma:

```
const perguntas = [  
  {  
    enunciado: "Qual é o maior planeta do sistema  
solar?",  
    alternativas: ["Júpiter", "Saturno", "Urano",  
"Netuno"],  
    resposta: "Júpiter",  
  },  
  {  
    enunciado: "Qual é o menor planeta do sistema  
solar?",  
    alternativas: ["Mercúrio", "Vênus", "Marte",  
"Plutão"],  
    resposta: "Mercúrio",  
  },  
];
```

Para obter a primeira alternativa da primeira pergunta do jogo, como ele está disponível no repositório, o acessamos como:

```
perguntas[0].alternativas[0]
```

Para recuperar as quatro alternativas, precisaríamos usar essa sintaxe quatro vezes. De uma forma simples, podemos desestruturar o *array* de alternativas salvando o conteúdo do *array* em variáveis:

```
const [ alt1, alt2, alt3, alt4 ] = perguntas[0].  
alternativas;
```

Dessa forma, nossa função de exibir a pergunta pode ser escrita da seguinte forma:

```
const exibePergunta = (numero) => {  
    enunciado.innerText = perguntas[numero].enunciado;  
    const [alt1, alt2, alt3, alt4] = perguntas[numero].  
alternativas;  
  
    alternativa1.innerText = alt1;  
    alternativa2.innerText = alt2;  
    alternativa3.innerText = alt3;  
    alternativa4.innerText = alt4;  
};
```

Ou ainda, de uma forma mais simples, sem usar variáveis, podemos escrever:

```
const exibePergunta = (numero) => {
    enunciado.innerText = perguntas[numero].enunciado;
    [
        alternativa1.innerText,
        alternativa2.innerText,
        alternativa3.innerText,
        alternativa4.innerText,
    ] = perguntas[numero].alternativas;
};
```

Resumindo, a desestruturação de *arrays* extrai os elementos de dentro do *array* para variáveis conforme a ordem em que forem definidas. Caso haja menos variáveis do que elementos do *array*, é possível criar, nessa sintaxe, um subconjunto do *array* em uma nova variável.

```
const [alt1, ...outras] = perguntas[numero].alternativas;
```

Nesse caso, a variável “alt1” contém a primeira alternativa, e a variável “outras” contém um *array* com as alternativas 2, 3 e 4.

A desestruturação de objetos funciona de forma similar, porém não a partir da ordem das variáveis, e sim usando os nomes das propriedades. Em nosso caso, podemos capturar as propriedades da pergunta de forma rápida, da seguinte maneira:

```
const { enunciado, alternativas, resposta } =
perguntas[0];
```

Se por acaso, quisermos chamar o “enunciado” de “pergunta”, para não dar conflito com o elemento “enunciado” do HTML, podemos redirecionar a propriedade da seguinte forma:

```
const { enunciado: pergunta, alternativas, resposta }  
= perguntas[0];
```

No caso de objetos, não precisamos usar as variáveis na mesma ordem, pois são as chaves que identificam os valores.

FUNÇÕES DE ORDEM SUPERIOR (HOFs)

As **Funções de Ordem Superior** (*Higher Order Functions* – HoFs) são funções que recebem outras funções como argumentos, ou retornam uma função (Sebesta, 2011). Elas são fundamentais para manipular *arrays* de forma poderosa e eficiente.

Em nosso jogo, usaremos HoFs para manipular as perguntas, embaralhar respostas e avaliar o progresso do jogador. Introduziremos algumas funções de ordem superior, como “map”, “filter” e “reduce”. Cada uma delas será explicada e aplicada para enriquecer nosso jogo:

- **map**: transforma cada item do *array* de perguntas em outro formato ou adiciona informações.
- **filter**: retorna apenas os elementos que satisfazem uma condição, como perguntas de uma categoria específica.
- **reduce**: acumula valores. É ideal para calcular a pontuação total.

Essas funções, especificamente, são funções definidas para atuarem sobre *arrays*, sendo que “**filter**” e “**map**” retornam um *array* novo, enquanto “**reduce**” retorna um elemento só, o qual podemos definir o formato.

No fundo, todas essas funções são personalizações de um looping *for* que percorre o *array*, iterando por todos os elementos. As funções que estudaremos recebem como parâmetro de entrada uma função.

Suponhamos que as perguntas tenham níveis, nos quais queremos que as cinco primeiras rodadas exibam apenas perguntas fáceis, as próximas cinco, de nível médio, e as demais, apenas difíceis. Assim, nossa pergunta deve ficar da seguinte forma:

```
{
    enunciado: "Qual é o maior planeta do sistema solar?",
    alternativas: ["Júpiter", "Saturno", "Urano",
    "Netuno"],
    resposta: "Júpiter",
    dificuldade: "Fácil",
}
```

Podemos filtrar o *array* de perguntas, criando um *array* apenas com as perguntas mais fáceis, com o seguinte código:

```
const perguntasFáceis = perguntas.filter((perg) =>
perg.dificuldade === "Fácil");
```

O parâmetro “*perg*” da função recebida na entrada pode ter qualquer nome, e ele recebe o item do *array* relativo a cada iteração, ou seja, “*perg*” é o item da vez, do primeiro até o último.

Em linhas gerais, a função “*filter*” percorre todo o *array* original, realizando o teste condicional declarado na função de entrada e retorna um *array* novo apenas com os elementos cuja função retorna “verdadeiro”.

Assim, a função “*filter*” gera um novo *array*, sempre com um número menor ou igual de elementos em relação ao original. Já a função “*map*” retorna o mesmo número de elementos que o original, mas com alguma modificação.

Podemos, por exemplo, gerar um novo *array* de perguntas, com apenas o enunciado e as alternativas, sem as respostas, da seguinte forma:

```
const arrayPerguntas = perguntas.map((perg) => ({
    pergunta: perg.enunciado,
    alt1: perg.alternativas[0],
    alt2: perg.alternativas[1],
    alt3: perg.alternativas[2],
    alt4: perg.alternativas[3],
})) ;
```

A função “**map**” percorre todo o *array*, retornando um *array* novo, onde cada elemento é o resultado direto da função de entrada, aplicada ao elemento em cada posição. Assim, a função “**map**” sempre retorna um *array* com o mesmo número de elementos do *array* original.

Em nosso sistema, não precisaremos realizar um cálculo de totalização de pontuação, porque usaremos uma tabela de pontuações nos moldes do *Show do Milhão*, mas é possível realizar totalizações, somas e demais acumulações de valores com a função “**reduce**”.

A função “**reduce**”, assim como “**map**” e “**filter**”, é uma repetição entre os elementos do *array*, com a diferença que ela possui um elemento extra, chamado “acumulador”. Se não for definido de forma explícita, o acumulador assume o primeiro elemento do *array* como valor. Esse acumulador pode ser atualizado como quisermos a cada iteração e, então, é transmitido para a próxima iteração, até chegar ao final, onde é retornado como valor final. Funciona com a seguinte sintaxe:

```
array.reduce((acumulador, valorAtual) => {
    operações;
    return acumulador;
}, acumuladorInicial);
```

O valor “acumuladorInicial” é opcional, mas pode, também, ser de qualquer tipo: número, *string*, *array*, objeto etc.

Suponhamos que cada pergunta tenha seu próprio valor de pontuação. Poderíamos somar as pontuações da seguinte forma:

```
const totalPontos = perguntas.reduce((acumulador,
  valorAtual) => acumulador + valorAtual.pontos, 0);
```

Como a nossa função teria apenas uma linha de retorno, com *acumulador + valorAtual.pontos*, aproveitamos a propriedade da *arrow function* que permite abreviar a sintaxe.

```
> perguntas
< ▷ (2) [{...}, {...}]
  ▷ 0:
    ▶ alternativas: (4) ['Júpiter', 'Saturno', 'Urano', 'Netuno']
    dificuldade: "Fácil"
    enunciado: "Qual é o maior planeta do sistema solar?"
    pontos: 50
    resposta: "Júpiter"
    ▶ [[Prototype]]: Object
  ▷ 1:
    ▶ alternativas: (4) ['Mercúrio', 'Vênus', 'Marte', 'Plutão']
    dificuldade: "Médio"
    enunciado: "Qual é o menor planeta do sistema solar?"
    pontos: 60
    resposta: "Mercúrio"
    ▶ [[Prototype]]: Object
    length: 2
    ▶ [[Prototype]]: Array(0)
> const totalPontos = perguntas.reduce((acumulador, valorAtual) => acumulador + valorAtual.pontos, 0);
< undefined
> console.log(totalPontos)
  110
< undefined
```

VM1366:1

Figura 1 - Execução do código do “reduce” no console do navegador / Fonte: o autor.

Descrição da Imagem: captura de tela do console, com o texto: perguntas (2) [..., ...]0: alternativas: (4) ['Júpiter', 'Saturno', 'Urano', 'Netuno'] dificuldade: "Fácil" enunciado: "Qual é o maior planeta do sistema solar?" pontos: 50 resposta: "Júpiter"[[Prototype]]: Object1: alternativas: (4) ['Mercúrio', 'Vênus', 'Marte', 'Plutão'] dificuldade: "Médio" enunciado: "Qual é o menor planeta do sistema solar?" pontos: 60 resposta: "Mercúrio"[[Prototype]]: Objectlength: 2[[Prototype]]: Array(0)const totalPontos = perguntas.reduce((acumulador, valorAtual) => acumulador + valorAtual.pontos, 0); undefined. console.log(totalPontos). Fim da descrição.

É possível realizar a mesma operação com loopings tradicionais, como *for* ou *while*, mas é fácil perceber como o código fica mais simples e enxuto com o uso de “reduce”.

ARMAZENANDO DADOS NO NAVEGADOR COM WEBSTORAGE

Já aconteceu de estar jogando algo no celular ou preenchendo um cadastro, daí você sai por algum motivo (fecha sem querer, bate o dedo no F5) e perde tudo o que já tinha feito?

Com o **WebStorage** (`localStorage` e `sessionStorage`), o jogo pode armazenar dados que persistem entre sessões do usuário. Utilizaremos o `localStorage` para registrar o progresso do jogador, as perguntas respondidas e a pontuação total. Com isso, mesmo que o usuário saia da página, o progresso é salvo e retomado ao retornar.

Tanto **localStorage** quanto **sessionStorage** são formas de armazenamento no navegador que permitem guardar dados, como *strings*, de forma fácil e acessível, mas existem algumas diferenças importantes entre eles (MDN Contributors, 2024, on-line).



PERSISTÊNCIA DOS DADOS

A principal diferença entre os dois é o tempo de persistência dos dados. Estes, quando armazenados no **localStorage**, são permanentes e continuam salvos mesmo após o fechamento do navegador, enquanto os dados no **sessionStorage** são temporários e apagados assim que a aba ou janela do navegador é fechada.

ESCOPO DE SESSÃO

O sessionStorage é limitado à aba ou à janela em que foi criado. Isso significa que os dados armazenados não estarão acessíveis em outras abas ou janelas, mesmo que estejam abertas no mesmo site. Em contrapartida, o **localStorage** é acessível por qualquer aba ou janela do mesmo navegador e mantém os dados compartilhados entre todas as sessões do site.

CASOS DE USO

Devido à sua persistência, o **localStorage** é ideal para armazenar preferências de usuário, *tokens* de autenticação ou configurações que devem ser mantidas entre visitas. Já o **sessionStorage** é mais adequado para informações de curto prazo, como dados temporários de uma sessão ativa ou estado de formulários que não precisam persistir depois que o usuário sair da página.

Usaremos, basicamente, o localStorage, mas a forma de salvar ou recuperar dados nos dois formatos é o mesmo.

De forma básica, visualizaremos o localStorage como um objeto Javascript, com a diferença que só armazena *strings*. Para salvar dados, definimos um par chave e valor, da mesma maneira que faríamos com um objeto comum ou usando a função “setItem”:

```
localStorage.setItem('pontuacao', '30');

localStorage['perguntaAtual'] = 'Qual é o maior
planeta do sistema solar?'

localStorage.resposta = 'Júpiter'
```



Caso queiramos saber se tudo está sendo salvo corretamente, podemos abrir as ferramentas de desenvolvimento (*DevTools*) apertando F12 e, em seguida, clicaremos na aba “Aplicativo”. Ela possui as informações armazenadas no site que estamos visitando.

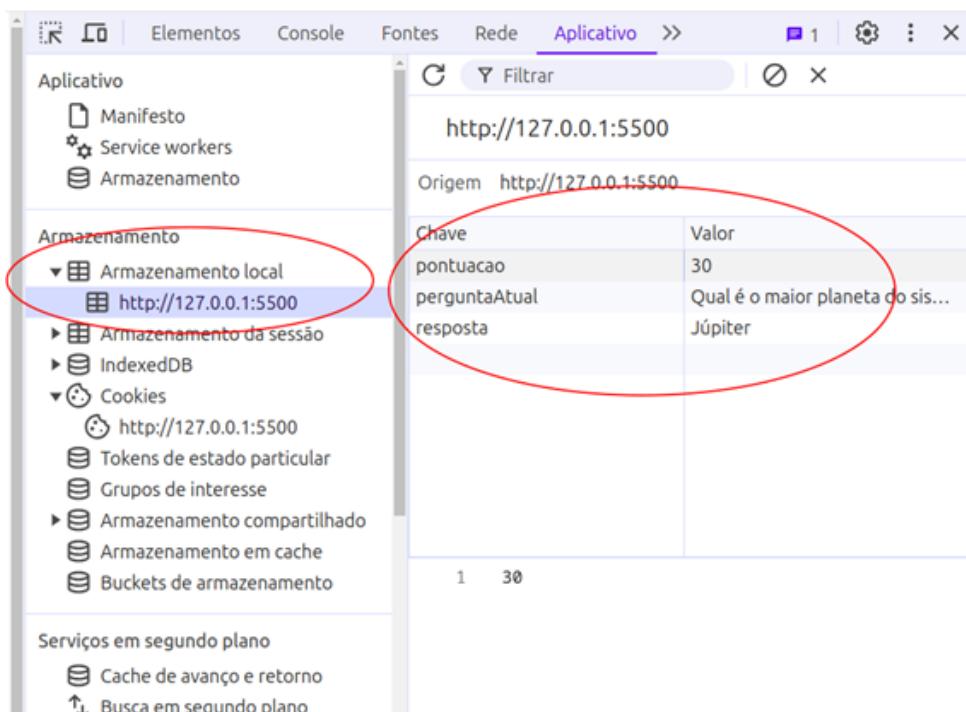


Figura 2 - Visualização dos dados armazenados no localStorage / Fonte: o autor.

Descrição da Imagem: captura de tela das ferramentas de desenvolvedor do navegador. Na coluna da esquerda, há os itens: Aplicativo, Manifesto, Service Workers, Armazenamento, uma divisória, Armazenamento, Armazenamento local, http://127.0.0.1:5500, que está selecionado e com um círculo vermelho destacando, Armazenamento da sessão, IndexedDB, Cookies, http://127.0.0.1:5500, Tokens de estado particular, Grupos de interesse, Armazenamento compartilhado, Armazenamento em cache, Buckets de armazenamento, uma divisória, Serviços em segundo plano, Cache de avanço e retorno, Busca em segundo plano. Na coluna da direita, há os dados com origem http://127.0.0.1:5500, seguido por uma tabela, Chave, Valor, pontuacao, 30, perguntaAtual, Qual é o maior planeta do sis..., resposta, Júpiter. Fim da descrição.

De forma similar, podemos recuperar as informações do localStorage pela função “`getItem`” ou a sintaxe utilizada com objetos. De forma prática, colocamos no início do nosso arquivo `script.js` uma verificação para saber há dados no localStorage. Guardamos a informação da rodada atual e qual é a pergunta atual.

```
if (localStorage.getItem("rodada") === null) {  
    rodada = 0;  
    localStorage.setItem("rodada", rodada);  
}  
else {  
    rodada = localStorage.getItem("rodada");  
}  
  
// Gera um número aleatório entre 0 e o total de  
perguntas  
const geraNumeroAleatorio = () => Math.floor(Math.  
random() * perguntas.length);  
  
if (localStorage.getItem("numPergunta") === null) {  
    numPergunta = geraNumeroAleatorio();  
    localStorage.setItem("numPergunta", numPergunta);  
}  
else {  
    numPergunta = localStorage.getItem("numPergunta");  
}
```

O sistema verifica, então, se há informações armazenadas no localStorage ao iniciar. Se houver, recupera o estado do sistema para o mesmo ponto. Depois, toda vez que atualizar a rodada ou a pergunta, essas informações deverão ser atualizadas.

DESENVOLVENDO O COMPORTAMENTO DO JOGO

Com as bases sólidas em Javascript moderno e armazenamento local, podemos, agora, expandir nosso jogo de trívia, tornando-o mais interativo e funcional. Implementaremos a lógica de pontuação e adicionaremos *feedback* visual conforme o usuário responde às perguntas corretam ou incorretamente.

Além disso, para tornar a experiência mais completa, adicionaremos animações ou efeitos visuais simples com Javascript, enriquecendo a interface sem a necessidade de bibliotecas externas. Esse tipo de desenvolvimento é muito valorizado no mercado, pois exige tanto conhecimento técnico quanto criatividade para criar experiências memoráveis aos usuários.

Agora, tanto o botão “Parar” quanto o botão “Perguntar” atualizarão o campo de mensagem que fica na parte direita da tela.

```
// Define o comportamento do botão Parar
btnParar.addEventListener("click", () => {
    delete localStorage.rodada;
    delete localStorage.numPergunta;
    mensagem.innerText = "Você parou o jogo!\nGanhou "
+ boxParar.innerHTML;
    rodada = 0;
    proximaPergunta();
});

// Define o comportamento do botão Perguntar
btnPerguntar.addEventListener("click", () => {
    if (document.querySelector(".selecionada") === null) {
        mensagem.innerText = "Selecione uma
alternativa!";
        return;
    }
}

const resposta = perguntas[numPergunta].resposta;
```

```
const alternativa = document.querySelector(".selecionada").innerText;

if (resposta === alternativa) {

    mensagem.innerText = "Você acertou!\nGanhou "
+ boxAcertar.innerHTML;

    rodada++;

    localStorage.rodada = rodada;

} else {

    mensagem.innerText = "Você errou!\nGanhou "
+ boxErrar.innerHTML;

    delete localStorage.rodada;

    rodada = 0;

}

proximaPergunta();

}) ;

const proximaPergunta = () => {

    numPergunta = geraNumeroAleatorio();

    localStorage.numPergunta = numPergunta;

    exibePergunta(numPergunta);

};
```

Assim, ao clicar no botão “Parar”, o sistema exibe na tela a mensagem de que o jogo foi parado, com a quantidade que o jogador ganhou. Em seguida, reinicia-se o jogo, chamando a próxima pergunta. Poderia apenas ficar parado e haver um botão para reiniciar, mas foi feito assim para aproveitar o trabalho já realizado.



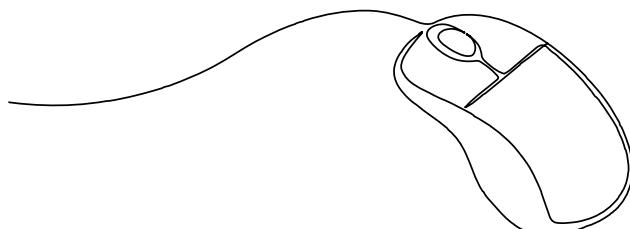


Figura 3 – Tela do jogo no navegador / Fonte: o autor.

Descrição da Imagem: interface em design colorido e chamativo de um jogo de perguntas e respostas com o título *Show do Javascriptão*. No lado esquerdo, há uma seção em roxo com bordas bem como elementos em amarelo e vermelho. A pergunta no topo é: "Qual é o maior planeta do sistema solar?". Abaixo da pergunta, há quatro opções de resposta, cada uma com um botão vermelho e bordas amarelas, essas opções são numeradas de 1 a 4, contendo as opções: Júpiter, Saturno, Urano, Netuno. Abaixo das respostas, existem dois botões adicionais, um roxo com bordas amarelas escrito "Pode perguntar?", um vermelho com bordas amarelas escrito "Parar!". Na parte inferior da seção roxa, há três botões que representam os valores de prêmios: R\$ 5 mil, marcado como "Errar", R\$ 10 mil, marcado como "Parar", R\$ 20 mil, marcado como "Acertar". No lado direito da tela, em um fundo branco, aparece a mensagem de feedback: "Você acertou! Ganhou R\$ 10 mil", escrita em roxo e em letras maiúsculas. Fim da descrição.

O botão “Pode perguntar?” é um pouco mais elaborado, pois ele precisa capturar a alternativa selecionada e comparar com a resposta correta. Caso esteja certo, segue adiante, mostrando quanto o jogador ganhou, e passa para a próxima rodada, atualizando, também, os valores.

Caso a resposta esteja errada, o jogador ganha o valor mais baixo, que aparece na caixa “boxErrar”. Em seguida, o jogo reinicia. Deveria haver um teste para quando o jogador chega ao final, na pergunta de R\$ 1 milhão, mas isso não foi implementado ainda.



**EM FOCO**

Assista à videoaula para mais informações e conhecimentos sobre este tema de aprendizagem. **Recursos de mídia disponíveis no conteúdo digital do ambiente virtual de aprendizagem.**

NOVOS DESAFIOS

Neste tema, foram exploradas técnicas essenciais de Javascript para criar sistemas interativos e dinâmicos, mergulhando em conceitos fundamentais, como funções de ordem superior e armazenamento local. Cada técnica praticada e cada linha de código escrita aproximam você do ambiente profissional, onde a capacidade de transformar ideias em aplicações úteis é uma habilidade altamente valorizada. No mercado de trabalho, espera-se que o desenvolvedor consiga criar experiências de usuário ricas e dinâmicas, e as habilidades que você desenvolveu aqui são o primeiro passo para atender a essa demanda.

A medida que você avança na construção de aplicações, será cada vez mais importante conectar teoria e prática, de maneira ágil. Funções de ordem superior e manipulação de dados, por exemplo, são técnicas que não só melhoram a eficiência do código, mas também o tornam mais legível e escalável, características essenciais em equipes profissionais. Saber como aplicar esses recursos de maneira prática, em contextos como o jogo de trívia, lhe permite ir além da teoria, compreendendo como resolver problemas reais.

No ambiente de trabalho, você encontrará a necessidade de desenvolver aplicações que atendam às especificações técnicas bem como às necessidades do usuário final. A capacidade de armazenar informações de maneira local e criar sistemas que reagem às interações do usuário, por exemplo, é uma habilidade altamente requisitada para construir interfaces responsivas e funcionais. Cada experiência interativa criada por você fortalece essa competência e aumenta seu diferencial no mercado.

Além disso, esse aprendizado lhe leva a outro patamar de desenvolvimento: entender como os dados podem ser manipulados e armazenados de forma eficaz. Empresas valorizam profissionais que sabem como lidar com dados e otimizá-los para fornecer a melhor experiência. Com o conhecimento adquirido, você pode começar a aplicar essas técnicas em projetos mais complexos, explorando novas oportunidades e abordagens.

Enfrentar esses desafios e refletir sobre a aplicação prática do Javascript moderno não só aprimora suas habilidades técnicas, mas também constrói uma base sólida ao desenvolvimento contínuo. Em cada projeto, você aproxima-se mais das exigências do mercado, o qual está equipado para criar sistemas completos e preparados ao usuário final, transformando teoria em prática e prática em experiência.



VAMOS PRATICAR

1. Não basta escrever um código bom. Ele precisa ser mantido sempre limpo. Todos nós já vimos códigos estragarem e degradarem com o tempo. Portanto, precisamos assumir um papel ativo na prevenção da degradação. O Javascript moderno apresenta novas funções e melhorias de sintaxe para criar softwares com código mais limpo e direto (Martins, 2009).

A respeito do Javascript moderno, assinale a alternativa correta:

- a) Também chamado de JScript, foi desenvolvido pela Microsoft para ser executado em navegadores da nova geração.
 - b) As definições de variáveis com a palavra reservada "var" foram introduzidas no ES6 para habilitar a propriedade de *hoisting*, onde a variável escapa de seu escopo.
 - c) No ES6 e nas versões seguintes, surgiram recursos como *arrow functions*, desestruturação de objetos e *arrays*, além de operadores de espalhamento.
 - d) As versões mais recentes de Javascript trazem um amadurecimento no desenvolvimento front-end, o que gerou ruptura e incompatibilidade com versões mais antigas da linguagem.
 - e) A utilização de Javascript moderno é possível graças à utilização da máquina virtual pelo navegador, conhecida como JVM, *Javascript Virtual Machine*.
2. Funções de Ordem Superior são aquelas que recebem ou retornam outras funções, permitindo manipular e transformar coleções de dados de forma declarativa.

Sobre as Funções de Ordem Superior, avalie as afirmações, a seguir:

- I - A função "filter" é usada para criar um *array* contendo apenas elementos que atendam a uma condição.
- II - A função "map" pode modificar cada elemento de um *array*, sem alterar o original.
- III - A função "reduce" é utilizada para combinar todos os elementos de um *array* em um único valor.

É correto o que se afirma em:

- a) I, apenas.
- b) III, apenas.
- c) I e II, apenas.
- d) II e III, apenas.
- e) I, II e III.

VAMOS PRATICAR

3. Para jogos no navegador, o armazenamento de dados nesse local é essencial para salvar progresso e preferências.

Qual é a maneira correta de salvar uma informação usando o localStorage?

- a) localStorage.save("progresso", progresso)
- b) localStorage.setItem("progresso", progresso)
- c) localStorage.savelitem("progresso", progresso)
- d) localStorage.addltem("progresso", progresso)
- e) localStorage.store("progresso", progresso)

REFERÊNCIAS

- GRONER, L. **Estruturas de dados e algoritmos com Javascript**. 2. ed. São Paulo: Novatec, 2018.
- MARTIN, R. C. **Código limpo**: habilidades práticas do Agile Software. Rio de Janeiro: Alta Books, 2009.
- MDN CONTRIBUTORS. API de Armazenamento na Web. **MDN Web Docs**, *ls. lJ*, 29 jul. 2024. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/API/Web_Storage_API. Acesso em: 14 jan. 2025.
- PINHO, D. M. **ECMAScript 6**: entre de cabeça no futuro do Javascript. São Paulo: Casa do Código, 2018.
- SEBESTA, R. W. **Conceitos de linguagens de programação**. 9. ed. Porto Alegre: Bookman, 2011.

CONFIRA SUAS RESPOSTAS

1. Alternativa C.

No ES6 e nas versões seguintes, surgiram recursos como *arrow functions*, desestruturação de objetos e *arrays*, além de operadores de espalhamento.

A alternativa A está incorreta, pois JQuery é uma versão antiga de Javascript criada pela Microsoft.

A alternativa B está incorreta, pois as definições de variáveis com "var" existem desde o início da linguagem.

A alternativa D está incorreta, pois as novas versões de Javascript se mantém compatíveis com as anteriores.

A alternativa E está incorreta, pois os navegadores não utilizam máquina virtual para interpretar Javascript.

2. Alternativa E.

A afirmativa I está correta, pois "filter" é usada para criar um *array* contendo apenas elementos que atendam a uma condição.

A afirmativa II está correta, pois "map" pode modificar cada elemento de um *array*, sem alterar o original.

A afirmativa III está correta, pois "reduce" é utilizada para combinar todos os elementos de um *array* em um único valor.

3. Alternativa B.

A forma correta é utilizar o método `.setItem()`

A alternativa A está incorreta, pois não existe "save" no `localStorage`.

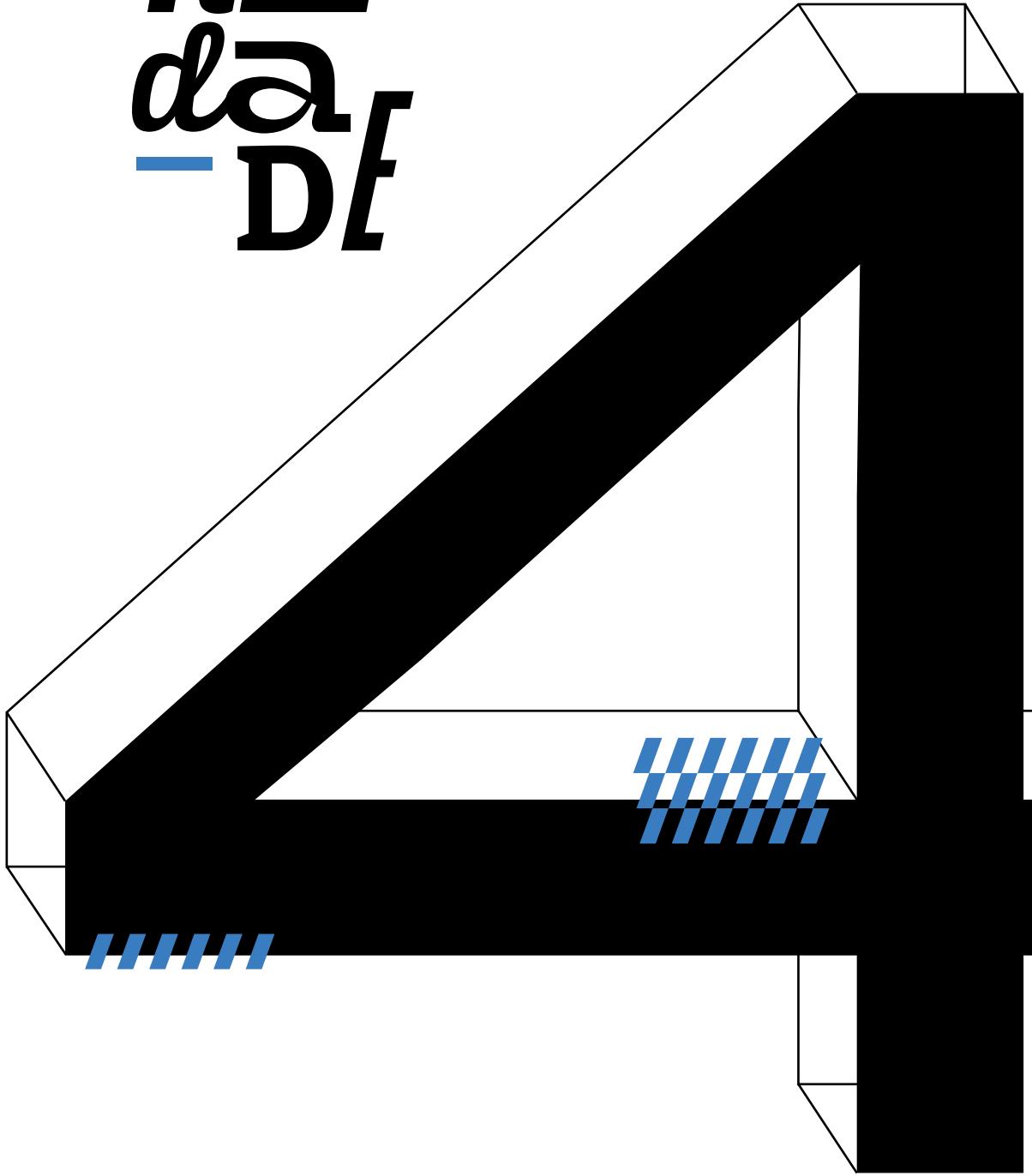
A alternativa C está incorreta, pois não existe "saveltem" no `localStorage`.

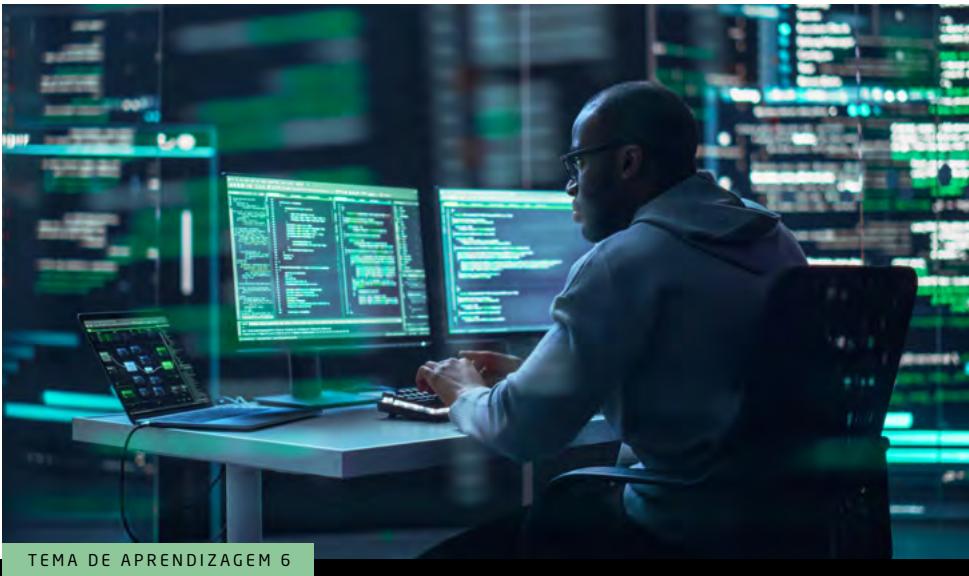
A alternativa D está incorreta, pois não existe "addltem" no `localStorage`.

A alternativa E está incorreta, pois não existe "store" no `localStorage`.



uni
da
- DF





TEMA DE APRENDIZAGEM 6

SISTEMAS QUE CONVERSAM ENTRE SI

MINHAS METAS

- Manipular dados em formato JSON.
- Programar de forma assíncrona.
- Dominar o conceito de promessas.
- Entender o uso das API.
- Consumir API públicas no sistema.
- Aprender a autenticação por chaves.
- Criar um sistema de filmes.

INICIE SUA JORNADA

Por que desenvolvemos software? Basicamente, para solucionar problemas de uma forma mais otimizada. Se um problema já está resolvido de forma satisfatória, não há motivos para criar novamente uma solução, podemos seguir adiante para os próximos problemas.

Imagine que sua empresa deseja criar um serviço online de streaming audiovisual, onde publicará os vídeos produzidos pela própria empresa. Você é a pessoa encarregada de desenvolver esse sistema, mas nunca lidou com vídeos. O que você faz?

Bem, primeiro, estude o padrão de arquivos de vídeos e como eles são interpretados pelos *codecs*. Em seguida, desenvolva um conversor a partir da documentação do *codec* e um player de vídeo. Não, espere. Você não precisa “redesenvolver” coisas que já possuem inúmeras soluções boas disponíveis.

O seu trabalho não é escrever a maior quantidade de códigos que puder ou desenvolver tudo desde o início. Seu trabalho é entregar uma solução funcional de streaming de vídeos para a empresa. Para isso, pode apoiar-se em *players* consagrados no mercado, em suporte do navegador, em bibliotecas disponíveis. Além disso, há a possibilidade de trocar informações entre sistemas por meio das API, que permitem reaproveitar conhecimento bem como tecnologias. Você pode, por exemplo, hospedar seus vídeos em alguma plataforma conhecida e trazer ao seu sistema, pelo acesso às API.

Assim como engenheiros não precisam reinventar a roda para desenvolver um carro, você não precisa reescrever soluções disponíveis. Já imaginou o tempo que demoraria para entregar um sistema, caso precisasse desenvolver tudo desde o princípio?

PLAY NO CONHECIMENTO

Em nosso podcast, tratamos de exemplos conhecidos de sistemas com as API públicas, as quais possibilitam integração e criação de soluções profissionais. Não deixe de ouvir, porque será uma boa inspiração para o seu desenvolvimento e seus estudos. **Recursos de mídia disponíveis no conteúdo digital do ambiente virtual de aprendizagem.**



VAMOS RECORDAR?

Para uma boa compreensão de arquivos JSON, é bom entender objetos em Javascript, assim como é recomendado saber manipulá-los. Se quiser entender bem a definição e o uso de objetos, aproveite o vídeo sobre o assunto, que apresenta como tudo é objeto em Javascript.

Acesse: <https://www.youtube.com/watch?v=n5uiJr-voKQ>

DESENVOLVA SEU POTENCIAL

Existe uma palavra mágica em desenvolvimento de software, que é “**reuso**”. Como disse o grande Abelardo Barbosa, o Chacrinha, “na TV nada se cria, tudo se copia” (Martins, 2017, on-line). Não apenas na TV, mas em diversas áreas, o conhecimento é reaproveitado e reciclado.

Em programação, desde o princípio, estudamos o reaproveitamento de código a partir de funções em nosso próprio código, depois, em bibliotecas as quais podemos adicionar em nosso sistema. Agora, ampliaremos um pouco mais essa noção, reaproveitando o desenvolvimento disponível em outros sistemas (ou outras partes de nossos sistemas).

No desenvolvimento web, é muito comum a troca de informações por meio das **API**, onde o sistema que roda no back-end fornece uma API, a qual pode ser “consumida” pelo sistema que roda no front-end, possibilitando a troca de informações nas duas direções. Isso é feito em um mesmo sistema ou entre sistemas diferentes que trocam informações.

Como o importante é trocar a informação de forma leve e sem se prender a uma interface específica, é comum essa troca de informações ser feita pelo mesmo padrão de codificação das informações, o que, geralmente, ocorre via arquivos no formato **JSON**, que é por onde iniciaremos.

TROCANDO INFORMAÇÕES POR MEIO DE JSON

Ao trocar informações via rede, nossa intenção é transmitir a menor quantidade de dados possível, por isso preferimos trafegar apenas os dados, sem penduricalhos, para otimizar a comunicação.

Pensando nisso, uma das formas mais otimizadas e utilizadas é por meio de um formato que prioriza apenas a informação em um conjunto de pares “chave-valor”. O formato **JSON** (*Javascript Object Notation*), como o nome diz, segue o padrão de objetos Javascript, que engloba pares de informações no formato “propriedade: valor”, com pequenas diferenças em relação a objetos utilizados nativamente nessa linguagem (MDN Contributors, 2024, on-line):

- JSON é puramente *string* (texto puro).
- Os valores de texto e os nomes de propriedades são sempre definidos entre aspas duplas.
- O JSON é, puramente, um formato de dados, contendo apenas propriedades, sem métodos.
- Uma vírgula ou dois-pontos em lugar errado podem fazer o JSON não funcionar.

Apesar das diferenças, assim como objetos nativos, o JSON pode incluir os mesmos tipos de dados básicos, por exemplo, *strings*, números, matrizes, booleanos e outros literais de objeto. Veja um exemplo:

```
{  
    "nome": "Clark Kent",  
    "cidade": "Metrópolis",  
    "habilidades": [  
        "Voo",  
        "Visão de Calor",  
        "Visão de raio-x",  
        "Superforça",  
        "Supervelocidade",  
        "Sopro congelante",  
        "Habilidade de se trocar dentro de uma cabine  
        telefônica"  
    ]  
}
```

Caso você queira, é possível utilizar alguma ferramenta para testar se o seu JSON é válido e se a sintaxe está correta, como é o caso da ferramenta JSON Lint (jsonlint.com).

Para trabalhar com JSON dentro de um programa em Javascript, podemos converter o JSON para objeto nativo, ou um objeto nativo para JSON, por meio das funções **JSON.parse()** e **JSON.stringify()**, respectivamente.

```
1 const meuJSON = '{"nome": "Clark Kent", "idade": 30}';  
2 const meuObjeto = JSON.parse(meuJSON);  
3 console.log(meuObjeto.nome); // Clark Kent  
4 console.log(meuObjeto.idade); // 30  
5 console.log(meuObjeto); // { nome: 'Clark Kent',  
  idade: 30 }  
6 console.log(JSON.stringify(meuObjeto)); //  
  {"nome":"Clark Kent","idade":30}
```

No código apresentado, iniciamos criando a variável “meuJSON”, que é um JSON contido em uma *string*. Para utilizar como objeto, na linha 2, criamos uma variável “meuObjeto” que converte o JSON para objeto, pela função **JSON.parse()**.

A partir da linha 3, exibimos alguns valores e temos, nos comentários, as saídas produzidas por cada linha. Na linha 3, ao exibir no console a propriedade `meuObjeto.nome`, recebemos o retorno “Clark Kent”. Na linha 4, exibimos `meuObjeto.idade`, que produz o retorno 30, do tipo numérico. Mesmo o JSON sendo uma *string*, na conversão, o objeto entende que o valor é do tipo numérico.

Na linha 5, exibimos o objeto, e na linha 6, exibimos o objeto convertido de volta para JSON. As informações são as mesmas, mas note que o retorno da linha 6 está estruturado dentro das regras do JSON, e a própria visualização no console mostra a diferença (Figura 1).

```

Clark Kent           script.js:3
30                  script.js:4
▼ {nome: 'Clark Kent', idade: 30} i   script.js:5
  idade: 30
  nome: "Clark Kent"
▶ [[Prototype]]: Object
{"nome":"Clark Kent","idade":30}         script.js:6

```

Figura 1 - Captura de tela do console Javascript / Fonte: o autor.

Descrição da Imagem: captura de tela que mostra o seguinte texto: Linha 1: Clark Kent script.js:3. Linha 2: 30 script.js:4. Linha 3: {nome: 'Clark Kent', idade: 30} script.js:5. Linha 4: idade: 30. Linha 5: nome: "Clark Kent", Linha 6: [[Prototype]]: Object. Linha 7: {"nome":"Clark Kent","idade":30} script.js:7. Fim da descrição.

Enquanto a linha 5 produz um objeto Javascript, a linha 6 produz uma *string*, que é nosso JSON.

CONSUMINDO AS API PÚBLICAS

Em linhas gerais, uma **API** (*Application Programming Interface*) é uma interface para programação que consiste em um conjunto de regras ou protocolos, para que os sistemas possam trocar informações entre eles (Goodwin, 2024, on-line).

Existem inúmeras API públicas disponíveis para você utilizar em projetos ou mesmo para estudar o funcionamento. Se você pesquisar por “API públicas” (ou pelo correspondente em inglês, “*public apis*”), encontrará diversos catálogos de API dos mais variados tipos (informações meteorológicas, bases de dados de filmes, catálogos de Pokémon).

Basicamente, o funcionamento consiste em enviar uma requisição para a API, receber a resposta e tratar essa resposta, mas, para isso, precisaremos aprender um pouco de comunicação assíncrona e como usá-la em programação.



Trabalhando de forma assíncrona

Em nossas relações interpessoais, estamos bem acostumados com a comunicação síncrona e assíncrona.

Quando estamos conversando diretamente com uma pessoa, em uma mesma sala, a conversa está sendo feita de forma **síncrona**, na qual uma pessoa fala, termina e espera a fala da outra pessoa, para poder falar novamente e assim por diante. A comunicação toda envolve uma mesma sessão e a espera de uma mensagem para enviar a próxima.

Quando conversamos com alguém por meio de um mensageiro, como o WhatsApp, por exemplo, não precisamos ficar esperando uma resposta. Podemos enviar uma ou várias mensagens em texto, ou áudio, em seguida, há a possibilidade de fechar ou mesmo enviar mensagens a outras pessoas. Quando chegar uma resposta, então daremos atenção a ela, para continuar a conversa. Essa é a essência de uma conversa **assíncrona**: enviamos as mensagens e estamos disponíveis para fazer qualquer outra coisa. Quando a mensagem de resposta chegar, decidimos o que fazer a seguir.

Em programação, ao solicitar uma informação a uma API, estamos fazendo uma requisição via rede. Existe todo o tempo de tráfego de ida da requisição, de vinda da resposta e, ainda, do tempo de processamento da API. Na prática, fazemos uma requisição e não sabemos quando chegará (e se chegará) a resposta. Por isso, precisamos de uma estratégia para trabalhar com requisições assíncronas.

Para isso, trabalharemos com **funções assíncronas**, ou seja, funções que retornam uma “promessa” (as famosas *promises* do Javascript). Uma **promise** é uma resposta que pode estar em um desses três estados:

PENDENTE (PENDING)

Um estado de espera, onde a requisição não foi cumprida nem rejeitada.

CUMPRIDA (FULFILLED)

Significa que a requisição foi concluída com sucesso, e o valor de resposta foi entregue.

REJEITADA (REJECTED)

Indica que a requisição falhou de alguma forma.

Dessa forma, não podemos executar uma requisição assíncrona e já trabalhar com o resultado, porque não temos o resultado de forma imediata, temos apenas uma promessa de resultado. Nesse ponto, entram as funções “**callback**”, que são funções escritas para serem executadas apenas quando a requisição for concluída. Assim, pedimos uma informação e indicamos: “Quando a informação chegar, execute essa função com os dados recebidos”. Veremos isso, na prática.



**EU INDICO**

Existe uma API muito útil e bastante simples que retorna os dados de qualquer CEP brasileiro. Acesse: <https://viacep.com.br/ws/87013000/json/>

Se você abrir esse endereço em seu navegador, verá os seguintes dados, em JSON:

```
{  
  "cep": "87013-000",  
  "logradouro": "Avenida Brasil",  
  "complemento": "de 2731/2732 a 4569/4570",  
  "unidade": "",  
  "bairro": "Zona 01",  
  "localidade": "Maringá",  
  "uf": "PR",  
  "estado": "Paraná",  
  "regiao": "Sul",  
  "ibge": "4115200",  
  "gia": "",  
  "ddd": "44",  
  "siafi": "7691"}  
}
```

A partir desses dados, é possível fazer sistemas que completam os dados de endereço de um formulário, de maneira bastante simples.

Antes, precisamos entender rapidamente como funcionam os métodos HTTP para transferência de dados.



Métodos HTTP

Toda requisição feita via HTTP possui um método, ele é o que indica o comportamento da requisição. Os métodos principais são:

GET

Utilizado para fazer requisições de informações. A resposta a esse método deve conter apenas dados.

POST

Empregado para enviar informações quando queremos criar um recurso.

PUT

Utilizado para alterar um recurso, faz a troca de seu conteúdo pelo conteúdo enviado.

PATCH

Empregado para alterar apenas parte de um recurso.

DELETE

Utilizado para remover um recurso.

Essa requisição que fizemos ao acessar o endereço pelo navegador é uma requisição GET, apenas para buscar informações. Quando preenchemos um formulário e enviamos informações, usamos o método POST.

Fazendo requisições via Javascript

Agora, podemos juntar isso tudo o que conversamos, para colocar a mão na massa e receber os dados da API.

Montaremos um formulário de cadastro de endereço:

```
<form action="">

    <input type="text" id="cep" placeholder="Digite o
    CEP" />

    <input type="text" id="logradouro"
    placeholder="Logradouro" />

    <input type="text" id="bairro" placeholder="Digite o
    bairro" />

    <input type="text" id="localidade"
    placeholder="Digite a cidade" />

    <input type="text" id="estado" placeholder="Digite o
    estado" />

    <button type="submit">Enviar</button>

</form>
```

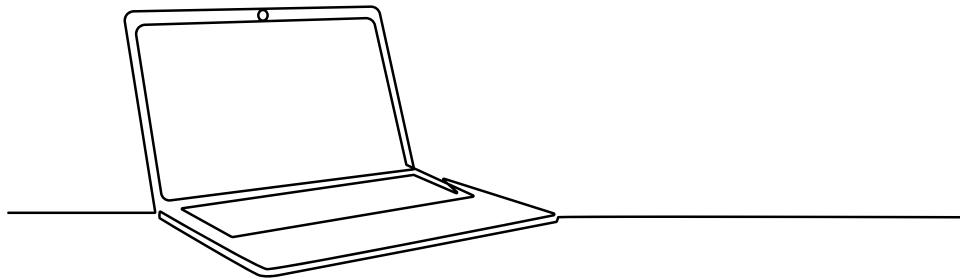
A partir desse formulário, escreveremos o nosso código. Como faremos uma requisição que nos retornará uma promessa (*promise*), temos duas abordagens possíveis. Podemos usar o método **.then()** da *promise*, que executa uma função no momento da conclusão da promessa, ou podemos usar as diretivas **async/await**, as quais indicam que a resposta deve ser esperada para executar a sequência. Veremos as duas formas:

```
1 const cep = document.getElementById("cep");
2 const logradouro = document.getElementById("logradouro");
3 const bairro = document.getElementById("bairro");
4 const localidade = document.getElementById("localidade");
5 const estado = document.getElementById("estado");
6
7 cep.addEventListener("blur", () => {
8     fetch(`https://viacep.com.br/ws/${cep.value}/json/`)
9         .then((response) => response.json())
10        .then((data) => {
11            logradouro.value = data.logradouro;
12            bairro.value = data.bairro;
13            localidade.value = data.localidade;
14            estado.value = data.uf;
15        });
16});
```

Nas linhas 1 a 6, apenas definimos variáveis para os elementos e, assim, facilitar a manipulação. Na linha 7, definimos uma função “call-back” para o evento *blur*, que será executada ao deixar o foco do campo “cep”. Nesse código, apenas para ficar mais sucinto, não verificamos se o CEP está digitado corretamente ou não. A API permite a digitação do CEP com ou sem o hífen.

Na linha 8, executamos a função “fetch”, que faz a requisição para dada URL. Esta foi passada com o valor de “cep.value”, ou seja, o CEP digitado no formulário.

Como a função “fetch” retorna uma *promise*, chamamos diretamente o método **.then()**, na linha 9, passando uma função **(response) => response.json()**, isto é, quando chegar a resposta da *promise*, será executado o método **.json()**, que extrai o JSON da resposta.



Este último método também retorna uma *promise*, por isso chamamos outro método `.then()`, na linha 10, o qual será executado quando o JSON estiver pronto. O retorno será armazenado no parâmetro “data”, então, basta colocar os valores recebidos nos demais campos do formulário, o que é feito das linhas 11 a 14. Visualmente, temos o formulário exibido na Figura 2, que recebe os valores após preencher o CEP e sair do campo.

Endereço

Figura 2 – Formulário simples para cadastro de endereço
Fonte: o autor.

Descrição da Imagem: formulário de cadastro com o título “Endereço”, logo baixo, há cinco campos de texto com os respectivos valores: “87010000”, “Avenida Cerro Azul”, “Zona 2”, “Maringá”, “PR” e um botão escrito “Enviar”. O segundo campo de texto está selecionado, pois apresenta borda azul. Fim da descrição.

Ao realizar as requisições assíncronas dessa forma, dependemos, por vezes, de um encadeamento de métodos THEN que pode ficar não muito elegante. Em versões mais recentes da linguagem, foi adicionado o par de diretivas **async/await**, as quais permitem a declaração de funções assíncronas (que retornam *promises*), com a possibilidade de esperar o processamento assíncrono ser executado, como se fosse síncrono (Baradel, 2024, on-line).

Refatorando nosso código, podemos realizar a mesma operação da seguinte forma:

```
1 const cep = document.getElementById("cep");
2 const logradouro=document.getElementById("logradouro");
3 const bairro = document.getElementById("bairro");
4 const localidade=document.getElementById("localidade");
5 const estado = document.getElementById("estado");
6
7 cep.addEventListener("blur", async () => {
8     const response = await fetch(`https://viacep.
com.br/ws/${cep.value}/json/`);
9     const data = await response.json();
10    logradouro.value = data.logradouro;
11    bairro.value = data.bairro;
12    localidade.value = data.localidade;
13    estado.value = data.uf;
14});
```

O código ficou bem parecido com o anterior. As definições das linhas 1 a 5 permaneceram iguais. Na linha 7, foi adicionada a palavra *async* antes da função, que define a função como assíncrona. Só podemos usar a diretiva *await* dentro de uma função se ela for assíncrona.

Nas linhas 8 e 9, fizemos as chamadas às funções assíncronas por meio da diretiva *await*, armazenando o retorno a variáveis. O *await* indica que assim que a promessa for cumprida ou rejeitada, o retorno será passado para as variáveis.

Depois disso, apenas colocamos os valores nos demais campos, da mesma forma que fizemos no código anterior. Pronto, você já sabe fazer requisições às APIs e tratar o retorno de forma assíncrona.

DESENVOLVENDO UM SISTEMA DE LISTAGEM DE FILMES

Existem diversas API públicas que podemos utilizar para desenvolver sistemas, algumas gratuitas, outras pagas, e ainda, outras parcialmente gratuitas.

Para desenvolver, aproveitaremos a listagem de API públicas disponível no GitHub, no repositório “public-apis” (<https://github.com/public-apis/public-apis>). Assim, você tem a chance de explorar as diversas API listadas ali.

Dentre as informações listadas, há a indicação se a API requer autenticação ou não, assim como o tipo de autenticação, onde há um dos seguintes valores:

NO
Não necessita de autenticação para usar, como fizemos com a consulta de CEP.
APIKEY
É necessário autenticar com uma chave de API gerada pelo sistema.
OAUTH
Um protocolo de autenticação em padrão aberto para autenticar de forma segura, sem revelar nenhuma senha (Sobers, 2022, on-line).

Utilizaremos uma API com autenticação via *API Key*, para colocar a parte de autenticação, porém de uma forma ainda mais simples.

Gerando sua chave de API

Uma chave de API é um valor único que identifica determinado usuário. Esse valor único não pode ser sequencial, nem ser facilmente descoberto, porque essa chave libera o acesso ao uso da API, ligando as ações ao usuário ao qual ela está ligada.

Não é o método mais seguro, nem é usado para todo tipo de aplicação. Em nosso caso, no qual apenas consumiremos os dados de uma API, cujos dados já estão disponíveis online, um vazamento de chave não traria muito risco. Para sistemas mais complexos e maior segurança, vale a pena estudar o funcionamento do *OAuth*.



EU INDICO

Quem gera a chave de API é sempre o sistema da API, portanto, as regras e exigências podem variar bastante. Utilizaremos a *API Open Movie Database*, que contém dados sobre filmes e séries em geral. Para isso, você precisa criar a sua chave de API pelo site. Acesse: <https://www.omdbapi.com/>

Clique em "API Key" no menu, preencha o cadastro com seus dados, escolhendo a versão "Free". Uma chave de API chegará em seu e-mail, com um link para ativação, basta clicar nele.

Após esse processo, você já pode usar a API para fazer consultas, por meio da sua chave.

Realizando consultas

Para esse sistema, utilizaremos apenas o método GET, do HTTP, o que facilita os testes, porque podemos testar todos os *endpoints*, diretamente pelo navegador.

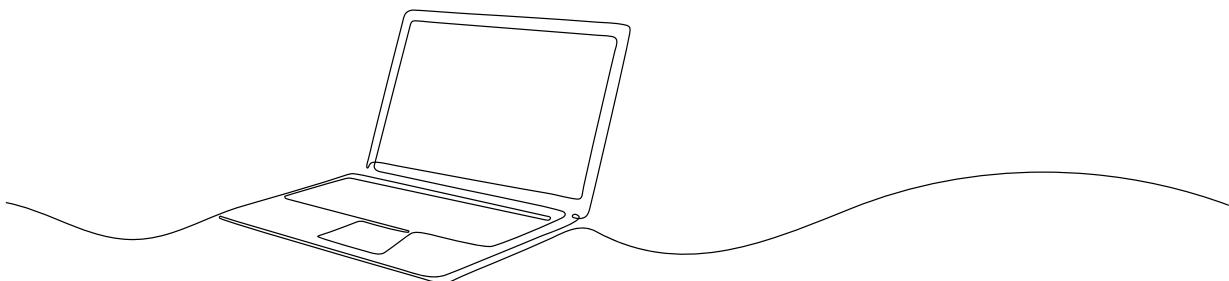
É importante sempre ler a documentação de uma API, para utilizá-la corretamente. Pela documentação da OMDb API, aprendemos a fazer as consultas e utilizar os parâmetros corretamente. As consultas são feitas pela seguinte URL:

`http://www.omdbapi.com/?apikey=[sua-chave] & [parâmetros]`

Podemos fazer a requisição pelo ID, ou título do filme, ou série, no qual a API retorna apenas um resultado, ou realizar uma busca, com vários resultados. Os parâmetros aceitos no primeiro caso são:

PARÂMETRO	OBRIGATÓRIO	VALORES VÁLIDOS	DESCRIÇÃO
i	Opcional (mas precisa ter i ou t).		Um ID do IMDb válido (ex. tt1285016).
t	Opcional (mas precisa ter i ou t).		O título a ser buscado.
type	Não	movie, series, episode	Tipo de resultado retornado.
y	Não		Ano de lançamento.
plot	Não	short, full	Retorna sinopse curta ou completa.
r	Não	json, xml	Tipo de dado a retornar.
callback	Não		Nome da callback JSONP.
v	Não		Versão da API (para uso futuro).

Quadro 1 - Parâmetros para requisição por ID ou título / Fonte: o autor.



Para uma busca com retorno de um conjunto de filmes ou séries, podemos usar os seguintes parâmetros:

PÁGINA 147 – Quadro 1 – Parâmetros para requisição por ID ou título

PARÂMETRO	OBRIGATÓRIO	VALORES VÁLIDOS	DESCRIÇÃO
i	Opcional (mas precisa ter i ou t).		Um ID do IMDb válido (ex. tt1285016).
t	Opcional (mas precisa ter i ou t).		O título a ser buscado.
type	Não	movie, series, episode	Tipo de resultado retornado.
y	Não		Ano de lançamento.
plot	Não	short, full	Retorna sinopse curta ou completa.
r	Não	json, xml	Tipo de dado a retornar.
callback	Não		Nome da callback JSONP.
v	Não		Versão da API (para uso futuro).

INÍCIO DESCRIÇÃO: A imagem “Quadro 1 – Parâmetros para requisição por ID ou título” exibe uma tabela de documentação de API que descreve vários parâmetros que podem ser usados em requisições.

A tabela tem quatro colunas: "PARÂMETRO", "OBRIGATÓRIO", "VALORES VÁLIDOS" e "DESCRIÇÃO".

Aqui está a descrição de cada linha da tabela:

PARÂMETRO: i

OBRIGATÓRIO: Opcional (mas precisa ter i ou t).

VALORES VÁLIDOS: (célula vazia)

DESCRIÇÃO: Um ID do IMDb válido (ex. tt1285016).

PARÂMETRO: t

OBRIGATÓRIO: Opcional (mas precisa ter i ou t).

VALORES VÁLIDOS: (célula vazia)

DESCRIÇÃO: O título a ser buscado.

PARÂMETRO: type

OBRIGATÓRIO: Não

VALORES VÁLIDOS: movie, series, episode

DESCRIÇÃO: Tipo de resultado retornado.

PARÂMETRO: y

OBRIGATÓRIO: Não

VALORES VÁLIDOS: (célula vazia)

DESCRIÇÃO: Ano de lançamento.

PARÂMETRO: plot

OBRIGATÓRIO: Não

VALORES VÁLIDOS: short, full

Descrição: Retorna sinopse curta ou completa.

PARÂMETRO: r

OBRIGATÓRIO: Não

VALORES VÁLIDOS: json, xml

Descrição: Tipo de dado a retornar.

PARÂMETRO: callback

OBRIGATÓRIO: Não

VALORES VÁLIDOS: (célula vazia)

Descrição: Nome da callback JSONP.

PARÂMETRO: v

OBRIGATÓRIO: Não

VALORES VÁLIDOS: (célula vazia)

Descrição: Versão da API (para uso futuro).

A tabela fornece uma referência clara para desenvolvedores sobre como construir requisições para a API, detalhando os parâmetros disponíveis, sua obrigatoriedade, valores aceitos e o propósito de cada um. **FIM DESCRIÇÃO**.

PARÂMETRO	OBRIGATÓRIO	VALORES VÁLIDOS	DESCRIÇÃO
s	Sim		Título a ser buscado.
type	Não	movie, series, episode	Tipo de resultado a retornar.
y	Não		Ano de lançamento.
r	Não	json, xml	Tipo de dado a retornar.
page	Não	1-100	Número da página a retornar (só retornados apenas dez resultados por página).
callback	Não		Nome da callback JSONP.
v	Não		Versão da API (para uso futuro).

Quadro 2 - Parâmetros para requisições de busca / Fonte: o autor.

Você pode testar, diretamente do navegador, executando uma busca por:

```
https://www.omdbapi.com/?apikey=[sua-chave]&s=friends
```

Verá um JSON com uma lista de resultados que combinem com a palavra “friends” (se usar uma chave de API válida).

Criando nossa interface

Para nosso sistema, utilizaremos um HTML simples, com a estrutura básica e uma tag <main>, onde colocaremos o nosso conteúdo via Javascript. Basicamente, listaremos alguns filmes, os exibiremos e os formataremos de uma forma “bonitinha”. Você pode ver o resultado nesse repositório do GitHub: <https://github.com/andre-noel/filmesflix>

PÁGINA 148 – Quadro 2 – Parâmetros para requisições de busca

PARÂMETRO	OBRIGATÓRIO	VALORES VÁLIDOS	DESCRIÇÃO
s	Sim		Título a ser buscado.
type	Não	movie, series, episode	Tipo de resultado a retornar.
y	Não		Ano de lançamento.
r	Não	json, xml	Tipo de dado a retornar.
page	Não	1-100	Número da página a retornar (só retornados apenas dez resultados por página).
callback	Não		Nome da callback JSONP.
v	Não		Versão da API (para uso futuro).

INÍCIO DESCRIÇÃO: A imagem “Quadro 2 – Parâmetros para requisições de busca” apresenta uma tabela que faz parte de uma documentação detalhando os parâmetros que podem ser utilizados ao realizar uma busca na API.

A tabela é composta por quatro colunas: "PARÂMETRO", "OBRIGATÓRIO", "VALORES VÁLIDOS" e "DESCRIÇÃO".

Aqui está a descrição de cada parâmetro na tabela:

PARÂMETRO: s

OBRIGATÓRIO: Sim

VALORES VÁLIDOS: (célula vazia)

DESCRIÇÃO: Título a ser buscado.

PARÂMETRO: type

OBRIGATÓRIO: Não

VALORES VÁLIDOS: movie, series, episode

DESCRIÇÃO: Tipo de resultado a retornar.

PARÂMETRO: y

OBRIGATÓRIO: Não

VALORES VÁLIDOS: (célula vazia)

DESCRIÇÃO: Ano de lançamento.

PARÂMETRO: r

OBRIGATÓRIO: Não

VALORES VÁLIDOS: json, xml

DESCRIÇÃO: Tipo de dado a retornar.

PARÂMETRO: page

OBRIGATÓRIO: Não

VALORES VÁLIDOS: 1-100

DESCRIÇÃO: Número da página a retornar (são retornados apenas dez resultados por página).

PARÂMETRO: callback

OBRIGATÓRIO: Não

VALORES VÁLIDOS: (célula vazia)

DESCRIÇÃO: Nome da callback JSONP.

PARÂMETRO: v

OBRIGATÓRIO: Não

VALORES VÁLIDOS: (célula vazia)

DESCRIÇÃO: Versão da API (para uso futuro).

A tabela fornece um guia para os usuários da API sobre como formular suas requisições de busca, especificando quais parâmetros são obrigatórios, quais valores são aceitos para cada um e o propósito de cada parâmetro. **FIM DESCRIÇÃO.**

Começaremos, no arquivo script.js, definindo a chave de API e a URL de acesso. Em um sistema em produção, não deixaríamos a chave direto no código, mas protegeríamos em variáveis de ambientes ou em arquivos não acessíveis por outros.

```
const APIKEY = "SUA-CHAVE-DE-API";  
  
const url = `https://www.omdbapi.com/?apikey=${APIKEY}`;
```

Como vimos, a forma de requisitar uma informação é sempre muito parecida, então, podemos criar funções com o que desejamos fazer. Buscaremos os filmes e os detalhes deles na API.

```
const buscaTitulos = async (titulo) => {  
  
    const response = await  
fetch(` ${url}&s=${titulo}&type=movie`);  
  
    const data = await response.json();  
  
    return data.Search;  
  
};  
  
const buscaDetalhes = async (id) => {  
  
    const response = await fetch(` ${url}&i=${id}`);  
  
    const data = await response.json();  
  
    return data;  
  
};
```

Depois, criaremos a parte visual, definindo *cards* que conterão as informações dos filmes. Basicamente, a partir do JSON retornado, apenas preencheremos as informações, mas os cards serão iguais.

```
const criaCard = (filme) => {

    const card = document.createElement("div");
    card.classList.add("card");
    card.innerHTML = `

        
        <h3>${filme.Title}</h3>
        <p>${filme.Year}</p>
        <button onclick="mostraDetalhes('${filme.imdbID}')">Detalhes</button>
    `;

    return card;
};

const mostraDetalhes = async (id) => {

    const detalhes = await buscaDetalhes(id);
    const modal = document.createElement("div");
    modal.classList.add("modal");
    modal.innerHTML = `

        <div class="modal-content">
            <h3>${detalhes.Title}</h3>
            <p>${detalhes.Plot}</p>
            <p>Diretor: ${detalhes.Director}</p>
            <p>Elenco: ${detalhes.Actors}</p>
            <button onclick="fechaDetalhes()"
                    >Fechar</button>
        </div>
    `;
}
```

```
modal.addEventListener("click", fechaDetalhes);  
document.body.appendChild(modal);  
};  
  
const fechaDetalhes = () => {  
  document.querySelector(".modal").remove();  
};
```

Veja que, além do *card*, criamos até um modal, uma janelinha cujas informações aparecem ao clicarmos no botão “Detalhes” e que fecha ao clicarmos em qualquer área do modal.

Agora, basta colocar as funções para rodar. No exemplo do repositório, há quatro categorias listadas num looping. Aqui, serão apresentadas apenas uma, pela simplicidade.

```
const mainArea = document.querySelector("main");  
const h2 = document.createElement("h2");  
h2.innerHTML ="Filmes do Batman;  
mainArea.appendChild(h2);  
  
const cards = document.createElement("div");  
cards.classList.add("cards");  
mainArea.appendChild(cards);  
  
const titulos = await buscaTitulos("batman");  
titulos.forEach((filme) => {  
  cards.appendChild(criaCard(filme));  
});
```

Pronto, agora o nosso sistema lista dez filmes do Batman, retornados pela API. Basta capricharmos no CSS, e o céu é o limite. Podemos também listar mais categorias, incluir paginação, carregamento dinâmico etc.

A partir daqui, será possível consumir informações de API facilmente, seja do sistema back-end, seja de uma API de terceiros, como essa que usamos.

 **EM FOCO**

Assista à videoaula para mais informações e conhecimento sobre este tema de aprendizagem. Recursos de mídia disponíveis no conteúdo digital do ambiente virtual de aprendizagem.

NOVOS DESAFIOS

Boa parte do desenvolvimento web moderno apoia-se na comunicação entre front-end e back-end por meio das API, sendo importante que todo profissional dessa área entenda o funcionamento de API, para que a comunicação seja feita de forma otimizada e em um padrão.

Você pode criar interfaces variadas a partir do consumo de API, o que facilita a criação de diferentes interfaces para diferentes dispositivos. Dependendo da aplicação, a interação com o usuário via smartphones funciona de forma bem distinta do uso pelo computador, pela limitação de tamanho ou pelos recursos do dispositivo. Você pode criar diferentes interfaces que se comunicam de formas diferentes com a mesma API.

Um sistema, ainda, tem a possibilidade de demandar interfaces diversos para diferentes objetivos. Em um restaurante, por exemplo, um mesmo sistema deve estar integrado a todas as partes, porém a pessoa do caixa precisa de uma interface detalhada, com informações de todas as mesas e todos os pedidos, valores e detalhes, provavelmente, para uma tela de computador, enquanto os garçons precisam apenas de uma interface leve e rápida para anotar os pedidos em smartphones. Tudo com comunicação via API.

Aproveite as API públicas disponíveis, crie suas aplicações. Tem muita coisa divertida e interessante que pode ser feita.



VAMOS PRATICAR

1. "Javascript Object Notation (JSON) é um formato baseado em texto padrão para representar dados estruturados com base na sintaxe do objeto Javascript. É comumente usado para transmitir dados em aplicativos da Web (por exemplo, enviar alguns dados do servidor para o cliente, para que possam ser exibidos em uma página da Web ou vice-versa)" (MDN Contributors, 2024, on-line).

Considerando o texto, qual estrutura é utilizada para representar dados em JSON?

- a) Estrutura binária.
 - b) Array associativo.
 - c) Pares chave-valor.
 - d) Tabelas relacionais.
 - e) Vetores encadeados.
2. "Mesmo que se assemelhe à sintaxe literal do objeto Javascript, o JSON pode ser usado independentemente do Javascript, e muitos ambientes de programação possuem a capacidade de ler (analisar) e gerar JSON" (MDN Contributors, 2024, on-line).

A respeito do formato JSON, avalie as afirmações, a seguir:

- I - JSON é um formato de dados baseado em texto.
- II - JSON pode armazenar funções Javascript diretamente.
- III - JSON é frequentemente utilizado em comunicação cliente-servidor.

É correto o que se afirma em:

- a) I, apenas.
- b) III, apenas.
- c) I e III, apenas.
- d) II e III, apenas.
- e) I, II e III.

VAMOS PRATICAR

3. O protocolo HTTP (*HyperText Transfer Protocol* - Protocolo de Transferência de Hipertexto) data de 1996, época que os trabalhos conjuntos de Tim Berners-Lee, Roy Fielding e Henrik Frystyk Nielsen levaram à publicação de uma RFC (*Request for Comments*) descrevendo esse protocolo. Trata-se de um protocolo de camada de aplicação (segundo o modelo OSI) e, portanto, de relativa facilidade de manipulação em aplicações (Saudate, 2014).

Ao consumir uma API no front-end utilizando Javascript, qual método é frequentemente utilizado para fazer uma requisição HTTP assíncrona?

- a) map().
- b) fetch().
- c) parseJSON().
- d) XMLHttpRequest.
- e) JSON.stringify().

REFERÊNCIAS

BARADEL, L. Async/await: funções assíncronas no Javascript! **Medium**, [s. l.], 26 jan. 2024. Disponível em: <https://lucasbaradel.medium.com/async-await-fun%C3%A7%C3%B5es-ass%C3%ADncronas-no-Javascript-85768f3460fa>. Acesso em: 30 jan. 2025.

GOODWIN, M. O que é uma API (Interface de Programação de Aplicativos)? **IBM**, [s. l.], 9 abr. 2024. Disponível em: <https://www.ibm.com/br-pt/topics/api>. Acesso em: 30 jan. 2025.

MARTINS, G. Relembramos a história do apresentador Chacrinha, tema de minissérie da Globo. **O Globo**, Rio de Janeiro, 27 jan. 2017. Disponível em: <https://acervo.oglobo.globo.com/em-destaque/relembramos-historia-do-apresentador-chacrinha-tema-de-minisserie-da-globo-20834766>. Acesso em: 30 jan. 2025.

MDN CONTRIBUTORS. Trabalhando com JSON - Aprendendo desenvolvimento web. **MDN Web Docs**, [s. l.], 27 jul. 2024. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/Objects/JSON>. Acesso em: 30 jan. 2025.

SAUDATE, A. **REST**: Construa APIs inteligentes de maneira simples. São Paulo: Casa do Código, 2014.

SOBERS, R. O que é o OAuth? Definição e como funciona. **Varonis**, [s. l.], 4 mar. 2022. Disponível em: <https://www.varonis.com/pt-br/blog/what-is-oauth>. Acesso em: 30 jan. 2025.

CONFIRA SUAS RESPOSTAS

1. Alternativa C.

O JSON representa dados por meio de pares chave-valor, onde cada chave é uma *string* e cada valor pode ser um número, *string*, booleano, *array*, outro objeto JSON ou nulo.

A alternativa A está incorreta, pois não é utilizada uma estrutura binária.

A alternativa B está incorreta, pois, apesar de poder ter *arrays*, o JSON não utiliza um *array* associativo para ser representado.

A alternativa D está incorreta, pois sistemas de bancos de dados relacionais utilizam tabelas relacionais.

A alternativa E está incorreta, pois JSON não utiliza vetores encadeados.

2. Alternativa C.

A afirmação I está correta, pois JSON é um formato de dados baseado em texto.

A afirmação II está incorreta, pois JSON não pode armazenar funções Javascript diretamente, armazena apenas texto.

A afirmação III está correta, pois JSON é frequentemente utilizado em comunicação cliente-servidor.

3. Alternativa B.

O método `fetch()` é amplamente utilizado para realizar requisições HTTP assíncronas no Javascript, permitindo acessar as API e manipular dados no formato JSON.

A alternativa A está incorreta, pois o método `map` é utilizado para fazer iterações em itens de *arrays*.

A alternativa C está incorreta, pois o método `JSON.parse()` (e não `parseJSON`) é feito para transformar os dados de um JSON em um objeto Javascript.

A alternativa D está incorreta, pois, antigamente, utilizava-se o método `XMLHttpRequest` (e não `XMLRequest`) para fazer requisições AJAX.

A alternativa E está incorreta, pois o método `JSON.stringify()` é utilizado para transformar um objeto Javascript em uma *string* JSON.



ACESSIBILIDADE PARA TODOS

MINHAS METAS

- Entender os conceitos de acessibilidade.
- Conhecer as diretrizes WCAG.
- Utilizar ferramentas de testes de acessibilidade.
- Aplicar tecnologias assistivas.
- Explorar formas de desenvolver design para acessibilidade.
- Entender como criar conteúdo acessível.
- Garantir uma estrutura correta em documentos web.



INICIE SUA JORNADA

Sendo uma pessoa que desenvolve para a web, seu objetivo é fazer um sistema que tenha usuários, senão o seu trabalho todo é realizado em vão. Mas quantos deles você deseja ter? Independentemente do número, geralmente, quanto mais usuários, melhor.

Você, porém, desenvolveu um sistema e descobre, após um tempo da disponibilização, que ele está com uma baixa taxa de retenção, ou seja, as pessoas entram, cadastram-se, mas uma boa parte não segue utilizando e, então, abandona o seu sistema.

Após levantamento e cruzamento de dados, você percebe que o sistema não está com boa usabilidade, e isso não é apenas uma questão de aparência, as pessoas estão com dificuldade de utilizar, principalmente pessoas com baixa ou nenhuma visão, pessoas que usam dispositivos mais simples, de qualidade inferior, e pessoas com dificuldade em selecionar as opções na tela.

Você percebe, então, que o seu sistema não está realmente disponível para todo mundo. Colocou, não intencionalmente, barreiras que impedem as pessoas de terem acesso a todos os recursos de sua aplicação.

Será possível criar, de fato, um sistema acessível a todos? As tecnologias atuais permitem desenvolver um sistema que tenha essa característica?



PLAY NO CONHECIMENTO

Ouça o podcast e fique sabendo como grandes empresas têm aplicado recursos de acessibilidade em seus sistemas para alcançar todas as pessoas e melhorar cada vez mais a usabilidade entre seus usuários. **Recursos de mídia disponíveis no conteúdo digital do ambiente virtual de aprendizagem.**

VAMOS RECORDAR?

Desenvolver para web, com acessibilidade, começa pelo desenvolvimento correto das páginas. A estruturação de um documento HTML com uma semântica apropriada ajuda muito a acessibilidade. Por isso, antes de seguir, confira esse conteúdo de como utilizar os recursos do HTML 5 para uma boa semântica em um site: https://www.youtube.com/watch?v=IHM0u_KEW-o

DESENVOLVA SEU POTENCIAL

ACESSIBILIDADE É PARA VOCÊ (E PARA TODOS)

Um dos principais desafios ao escrever sobre acessibilidade é segurar a atenção das pessoas para ler o conteúdo até o final. A princípio, existe uma noção errônea de que desenvolver sistemas acessíveis demanda desperdício de esforço, dado que eles “funcionam” da forma como estão e não há percepção de que estejam afetando usuários atuais.

Você, que está lendo este material, porém, não pense assim. Um sistema acessível a todos não significa um sistema “adaptado” para atender a algumas necessidades específicas de grupos especiais, mas sim um sistema cujos dados estejam ao acesso de todos e a usabilidade não exclua nenhuma pessoa, seja por alguma deficiência, seja por alguma dificuldade momentânea.

Pensar em acessibilidade não significa apenas pensar em pessoas com deficiências, mas em todas as dificuldades de acesso, como explica Ferraz (2020, p. 13):



Por exemplo, garantir um bom contraste entre texto e o fundo permite que pessoas com baixa visão consigam ler um texto com mais facilidade, mas permite também que pessoas que utilizam o celular na rua, sob incidência de sol na tela, tenham menos dificuldade em ler as informações na tela do celular.

Desde o princípio, o desenvolvimento da web teve por objetivo conectar as pessoas e as informações, de forma a chegar em todo o mundo (W3C Brasil, 2014). Com o tempo, o hardware e o software foram se desenvolvendo para tornar mais viável o desenvolvimento com acessibilidade, sendo que, hoje, há diversas ferramentas à mão, para que o desenvolvimento acessível não seja tão dispendioso e torne o acesso mais democrático.

Neste material, conversaremos sobre como tem se dado o desenvolvimento acessível para a web, abordando diretrizes, recomendações e ferramentas para que o seu sistema, aluno, tenha a melhor experiência de usabilidade por todas as pessoas.

Segundo Ferraz (2020), diferentes condições ou situações podem se beneficiar da acessibilidade:

DEFICIÊNCIA VISUAL

Existem diferentes níveis de deficiência visual, os quais abrangem cegueira, baixa visão ou daltonismo. Pessoas que não conseguem ler a informação ou ver as imagens; precisam de tecnologias assistivas como leitores de telas para ter acesso à informação; com baixa visão, cuja necessidade é a leitura por meio de letras maiores e alto contraste e pessoas com daltonismo não podem depender de uma cor para entender a informação.

DEFICIÊNCIA AUDITIVA

Englobam casos de surdez ou baixa audição, cujas informações em áudio diretamente ou áudio em vídeos precisam de uma forma alternativa para transmitir a informação.

DEFICIÊNCIA MOTORA

Afeta pessoas que não conseguem movimentar os membros e não podem depender de toques na tela ou movimentos de mouse.

DEFICIÊNCIA COGNITIVA OU NEUROLÓGICA

Envolve diferentes transtornos cognitivos ou até doenças mentais, afetando a forma como as pessoas compreendem as informações.

PESSOAS IDOSAS

Com o avanço da idade, a habilidade com o mouse, o toque e até mesmo a acuidade visual, auditiva e cognitiva podem ser reduzidas.

DEFICIÊNCIA TEMPORÁRIA

Após cirurgias, acidentes ou traumas, a capacidade de acessar as informações corre o risco de ser reduzida de forma temporária ou permanente.

USUÁRIOS DE DISPOSITIVOS MÓVEIS

Ao utilizar dispositivos móveis, a acessibilidade costuma ser reduzida pelo tamanho da tela, pelo modo de interação ou até mesmo pela incidência de luz solar.

AMBIENTES SILENCIOSOS

Por vezes, é necessário acompanhar uma informação, como um vídeo, por meio de um dispositivo a partir do qual não é recomendado ou permitido emitir sons, dependendo, assim, de uma forma alternativa de transmitir a informação, por exemplo, de legendas.

Portanto, apesar de, ao ouvir sobre acessibilidade, ser comum pensar em uma deficiência, é importante ter em mente que a acessibilidade tem a ver com o acesso disponível a todas as pessoas.

DIRETRIZES DE ACESSIBILIDADE

Você já se perguntou de onde vem tudo isso que usamos? Quem define como um navegador deve entender os sites? De onde vêm as linguagens que usamos para o desenvolvimento?

O **W3C** (*World Wide Web Consortium*) é um consórcio internacional de empresas que define os padrões e recomendações da web. Dentro desses padrões, estão a forma como informações são descritas e exibidas para os usuários.

Em 1997, o W3C lançou o **WAI** (*Web Accessibility Initiative*), um esforço do W3C para aumentar a acessibilidade da web. Por meio do WAI, várias diretrizes para acessibilidade foram publicadas. Elas guiam o desenvolvimento para possibilitar o acesso da informação a todas as pessoas.

As diretrizes WCAG

Para guiar o desenvolvimento de forma que o produto seja acessível, foram definidas as diretrizes **WCAG** (*Web Content Accessibility Guidelines*), traduzida como Diretrizes de Acessibilidade para Conteúdo Web. Sua versão 2.2 foi publicada em 2023 (Web [...], 2023, on-line) e abrange diversas recomendações para tornar o conteúdo mais acessível (MDN Contributors, 2023, on-line).

Essas diretrizes guiam-se por quatro princípios que norteiam a acessibilidade na web:

PERCEPTÍVEL

As informações devem ser perceptíveis aos usuários, por meio de um ou mais de seus sentidos.

OPERÁVEL

Todos os componentes da interface precisam estar aptos a serem controlados de alguma forma, independentemente do dispositivo utilizado.

COMPREENSÍVEL

O conteúdo deve ser compreensível para os usuários.

ROBUSTO

O conteúdo precisa ser desenvolvido por padrões web amplamente adotados que funcionarão em diferentes navegadores, atuais e futuros.

Assim, as diretrizes, agrupadas por temas, guiam os objetivos básicos a serem atingidos, tornando mais nítido o caminho a ser seguido (Ferraz, 2020).

Como uma forma de mensurar a acessibilidade, as diretrizes também apontam **critérios de sucesso**, com descrições diretas, as quais detalham o que seria obter sucesso em cada diretriz. Esses critérios são definidos como:

NÍVEL A

Menor nível de conformidade, o que indica que as barreiras básicas de acesso foram eliminadas, possibilitando, assim, o acesso de pessoas com certos tipos de deficiência.

NÍVEL AA

As informações atingem um grupo maior de pessoas, eliminando mais barreiras de acesso, o que satisfaz também o nível A.

NÍVEL AAA

Possibilita o acesso a um grupo ainda maior, satisfazendo os níveis anteriores, porém esse nível é mais difícil de ser atingido.

Em linhas gerais, não se recomenda buscar o nível AAA como política para sites completos, mas para algumas áreas específicas.

WAI-ARIA

Focando na usabilidade de sistemas, a documentação **WAI-ARIA** (*Accessible Rich Internet Applications*) foi desenvolvida para lidar com aspectos dinâmicos de uma aplicação (Ferraz, 2020).

A utilização de modificações de conteúdo de forma dinâmica em um site, por meio de Javascript e requisições assíncronas, com frequência, altera a semântica de um sistema, ou seja, às vezes, um elemento HTML é utilizado para uma função diferente do que foi criado, por exemplo, uma *tag* “span” comportar-se como um botão.

Para lidar com essas situações, a documentação WAI-ARIA adicionou atributos especiais aos elementos HTML, que definem papéis (*role*), estados (*state*) e propriedades (*properties*).

Ao definir um elemento HTML, podemos indicar a função do elemento a partir da propriedade “**role**”. Note que a utilização de uma boa semântica, principalmente com elementos semânticos introduzidos pelo HTML5, deve já suprir essa necessidade, quando os elementos estão dentro de uma semântica correta. Em casos nos quais os elementos comportam-se de forma diferente à semântica original, o atributo “role” pode auxiliar muito na acessibilidade.

Existem diversos papéis predefinidos para serem utilizados com o atributo “**role**”, por isso vale a pena buscar a documentação. Aqui, listaremos apenas alguns principais, para ilustrar.

ATRIBUTO “ALERT”

Indica uma mensagem importante que deve ser imediatamente anunciada pelo leitor de tela.

ATRIBUTO “BANNER”

Define a área principal de cabeçalho de uma página (geralmente, inclui o logotipo e o título).

ATRIBUTO “BUTTON”

Define um elemento como um botão interativo.

ATRIBUTO “DIALOG”

Define uma caixa de diálogo ou modal que requer interação do usuário.

ATRIBUTO “MAIN”

Define a seção principal de conteúdo da página.

ATRIBUTO “NAVIGATION”

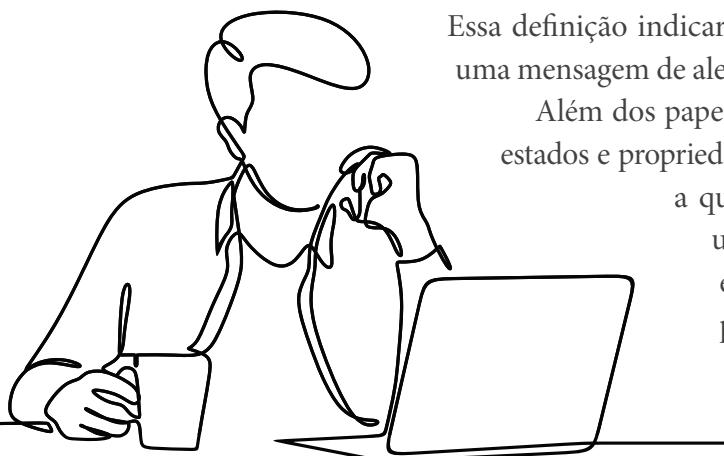
Define uma região de navegação, geralmente, usada para menus ou links principais.

ATRIBUTO “PROGRESSBAR”

Define um indicador de progresso, como uma barra de carregamento.

Dessa forma, caso seu sistema exiba uma mensagem que necessita de um destaque para o usuário, ele pode definir da seguinte forma:

```
<div role="alert">Erro ao salvar os dados!</div>
```



Essa definição indicará ao leitor de telas que é uma mensagem de alerta.

Além dos papeis, é possível ter, também, estados e propriedades a serem transmitidos a qualquer momento para o usuário, em tempo real ou em momento predefinido pelo autor (Ferraz, 2020).

Alguns dos principais atributos para definir estados e propriedades são:

ARIA-LABEL

Define um rótulo acessível para um elemento, útil quando não há texto visível associado ao elemento (como pode acontecer com ícones ou botões sem texto).

ARIA-LABELLEDBY

Define um ID de outro elemento que serve como rótulo acessível.

ARIA-DESCRIBEDBY

Define um ID de outro elemento que fornece uma descrição acessível adicional.

ARIA-HIDDEN

Esconde o elemento das tecnologias assistivas, como leitores de tela.

ARIA-EXPANDED

Indica se um elemento interativo (como um menu suspenso) está expandido (*true*) ou recolhido (*false*).

ARIA-CHECKED

Indica o estado de seleção de um controle, como um *checkbox* (*true*, *false* ou *mixed*).

ARIA-DISABLED

Indica que um elemento interativo está desativado e não pode ser usado (*true* ou *false*).

ARIA-LIVE

Indica que o conteúdo de uma região será atualizado dinamicamente e deve ser anunciado pelo leitor de tela, podendo ser de forma *polite* ("educada", espera a pausa do usuário) ou *assertive* (imediata).

Por exemplo, podemos ter o seguinte bloco:

```
<div aria-live="polite">  
    Carregando, por favor aguarde...  
</div>
```

Assim, por mais dinâmico que seja o sistema, os leitores de tela e demais assistentes podem ter mais controle sobre a informação, repassando aos usuários detalhes que facilitarão a assimilação e o controle do conteúdo.



FERRAMENTAS PARA TESTES DE ACESSIBILIDADE

Não é fácil manter a acessibilidade de um site e ter, apenas por intuição ou análise visual, a noção do quanto acessível seu sistema está. Por isso, existem ferramentas que auxiliam muito no teste de acessibilidade, indicando onde a sua informação deixa de estar acessível e como você pode melhorar esse aspecto.

Em geral, as ferramentas indicam necessidades de melhorias em contraste, legibilidade, navegação por teclado, descrição de imagens e outros aspectos técnicos, porém é interessante utilizar a experiência de especialistas em acessibilidade para identificar pontos de melhorias em um sistema (Gala, 2023, on-line).

Dentre as ferramentas mais comuns para testar a acessibilidade em sites, podemos destacar:

WAVE (WEB ACCESSIBILITY EVALUATION TOOL)

Uma ferramenta online que fornece relatórios detalhados sobre a acessibilidade de uma página, destacando áreas problemáticas e sugestões para melhorias.

AXE ACCESSIBILITY

Uma extensão de navegador e ferramenta de linha de comando que, após escanear páginas web, identifica problemas de acessibilidade em cada uma.

GOOGLE Lighthouse

Ferramenta de código aberto que automatiza testes de desempenho, SEO e acessibilidade. Ela gera relatórios detalhados, incluindo pontuações de acessibilidade. Existe a opção de instalar, mas já vem embutida no Google Chrome.

Também é recomendada a utilização de ferramentas assistivas, como leitores de telas, para simular a experiência de usuários que necessitam de auxílio, para um teste de usabilidade mais abrangente.

WAVE (Web Accessibility Evaluation Tool)

Para testar uma página com o WAVE, acesse o site <https://wave.webaim.org/>. Em seguida, digite no campo *Web page address* o endereço da página que será analisada. Depois de enviar, será exibido um relatório, após alguns segundos, como aparece no exemplo da Figura 1.

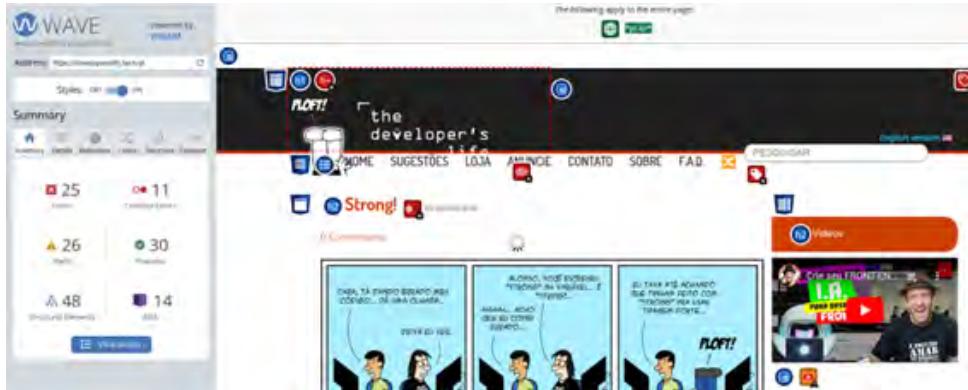


Figura 1 – Teste de acessibilidade em uma página web pelo WAVE / Fonte: o autor.

Descrição da Imagem: captura de tela da interface da ferramenta WAVE (*Web Accessibility Evaluation Tool*), utilizada para avaliar a acessibilidade de páginas web. Na barra lateral esquerda, há um painel de resumo com estatísticas sobre a análise da página, exibindo os seguintes dados: 25 erros, 11 erros de contraste, 26 alertas, 30 funcionalidades identificadas, 48 elementos estruturais e 14 elementos ARIA. No centro, está uma captura de tela do site analisado. O cabeçalho do site contém o logotipo e um menu de navegação com seções como Home, Sugestões, Loja, Anuncie, Contato, Sobre e F.A.Q. Há, também, uma barra de pesquisa à direita. Na análise, elementos específicos são destacados por ícones de erro (em vermelho), alertas (em amarelo) e boas práticas (em azul). Fim da descrição.

A partir da análise, é possível identificar erros e alertas, mostrados pela ferramenta na exibição do site e no relatório mais detalhado. A ferramenta indica o erro e fornece informações de como corrigi-lo.

No exemplo da Figura 1, um dos erros apontados foi o baixo contraste presente na data do artigo, que está escrita em cinza-claro sobre um fundo branco. Pela própria ferramenta, é possível testar diferentes cores para identificar se cumprem os níveis AA e AAA de recomendação.

Google Lighthouse

Não apenas para testar acessibilidade, mas para realizar um teste geral em desempenho e usabilidade do site, o Google Lighthouse fornece boas informações de como melhorar uma página web.

Existem formas diferentes de acessar a ferramenta. Analisaremos o uso a partir do Google Chrome, pela praticidade. No navegador, abra as ferramentas para

desenvolvedor (*Chrome DevTools*, aquela guia que abrimos quando clicamos no menu “Inspecionar” ou a partir da tecla F12). Então, acesse a aba Lighthouse, onde você pode solicitar a análise do site, que será exibida como aparece na Figura 2.

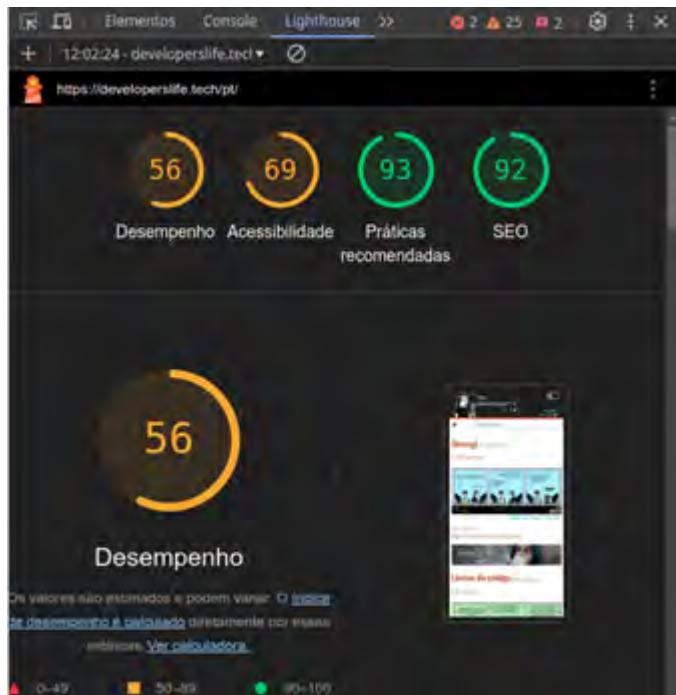


Figura 2 - Relatório de desempenho de um site pelo Google Lighthouse / Fonte: o autor.

Descrição da Imagem: captura de tela que apresenta os resultados de uma análise de desempenho realizada pela ferramenta Google Lighthouse, a qual retornou os resultados: Desempenho: 56 (em amarelo, indicando uma pontuação mediana), Acessibilidade: 69 (também em amarelo, sugerindo melhorias necessárias), Práticas recomendadas: 93 (em verde, representando boa aderência às melhores práticas), SEO: 92 (em verde, indicando boa otimização para mecanismos de busca). A seção de desempenho destaca dois indicadores principais: *First Contentful Paint*: 2,1 segundos (tempo para renderizar o primeiro conteúdo visível). *Largest Contentful Paint*: 6,4 segundos (tempo gasto até que o maior elemento visível seja carregado). Fim da descrição.

Essa ferramenta também exibe um relatório detalhado do que pode ser melhorado no site, indicando como melhorar o tempo de acesso, reduzir conteúdos redundantes e corrigir elementos que não estão acessíveis, por exemplo, imagens sem textos alternativos ou campos de formulários sem rótulos.

TECNOLOGIAS ASSISTIVAS

Já vimos que não desenvolvemos sistemas apenas para nós, o nosso desenvolvimento deve atender a diferentes necessidades. Vimos até aqui, porém, apenas como podemos preparar nossos sistemas para serem consumidos e utilizados por pessoas com necessidades de auxílio. Agora, veremos quais são as ferramentas a serem utilizadas por essas pessoas.

Existem diferentes tipos de deficiências que geram diferentes barreiras de acesso ao conteúdo. Por isso, existem também diferentes softwares e hardwares para auxiliar.

Segundo a Lei Brasileira de Inclusão da Pessoa com Deficiência, **tecnologia assistiva ou ajuda técnica** é definida como:



Produtos, equipamentos, dispositivos, recursos, metodologias, estratégias, práticas e serviços que objetivem promover a funcionalidade, relacionada à atividade e à participação da pessoa com deficiência ou com mobilidade reduzida, visando à sua autonomia, independência, qualidade de vida e inclusão social (Brasil, 2015).

Assim como define **barreiras**, por exemplo:



Qualquer entrave, obstáculo, atitude ou comportamento que limite ou impeça a participação social da pessoa, bem como o gozo, a fruição e o exercício de seus direitos à acessibilidade, à liberdade de movimento e de expressão, à comunicação, ao acesso à informação, à compreensão, à circulação com segurança, entre outros (Brasil, 2015).

Ou ainda, “podemos dizer que tecnologia assistiva são recursos de hardware e software que possibilitam o acesso de pessoas com deficiência ao dispositivo computacional e à Web” (Ferraz, 2020, p. 29).

Sem aprofundar em cada ferramenta, listaremos, segundo Ferraz (2020), algumas delas, de acordo com cada tipo de deficiência que elas podem auxiliar.



DEFICIÊNCIA VISUAL

Para deficiência visual no nível de cegueira, existem diversos **leitores de tela**.

JAWS (pago, grátis para testar, para Windows).

NVDA (grátis, para Windows).

ORCA (grátis, para Linux).

Chromevox (grátis, para Chromebook e plugin para o Google Chrome).

VoiceOver (nativo do IOs).

Talkback (grátis, para Android).

BAIXA VISÃO

No caso de baixa visão, em que a deficiência visual não é totalmente impeditiva, existem soluções como lupas digitais, que servem para aumentar ou diminuir fontes ou imagens, recursos para melhorar o contraste, ou ainda, para diferenciar cores, em casos de daltonismo:

LentePró (grátis, para Windows).

Magic (pago, para Windows).

High Contrast (grátis, plugin para Google Chrome).

Text Legibility (grátis, plugin para Firefox).

DEFICIÊNCIA AUDITIVA

Para auxiliar pessoas com deficiência auditiva, existem sistemas de tradução para Libras (Língua Brasileira de Sinais), ou ainda, ferramentas para transcrição de áudio. Dentre elas:

Handtalk (tradutor de Libras).

VLibras (tradutor de Libras).

WebCaptioner (ferramenta de transcrição).

DEFICIÊNCIA MOTORA

No caso de deficiências motoras, em que o usuário tem dificuldades para operar dispositivos, as tecnologias assistivas podem auxiliar por meio de ponteiras utilizadas na testa ou dispositivos de controle com a boca, além de sistemas de reconhecimento de voz ou movimentos. Algumas das aplicações são:

Configuração de navegação da Apple (para iPhone e iPad).

EVA Facial Mouse (para dispositivos Android).

Siri (para iPhone e iPad).

Pesquisa por voz do Google (para dispositivos Android).

DEFICIÊNCIA NEUROLÓGICA

Ainda, para o caso de deficiências neurológicas, existem ferramentas capazes de auxiliar na adaptação do ambiente e de textos, como as ferramentas que tornam a leitura mais agradável para pessoas com dislexia:

WebHelp (plugin para Google Chrome).

Dyslexia Friendly (plugin para Google Chrome).

Essas são apenas algumas dentre várias tecnologias assistivas disponíveis. O importante não é conhecer todas elas, mas saber como o seu desenvolvimento pode caminhar com elas para tornar a informação disponível a todas as pessoas. Fique atento aos seus usuários e às necessidades apresentadas no decorrer do projeto.

DESIGN PARA ACESSIBILIDADE

Produzir conteúdo para a internet não é algo fácil. Você precisa produzir algo pensando em atingir o maior número de pessoas, naturalmente, e isso já envolve dominar várias mídias diferentes: texto, áudio, vídeo, imagens etc. Para cada uma delas, ainda podemos pensar: “qual grupo de pessoas não receberá as informações se eu publicar apenas esse formato?”.

Mais do que produzir um sistema completo para, depois, pensar na acessibilidade, é importante introduzir a acessibilidade durante o design, durante as etapas de projeto, enquanto se pensa na usabilidade, e durante a implementação.

As diretrizes vistas anteriormente servem como um bom guia de desenvolvimento. Por exemplo, ao desenvolver um layout, o WCAG propõe sugestões para contraste de cores, para tamanho e espaçamento de texto, navegação via teclado etc. Se isso for incorporado ao planejamento de interfaces, muito esforço de manutenção é evitado.

Criando uma boa estrutura

A versão 5 do HTML introduziu diversas *tags* semânticas, assim como novas propriedades a elementos, para que a estrutura de documentos tenha um sentido intrínseco, para facilitar o acesso a informações a todo tipo de situação. Uma boa estruturação de páginas é o primeiro passo para uma boa acessibilidade.

Além de utilizar as *tags* corretas para os elementos corretos, baseado na semântica, a utilização de elementos corretos em formulários, por exemplo, também facilita muito o entendimento e a manipulação de informações. Embora antigamente fosse utilizado `<input type="text">` para quase todos os campos de entrada de dados, o HTML possibilitou a introdução de novos formatos ao input, como:

- Aqui está o texto com os **pontos e vírgulas (;) substituídos por pontos (.)**:
 - search: para campos de busca.
 - tel: para campo numérico de telefone.
 - email: para preenchimento de e-mail.
 - url: para preenchimento de um endereço na web.
 - date: para preenchimento de data.
 - month: para preenchimento de mês.
 - week: para preenchimento de semana.
 - time: para preenchimento de horário.
 - datetime-local: para preenchimento horário em determinada *timezone*.
 - number: campo numérico.
 - range: valor numérico para uso de slider.
 - color: para criar um *picker* de cor.

Utilizar os elementos corretos ajuda o usuário a interagir de forma correta com cada tipo de informação, principalmente ao utilizar tecnologias assistivas.

Também podemos definir as informações como idioma da página (ou de trechos da página), assim como uma informação bem estruturada em uma tabela, utilizando legendas (*caption*) e informações ajustadas sobre os cabeçalhos (Ferraz, 2020).

Pense no conteúdo

Passada a etapa de desenvolver uma boa estrutura, como o conteúdo será exibido? Existem ferramentas para que as pessoas responsáveis pelo conteúdo do site adicionem textos alternativos às imagens ou legendas aos vídeos?

EU INDICO

Além de transmitir o seu site para diferentes pessoas, com diferentes necessidades, com objetivo de elas o testarem, você pode utilizar validadores, como o **W3C Validator**, que faz uma verificação geral em sua página, para ver se o conteúdo ou a estrutura está quebrando as definições do HTML5.

Acesse: <https://validator.w3.org/>

Mais importante, sempre tente olhar para o seu conteúdo pensando em pessoas com diferentes necessidades que acessam o site por meio de diferentes dispositivos e situações. Você encontrará muitas melhorias a serem realizadas. Aproveite o conhecimento documentado nas diretrizes e em sistemas de testes automatizados.

EM FOCO

Assista à videoaula para mais informações e conhecimentos sobre este tema de aprendizagem. **Recursos de mídia disponíveis no conteúdo digital do ambiente virtual de aprendizagem.**



NOVOS DESAFIOS

A web foi desenvolvida para que o conteúdo estivesse disponível a todas as pessoas. Em nossa carreira de desenvolvimento, descobriremos, muitas vezes, que nossos sistemas ainda podem ser melhorados (sempre podem).

Esteja atento e sensível às necessidades alheias. Muitas vezes, o caminho do menor esforço ou da economia dirá: “não precisa fazer isso, quase ninguém será prejudicado por essa questão”. Acessibilidade não tem necessariamente a ver com a quantidade de pessoas, mas com a disponibilização do conteúdo chegar a todas as pessoas.

Ainda assim, pessoas que não possuem necessidades especiais, com frequência, menosprezam a quantidade real de pessoas afetadas por dificuldades permanentes ou temporárias.

Procure sempre estar atento a como as tecnologias que você usa dispõem de ferramentas para aumentar a acessibilidade e adicione essa prática ao seu dia a dia de desenvolvimento.

Você e as pessoas que utilizarem o seu sistema só têm a ganhar com isso.

VAMOS PRATICAR

1. O WCAG fornece um conjunto detalhado de diretrizes para tornar o conteúdo web mais acessível para pessoas com uma ampla variedade de deficiências (MDN Contributors, 2023, on-line).

Qual é o principal objetivo das diretrizes WCAG?

- a) Melhorar o design visual do site.
 - b) Aumentar o desempenho do site.
 - c) Garantir que os sites sejam responsivos.
 - d) Melhorar a otimização para mecanismos de busca.
 - e) Melhorar a otimização para mecanismos de busca.
2. Para melhor compreensão de tecnologia assistiva no âmbito da web, podemos dizer que tecnologia assistiva são recursos de hardware e software que possibilitam o acesso de pessoas com deficiência ao dispositivo computacional e à web (Ferraz, 2020).

Sobre as tecnologias assistivas, avalie as afirmações, a seguir:

- I - Leitores de tela são tecnologias assistivas que permitem a leitura do conteúdo de uma página.
- II - Softwares de reconhecimento de voz facilitam a navegação e interação sem uso do teclado.
- III - O contraste de cores adequado não tem impacto na acessibilidade.

É correto o que se afirma em:

- a) I, apenas.
- b) III, apenas.
- c) I e II, apenas.
- d) II e III, apenas.
- e) I, II e III.

VAMOS PRATICAR

3. As WCAG orientam, de forma muito simples, como identificar e implementar técnicas que eliminam barreiras de acesso para pessoas com deficiência (Ferraz, 2020).

Um site apresenta texto claro, alto contraste, além de ser completamente navegável pelo teclado. Qual princípio das WCAG está sendo priorizado nesse site?

- a) Perceptível.
- b) Operável.
- c) Robusto.
- d) Adaptável.
- e) Funcional.

REFERÊNCIAS

BRASIL. **Decreto-Lei nº 13.146, de 6 de julho de 2015.** Institui a Lei Brasileira de Inclusão da Pessoa com Deficiência (Estatuto da Pessoa com Deficiência). Brasília, DF: Diário Oficial da União, 2015. Disponível em: http://www.planalto.gov.br/ccivil_03/_Ato2015-2018/2015/Lei/L13146.htm. Acesso em: 15 jan. 2025.

FERRAZ, R. **Acessibilidade na web:** boas práticas para construir sites e aplicações acessíveis. São Paulo: Casa do Código, 2020.

GALA, A. S. Aprenda a avaliar a acessibilidade de sites de forma gratuita. **Hand Talk.** [s. l.], 23 nov. 2023. Disponível em: <https://www.handtalk.me/br/blog/avaliacao-de-acessibilidade-de-sites/>. Acesso em: 14 jan. 2025.

MDN CONTRIBUTORS. Entendendo as Diretrizes de Acessibilidade do Conteúdo Web. **MDN Web Docs.** [s. l.], 3 ago. 2023. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/Accuracy/Understanding_WCAG. Acesso em: 14 jan. 2025.

W3C BRASIL. **Cartilha de acessibilidade na web:** introdução. São Paulo: W3C Brasil, NIC.br, 2014. v. 1. Disponível em: <https://bibliotecadigital.acervo.nic.br/handle/123456789/2195>. Acesso em: 14 jan. 2025.

WEB Content Accessibility Guidelines (WCAG) 2.2. **W3C.** [s. l.], 5 out. 2023. Disponível em: <https://www.w3.org/TR/WCAG22/>. Acesso em: 14 jan. 2025.

CONFIRA SUAS RESPOSTAS

1. Alternativa E.

Assegurar que o conteúdo web seja acessível para todos, incluindo pessoas com deficiência. A alternativa A está incorreta, pois as WCAG têm a ver com acessibilidade, não com visual. A alternativa B está incorreta, pois as WCAG têm a ver com acessibilidade, não com desempenho.

A alternativa C está incorreta, pois as WCAG têm a ver com acessibilidade, não com responsividade.

A alternativa D está incorreta, pois as WCAG têm a ver com acessibilidade, não com busca.

2. Alternativa C.

A afirmativa I está correta, pois leitores de tela são tecnologias assistivas que permitem a leitura do conteúdo de uma página.

A afirmativa II está correta, pois softwares de reconhecimento de voz facilitam a navegação e interação sem uso do teclado.

A afirmativa III está incorreta, pois o contraste de cores adequado impacta diretamente a acessibilidade.

3. Alternativa B.

O princípio operável garante que todas as funcionalidades estejam acessíveis por meio de teclado.

A alternativa A está incorreta, pois significa que o conteúdo não pode ser invisível para mais de um sentido do usuário.

A alternativa C está incorreta, pois representa que a aplicação deve ser bem construída, de forma a ser acessível.

A alternativa D está incorreta, pois tem a ver com a capacidade de a aplicação se adaptar.

A alternativa E está incorreta, pois tem a ver com a aplicação cumprir sua função.

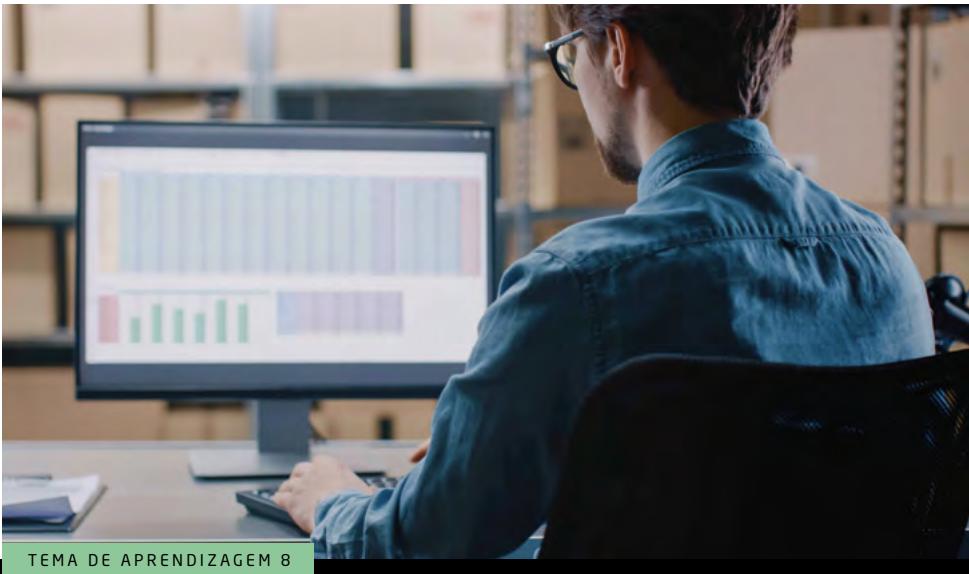
MEU ESPAÇO

MEU ESPAÇO



unidate



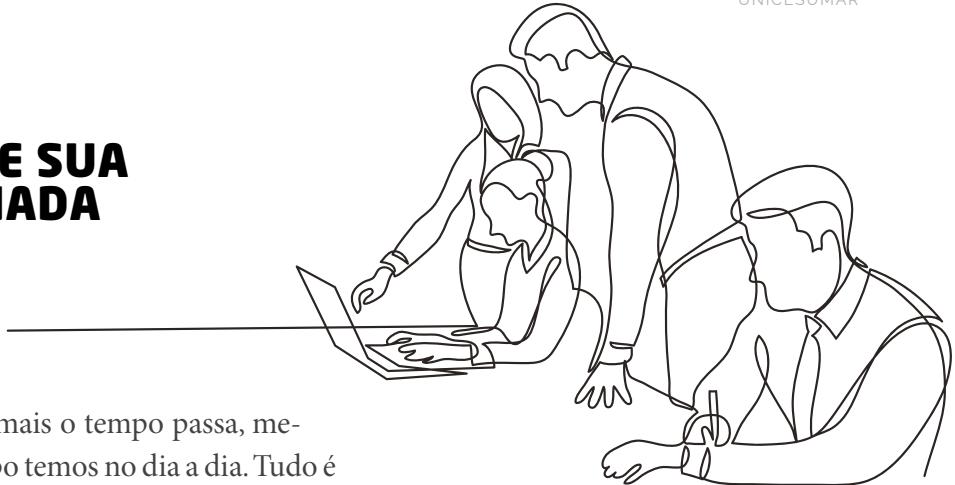


OTIMIZAÇÃO DE DESEMPENHO PARA WEBSITES RÁPIDOS

MINHAS METAS

- Compreender os fatores que afetam o desempenho de um site.
- Aplicar técnicas de otimização de código.
- Entender a otimização de imagens.
- Utilizar ferramentas de análise de desempenho.
- Monitorar continuamente o desempenho do site.
- Escrever código eficiente.
- Utilizar boas práticas de desenvolvimento.

INICIE SUA JORNADA



Quanto mais o tempo passa, menos tempo temos no dia a dia. Tudo é muito rápido, tudo é muito instantâneo, tudo precisa ser feito e entregue imediatamente. Os vídeos, assim como os momentos, estão cada vez mais curtos, enfim, tudo passa muito rápido.

Você criou um site para sua empresa, seu chefe trouxe relatórios dizendo que ele precisa ser otimizado. Está levando muito tempo para exibir as informações aos usuários, cerca de 6 segundos. Você precisa dar um jeito de deixar mais rápido.

Pesquisas indicam que, antigamente, as pessoas tinham tolerância de 5 segundos na espera do carregamento de um site, e ela caiu para cerca de 2 segundos. Todo segundo conta, então, se a informação não aparece de forma ágil, você pode perder um usuário para uma concorrência mais rápida.

Então, começa a analisar o que está deixando o carregamento lento. Seriam os tamanhos de imagens? Seriam as requisições ao banco de dados? Depois de certa análise e um trabalho de otimização, você alcança, finalmente, os 2 segundos. Hora de relaxar e curtir os 4 segundos que ficaram livres em seu dia.

Quando desenvolvemos para web, o que fazemos é replicado em muitos pontos e caminhos. Já parou para pensar no impacto que uma redução de tamanho ou de tempo pode fazer numa escala global?

PLAY NO CONHECIMENTO

Para começar o aquecimento n otimização de sites, que tal ouvir o nosso podcast, onde abordamos os principais pontos, trazendo dados sobre grandes empresas e como elas tiveram ganhos de performances com refatoração de código? **Recursos de mídia disponíveis no conteúdo digital do ambiente virtual de aprendizagem.**

VAMOS RECORDAR?

Um código otimizado é um código bem escrito e bem estruturado. Como estão as suas bases? E como está o seu conhecimento de programação cliente-servidor? Veja o vídeo, a seguir, a partir de 7:30, se quiser reforçar a noção de como funciona a troca de informações pela web.

Acesse: https://www.youtube.com/watch?v=_p6aR48FyyU

DESENVOLVA SEU POTENCIAL

Pegue o seu celular, ative o cronômetro do relógio e marque o tempo durante 5 segundos. Nesse tempo, fique olhando para o relógio e perceba o tempo passar. Vamos lá, eu espero.

O que você consegue fazer em 5 segundos? Pareceu muito tempo para esperar? Pareceu pouco? A noção de tempo não é a mesma para todo mundo, nem mesmo para uma mesma pessoa em momentos diferentes, às vezes, estamos em ritmo mais acelerado, às vezes, estamos em ritmo tranquilo.

É consenso, porém, que informações em sites devem ser exibidas de forma rápida, para manter cativa a atenção do usuário, atenção essa que já é menor do que o tempo de atenção de um peixe dourado, de acordo com uma pesquisa da Microsoft (Tecnologia [...], 2015, on-line).

Para melhorar o tempo de nosso site, precisamos, primeiro, identificar as causas da lentidão. Em seguida, analisar técnicas para otimizar o desempenho. Nesse processo, nos apoaremos em ferramentas de análise e monitoramento, para manter um código com boa performance.

IDENTIFICANDO AS CAUSAS DA LENTIDÃO

Quando a internet chegou ao usuário comum, era fácil saber as causas da lentidão: a web era muito limitada, os computadores eram muito limitados, tudo contribuía. Hoje, lidamos com máquinas mais velozes, internet rápida etc., mas ainda temos problemas de lentidão, que podem estar em diferentes fatores.



Figura 1 - Tirinha sobre otimização do BD

Fonte: <https://developerslife.tech/pt/2012/08/03/otimizacao-do-bd/>. Acesso em: 30 jan. 2025.

Descrição da Imagem: tirinha em preto e branco de três quadros cuja história se passa em uma sala de programação. O DBA está à esquerda, no computador, o chefe, ao centro, e um programador à direita, em outro computador. O chefe diz ao DBA "Eu prometo que se você melhorar o desempenho do banco de dados em 100%, eu te pago uma grade de cerveja...". O responde DBA: "Sério?? E se eu melhorar 200%??". O chefe: "Daí pago a grade e também uma feijoada...". O DBA: "Wow... E se eu melhorar 300%??". O programador: "Aí significa que o banco de dados estava uma porcaria e você que vai ter que pagar a cerveja...". O DBA cai para trás com os pés para cima, com a onomatopeia "Ploft". Na camiseta do programador, está a frase "Especialista em Bando de Dados". No rodapé da tirinha, há uma barra preta que traz, em letras brancas, no canto esquerdo, o título "The Developer's Life", no meio da barra, o texto "História real enviada por Lucas Simões". No canto direito, o número de publicação da tirinha "#678" e a indicação de Creative Commons. Fim da descrição.

Quando estamos investigando as causas da demora para renderizar uma informação, podemos ter diferentes “**gargalos**”, ou seja, diferentes pontos onde a velocidade pode estar reduzida, assim como acontece no gargalo de uma garrafa, o qual não permite que o conteúdo saia muito rapidamente.

O primeiro olhar sobre o problema pode ser feito diretamente no navegador. Todos os navegadores modernos contam com ferramentas para o desenvolvedor (*DevTools*), possíveis de serem acessadas ao inspecionar o site (ou apertar F12). Dentro do *DevTools*, há uma aba “Rede” (ou “Network”, se estiver em inglês). Nessa aba, há o registro de cada recurso carregado pelo site e o tempo levado para cada recurso. Após abrir a aba, recarregue o site para rastrear todos os elementos, você verá algo como na Figura 2.

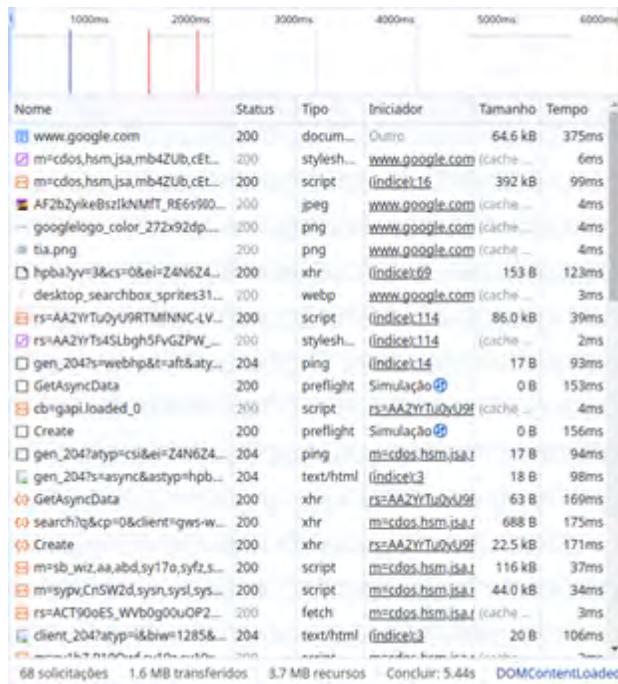


Figura 2 - Ferramentas para desenvolvedores do Google Chrome, aba “Rede” / Fonte: o autor.

Descrição da Imagem: captura de tela da aba “Rede” das ferramentas de desenvolvedor do Google Chrome. A captura exibe solicitações de rede durante o carregamento de uma página. A imagem apresenta colunas com informações sobre nome do recurso, status HTTP, tipo, iniciador, tamanho e tempo de carregamento, além de um gráfico cronológico no topo que mostra a distribuição das solicitações ao longo do tempo. Fim da descrição.

A partir dessas informações, é possível notar alguns carregamentos que levam muito tempo e podem ser otimizados, como scripts, imagens, requisições às API. Note que recursos externos, por exemplo, vídeos embutidos do YouTube, scripts de propagandas e outros, tendem a demorar para carregar, o que não atrapalha a experiência do usuário, caso não sejam o conteúdo principal, pois eles carregam assincronamente, sem impedir carregamento do que é importante.

Essa é, porém, uma verificação apenas superficial. Para ter uma noção real dos gargalos, você pode utilizar uma ferramenta como o **Lighthouse**, que também está presente nas *DevTools* do Google Chrome. Execute a verificação em seu site e execute, também, em um site concorrente, para comparar o tempo de execução de ambos.

Antes de ver o funcionamento do Lighthouse, precisamos conversar sobre algumas métricas importantes na análise de desempenho de um site, relacionadas ao impacto para o usuário nos primeiros momentos do carregamento:

FIRST CONTENTFUL PAINT (FCP)

O tempo entre o momento em que o usuário navegou até a página e o momento em que qualquer parte do conteúdo dela é renderizada na tela.

LARGEST CONTENTFUL PAINT (LCP)

O tempo de renderização da maior imagem, bloco de texto ou vídeo visível na janela de visualização, em relação ao momento em que o usuário navegou pela primeira vez até a página.

FIRST INPUT DELAY (FID)

O tempo de resposta de um site desde a primeira interação do usuário até a resposta do navegador.

INTERACTION TO NEXT PAINT (INP)

A velocidade com que a página se atualiza quando o visitante interage com ela.

CUMULATIVE LAYOUT SHIFT (CLS)

Medida do maior número de pontuações de mudança de layout para cada mudança de layout não esperada que ocorre durante todo o ciclo de vida de uma página.

Cada uma das métricas aponta para algo diferente do carregamento do site, o que pode pesar na usabilidade do usuário, porém pesa muito mais para a indexação do site por ferramentas de busca. O Google, por exemplo, utiliza um conjunto de indicadores, chamado **Core Web Vitals**, para classificar o desempenho de sites (Walton, 2020, on-line).

Em 2024, a Google passou a utilizar a métrica *Interaction to Next Paint* em vez de *First Input Delay* no Core Web Vitals. Isso não significa que o FIP deixou de ser importante, mas que o carregamento de novas informações ao interagir com a página tem sido vital (Fraga, 2024, on-line).

Portanto, se a intenção é analisar o tempo de carregamento, a ferramenta **PageSpeed Insights**, do Google, pode ser mais indicada (Monteiro, 2024, on-line). Apesar de estar bastante relacionada ao Lighthouse, fornece relatórios mais direcionados.

Além das ferramentas do Google, existem outras, como **WebPageTest** e **GTmetrix**, que também fazem bons trabalhos de análise. Vale a pena comparar os resultados.

TÉCNICAS DE OTIMIZAÇÃO

Como vimos, são vários os fatores que podem ocasionar lentidão no carregamento de um site. Em geral, as ferramentas de análise já nos propõem soluções para os problemas. Veremos, aqui, algumas das soluções mais comuns.

Minificação de HTML, CSS e Javascript

Ao trafegar conteúdo na rede, o tamanho do conteúdo sempre tem bom impacto, afinal, quanto menor ele é, mais rápida é a transferência de dados. Uma técnica comum é compactar os dados: compactá-los na origem, trafegá-los e descompactar no destino. Em geral, o tempo gasto para compactar e descompactar ainda é menor do que o tempo economizado na transferência.

Quando falamos em “**minificação**” de arquivos, a ideia é um pouco diferente. Como as linguagens HTML, CSS e Javascript baseiam-se em arquivos-texto interpretados, para o computador, importa o código que será executado e não os nomes de variáveis ou funções. A técnica de minificação consiste em remover do arquivo tudo o que é dispensável, como espaços em branco, quebras de linhas, comentários, caracteres dispensáveis em nomes de funções e variáveis. Por exemplo, observe o trecho de código, a seguir, em Javascript:

```
alts.forEach((box) => {
    box.addEventListener("click", (event) => {
        document.querySelector(".selecionada")?.classList.remove("selecionada");
        event.target.classList.add("selecionada");
        btnPerguntar.disabled = false;
        mensagem.innerText = "";
    });
});
```

Ignoraremos o contexto deste código e as definições de variáveis, mas ele poderia ser escrito, sem perda de funcionalidade, da seguinte forma:

```
a.forEach((b) => {b.addEventListener("click", (e) => {document.querySelector(".selecionada")?.classList.remove("selecionada"); e.target.classList.add("selecionada"); c.disabled=false;d.innerText=""})});
```

É um exemplo bem pequeno, mas o primeiro trecho possui 259 caracteres, enquanto o segundo possui 195 caracteres. O segundo é bem mais complicado para ser lido por humanos, mas isso é o que a máquina precisa a partir de todo aquele código.

Existem algumas ferramentas que você pode baixar para fazer esse serviço, mas a maioria dos *frameworks* atuais tem bibliotecas que fazem isso. É possível conferir a documentação para configurar a minificação, ao publicar o seu conteúdo.

Se estiver fazendo um projeto com HTML, CSS e Javascript puro, utilize uma ferramenta dessas para minificar o código. Existem várias delas de uso gratuito na internet.

Compactação de imagens

As imagens utilizadas em um site causam forte impacto no carregamento de um site. Você já deve ter ouvido o ditado: “uma imagem vale mais do que mil palavras”. Em relação ao tamanho dos dados, isso também é verdade.

As imagens não são embutidas no HTML, elas surgem após o carregamento do documento e, dependendo de como são usadas no site, podem segurar uma área vazia até o carregamento ser concluído. Por isso, é uma boa prática reduzir o tamanho de imagens que serão utilizadas.

Os formatos de imagens mais comuns, como JPG, PNG ou GIF, já são em si formatos de compactação que trabalham com diferentes algoritmos, porém existem programas, como **Trimage** ou **ImageOptim**, que conseguem “enxugar” o tamanho de arquivos de imagens sem perda de qualidade, retirando informações redundantes, o que é uma boa estratégia para imagens on-line.

Além disso, o tamanho das imagens também precisa ser observado. Você não precisa carregar uma imagem de 3000 px de largura, para colocar em uma área visível de 500 px. A diferença do tamanho do arquivo é bem grande.

Caso seu site ajuste-se bem a diferentes tamanhos de telas, você pode ter também diferentes versões da sua imagem, para serem carregadas segundo a necessidade. O HTML5 introduziu a tag `<picture>`, a qual permite utilizar diferentes imagens de acordo com a necessidade (MDN Contributors, 2024, on-line).

```
<picture>

    <source media="(max-width: 799px)" srcset="elva-
480w-close-portrait.jpg" />

    <source media="(min-width: 800px)" srcset="elva-
800w.jpg" />

</picture>
```

Utilizando a tag `<picture>`, você pode definir diferentes arquivos-fonte de imagens, com a tag `<source>`. Por meio da propriedade “media”, há a possibilidade de indicar para qual caso será carregada determinada imagem.

No exemplo anterior, se a tela for pequena (até 799 px de largura), a primeira imagem será exibida, senão a segunda. Por fim, a tag `` mantém uma imagem padrão caso o navegador não carregue uma das anteriores.

Apesar de bastante utilizados e suportados, os formatos PNG e JPG já são considerados ultrapassados no desenvolvimento web, sendo recomendados os usos de WEBP e AVIF para imagens em mapa de bits (*bitmaps*), ou imagens em SVG para vetores. Você também pode usar a tag `<picture>` para oferecer diferentes formatos.

```
<picture>

    <source type="image/svg+xml" srcset="pyramid.svg" />

    <source type="image/webp" srcset="pyramid.webp" />

</picture>
```

Caso o navegador não suporte um formato, ele tenta carregar o próximo. Dessa forma, você pode oferecer imagens em formato mais otimizado, com tamanho bastante menor. Você também encontra ferramentas gratuitas para converter suas imagens, como a **cwebp**.

Lazy loading

Em português, traduzimos *lazy loading* para “**carregamento lento**”, porque a tradução literal, “carregamento preguiçoso”, contém um significado negativo. É

uma estratégia bastante utilizada no desenvolvimento web e, segundo Patel (2023, on-line), “deveria ser lei em sua empresa”.

Como já vimos, alguns elementos, por exemplo, vídeos, imagens pesadas ou demais elementos embutidos, podem prejudicar o tempo de carregamento de um site. Por isso, a técnica consiste em atrasar, propositalmente, o carregamento desses elementos, para que o conteúdo principal seja exibido mais rapidamente.

Na prática, o *lazy loading* consiste em dividir os elementos carregados para utilizar um carregamento assíncrono. Essa técnica melhora a experiência para o usuário e reduz a taxa de rejeição, tornando os primeiros segundos de contato do usuário com o conteúdo mais agradáveis.

Existem algumas técnicas a serem observadas para implementar o carregamento lento (Patel, 2023, on-line):

DIVISÃO DE CÓDIGO

Em vez de carregar tudo de uma vez, carregue apenas o necessário, quando necessário.

JAVASCRIPT

O carregamento pode ser adiado até quando necessário, utilizando como “module”.

CSS

Muitas vezes, não é necessária toda definição logo no início, por isso, é possível carregar apenas o que se precisa para a primeira impressão.

FONTES

Por ser externo, priorize o carregamento apenas das fontes utilizadas no início.

IMAGENS E FRAMES

O atributo *loading* pode indicar ao navegador se o elemento precisa ser carregado logo no início ou apenas quando a página for descida.

Não são as únicas estratégias a serem observadas, mas já indicam uma melhoria e trazem uma luz sobre o que mais pode ser feito para o seu site.

CDN (*Content Delivery Network*)

Além do código e da renderização do navegador, quando tratamos de web, a transferência de dados é algo crucial para o tempo de resposta de um site. Já vimos estratégias para reduzir tamanhos de arquivos e os pacotes trafegados, mas o quanto bom deve ser o servidor para atender os usuários?

O desempenho dos servidores na entrega do conteúdo depende da capacidade do servidor e da localização dele em relação ao destino, para ter uma entrega que faça um caminho mais curto.

No intuito de resolver essa questão, existem as **CDN** (*Content Delivery Networks*), redes de entregas de conteúdos que utilizam diferentes servidores, em diferentes localidades, para servir cópias de um conteúdo. Com isso, o desempenho pode ser melhorado, servindo as informações de onde estiver mais próximo para o usuário (O QUE [...], 2023, on-line).

Além de oferecer cópias do conteúdo, uma CDN serve como um *cache* para o site, aliviando o número de requisições ao seu servidor original. Também funciona como uma camada intermediária, para prevenir até mesmo ataques como o DDoS (ataque de negação de serviços). Assim, o domínio do site não aponta mais para o servidor, mas ao CDN, que aponta para o servidor.

Existem várias opções de CDN, gratuitas ou pagas, capazes de melhorar muito o tempo de resposta e a disponibilidade dos sites. A configuração é bem simples e vale muito a pena utilizá-las.

BOAS PRÁTICAS PARA CÓDIGOS EFICIENTES

Entender como a web funciona nos faz produzir sites mais bem otimizados. Entender o que precisamos otimizar em nossos sites nos faz aprender mais como a web funciona. É um lindo ciclo de aprendizagem onde, sem exagero, podemos aprender todos os dias.

O site *web.dev* contém uma área com casos de sucessos, onde conta histórias de como grandes empresas obtiveram ganhos de performance por meio do atendimento às recomendações de melhorias apontadas por relatórios, melhorias essas que, basicamente, envolviam o código. A Netflix, por exemplo, teve bons ganhos de performance utilizando recursos do CSS que permitiram tirar trechos de código Javascript os quais realizavam a mesma função (Weeks *et al.*, 2024, on-line).

Um código bem escrito e otimizado é o principal fundamento para alcançar um bom desempenho no carregamento e no funcionamento em geral. Vejamos algumas dicas importantes.

Evite código redundante

Computadores são máquinas boas em repetir coisas, eles são basicamente feitos para isso, mas você não precisa se repetir, nem seu código. A repetição desnecessária de código aumenta o tamanho dos arquivos, assim como o risco de erros. Se ele estiver repetido, toda correção nele precisa ser replicada nas cópias.

Identifique padrões que possam ser replicados e crie funções reutilizáveis bem como suas próprias bibliotecas de código, ou utilize de terceiros (desde que tenham origens confiáveis).

Por exemplo, o código, a seguir, realiza duas vezes a mesma operação.

```
const total = preco * 1.1;
const totalDescontado = (preco - desconto) * 1.1;
```

O que seria essa multiplicação por 1,1? Seria uma multa de 10% por atraso? Talvez impostos? Seria uma conversão para outra moeda?

Se essa operação ocorre em diferentes pontos de código e, em determinado dia, esse valor mudar para 1,15, será preciso alterar todas as menções no código. Por isso, prefira uma solução como essa:

```
const aplicarTaxa = (valor) => valor * 1.1;

const total = aplicarTaxa(preco);

const totalDescontado = aplicarTaxa(preco - discount);
```

O código fica mais limpo, mais semântico e mais simples de ser alterado. Nesse caso, não houve ganho em quantidade de código escrito, porque a função tem apenas uma linha, mas, caso a função realizasse várias operações que se repetem, haveria um ganho visível.

Use estruturas de dados adequadas

Em geral, as estruturas de dados são menosprezadas por boa parte dos profissionais, que usam apenas *arrays* ou listas básicas. A estrutura de dados pode ser um fator de otimização das soluções, fornecendo melhor desempenho e economia de recursos.

Por exemplo, o código, a seguir, trata um *array* e executa sobre ele um filtro para remover os elementos repetidos.

```
const array = [1, 2, 2, 3];

const arraySemRepeticao = array.filter((value, index,
self) => self.indexOf(value) === index);
```

O mesmo pode ser obtido ao se utilizar uma estrutura de conjunto (Set), que já é uma estrutura otimizada para manter elementos sem repetição.

```
const array = [1, 2, 2, 3];

const arraySemRepeticao = [...new Set(array)];
```

Temos uma sintaxe mais limpa para o nosso código e, assim, deixamos a otimização para o interpretador.

Utilize *design patterns* (padrões de projeto)

Existem vários padrões de projetos que são técnicas já testadas e amplamente utilizadas para problemas recorrentes em desenvolvimento de software. Utilizando esses padrões, você evitar problemas comuns que acontecem no desenvolvimento. Serão destacados, aqui, três padrões:

- **Singleton**: utilizado para garantir uma única instância de um objeto, é comumente usado em conexões com o banco de dados, onde não é necessário abrir várias conexões, se uma estiver aberta, pode ser utilizada.
- **Factory**: utilizado para criar objetos de maneira mais controlada, criando mecanismos de instanciação de objetos padronizados.
- **Observer**: utilizado para gerenciar dependências entre diferentes objetos.

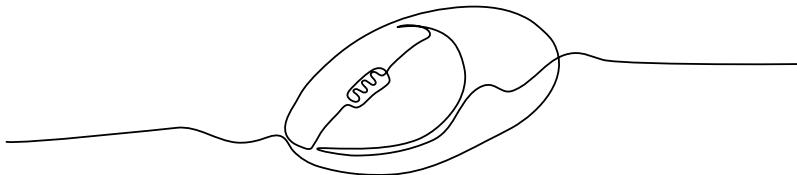
Como observado, porém, no livro *Código Limpo: Habilidades Práticas do Agile Software*, de Robert C. Martin (2009, p. 163):



Os padrões facilitam a reutilização de ideias e componentes, recrutam pessoas com experiência considerável, encapsulam boas ideias e conectam os componentes. Entretanto, o processo de criação de padrões pode, às vezes, ser muito longo para que o mercado fique à espera deles, e alguns padrões acabam se desviando das necessidades reais das pessoas a quem eles pretendem servir.



Em resumo, os padrões de design lhe auxiliarão bastante, mas você não precisa usar todos, nem se forçar a usar se não houver a necessidade. A prática é a melhor ferramenta para aprimorar o código.



EM FOCO

Assista à videoaula para mais informações e conhecimentos sobre este tema de aprendizagem. **Recursos de mídia disponíveis no conteúdo digital do ambiente virtual de aprendizagem.**

NOVOS DESAFIOS

Desenvolver para web é algo extremamente recompensador, porque publicamos nosso código em uma plataforma que possibilita acesso universal, porém, ao mesmo tempo, é bastante desafiador, pois ela está em constante evolução, de forma muito rápida.

Enquanto a exigência do usuário aumenta em uma taxa inversamente proporcional à paciência, precisamos oferecer boas soluções, de forma rápida e leve, para que os usuários tenham uma boa experiência na ponta dos dedos.

O caminho do desenvolvimento web demanda aprendizado constante. Não tenha medo, aluno, de abraçar as referências, consumir tutoriais e analisar, com carinho, todos os relatórios gerados por ferramentas de verificação de performance. É triste receber um relatório ruim sobre seu site, parece que estão falando mal de seu filho, mas é recompensador realizar alterações e ver as melhorias instantâneas nas notas de desempenho.

Então, siga em frente. Ninguém desenvolve sistemas perfeitos, sempre trabalhe para melhorar e aumentar a performance do seu software.

VAMOS PRATICAR

1. "Para definir a posição de um site em uma determinada busca, o Google considera vários sinais de experiência da página (page experience), incluindo seus Core Web Vitals. Quanto melhores forem as pontuações do Core Web Vitals do seu site, mais provável que ele ocupe uma posição alta nas páginas de resultados dos motores de busca (SERPs)" (Santana, 2024, on-line).

Qual ferramenta é amplamente utilizada para analisar o desempenho de um site?

- a) Figma.
 - b) GitHub.
 - c) Postman.
 - d) MongoDB.
 - e) PageSpeed Insights.
2. "Técnicas de otimização ajudam a aumentar o tráfego e as conversões de um site. Diferente das estratégias de SEO, que otimizam o site para mecanismos de busca tipo o Google, as estratégias de otimização priorizam a experiência dos visitantes" (Barro, 2024, on-line).

Considerando o texto apresentado, avalie as afirmações, a seguir:

- I - Minificar arquivos HTML, CSS e Javascript reduz o tempo de carregamento do site.
- II - Compactar imagens melhora o desempenho do site.
- III - Aumentar o número de requisições HTTP melhora o tempo de resposta do servidor.

É correto o que se afirma em:

- a) I, apenas.
- b) III, apenas.
- c) I e II, apenas.
- d) II e III, apenas.
- e) I, II e III.

VAMOS PRATICAR

3. "O First Input Delay (FID), também conhecido como latência de entrada, calcula o tempo de resposta de um site desde a primeira interação do usuário até a resposta do navegador. Um desenvolvedor percebeu que o site está lento devido ao carregamento de muitos arquivos JavaScript" (Estrella, 2024, on-line).

Qual seria a solução recomendada para esse caso apresentado no texto?

- a) Remover todos os arquivos Javascript.
- b) Usar *cache* no navegador.
- c) Dividir os arquivos Javascript em pequenos pacotes.
- d) Aumentar o número de bibliotecas Javascript.
- e) Reduzir o número de usuários simultâneos.

REFERÊNCIAS

BARRO, B. B. Otimização de Sites: 10 Estratégias para Melhorar a Velocidade, UX e SEO + Dicas de Ferramentas. **Hostinger Tutoriais**, *[s. l.]*, 17 dez. 2024. Disponível em: <https://www.hostinger.com.br/tutoriais/otimizacao-de-sites>. Acesso em: 31 jan. 2025.

ESTRELLA, C. O que é First Input Delay e como Melhorá-lo. **Hostinger Tutoriais**, *[s. l.]*, 19 abr. 2024. Disponível em: <https://www.hostinger.com.br/tutoriais/first-input-delay>. Acesso em: 31 jan. 2025.

FRAGA, R. Core Web Vitals: Google substituirá o First Input Delay (FID) pela nova métrica Interaction to Next Paint (INP). **Discovery**, *[s. l.]*, 5 fev. 2024. Disponível em: <https://googlediscovery.com/2024/02/05/core-web-vitals-google-substituir%C3%A1-o-first-input-delay-fid-pela-nova-m%C3%A9trica-interaction-to-next-paint-inp/>. Acesso em: 30 jan. 2025.

MARTIN, R. C. Código **limpo**: habilidades práticas do Agile Software. Rio de Janeiro: Alta Books, 2009.

MDN CONTRIBUTORS. Imagens responsivas - HTML: Linguagem de Marcação de Hipertexto. **MDN Web Docs**, *[s. l.]*, 21 dez. 2024. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/HTML/Responsive_images. Acesso em: 30 jan. 2025.

MONTEIRO, J. Interaction to Next Paint (INP): Entenda a Nova Métrica das Core Web Vitals e como Otimizá-la. **Hostinger Tutoriais**, *[s. l.]*, 19 abr. 2024. Disponível em: <https://www.hostinger.com.br/tutoriais/interaction-to-next-paint>. Acesso em: 30 jan. 2025.

O QUE É CDN (Content Delivery Network)? **IBM**, *[s. l.]*, 19 jun. 2023. Disponível em: <https://www.ibm.com/br-pt/topics/content-delivery-networks>. Acesso em: 31 jan. 2025.

PATEL, N. Lazy loading: o que é e quais são os benefícios. **Neil Patel Blog**, *[s. l.]*, 22 set. 2023. Disponível em: <https://neilpatel.com/br/blog/lazy-loading/>. Acesso em: 30 jan. 2025.

SANTANA, B. O que são Core Web Vitals e como medi-los para melhorar seu site. **Hostinger Tutoriais**, *[s. l.]*, 6 nov. 2024. Disponível em: <https://www.hostinger.com.br/tutoriais/core-web-vitals>. Acesso em: 31 jan. 2025.

TECNOLOGIA deixa humanos com atenção mais curta que de peixinho dourado, diz pesquisa. **BBC News Brasil**, *[s. l.]*, 16 maio 2015. Disponível em: https://www.bbc.com/portuguese/noticias/2015/05/150515_atencao_peixinho_tecnologia_fn. Acesso em: 30 jan. 2025.

WALTON, P. Web Vitals. **web.dev**, *[s. l.]*, 4 maio 2020. Disponível em: <https://web.dev/articles/vitals?hl=pt-br>. Acesso em: 30 jan. 2025.

WEEKS, J. et al. Liberando o poder das consultas de contêiner do CSS: lições da equipe da Netflix. **web.dev**, *[s. l.]*, 10 dez. 2024. Disponível em: https://web.dev/case-studies/netflix--cq?hl=pt_br. Acesso em: 31 jan. 2025.

CONFIRA SUAS RESPOSTAS

1. Alternativa E.

O PageSpeed Insights é uma ferramenta do Google projetada para analisar o desempenho de sites e sugerir melhorias.

A alternativa A está incorreta, pois o Figma é uma ferramenta para criar interfaces.

A alternativa B está incorreta, pois o GitHub é um servidor de repositórios de arquivos.

A alternativa C está incorreta, pois o Postman é um sistema para fazer requisições HTTP.

A alternativa D está incorreta, pois o MongoDB é um sistema de banco de dados NoSQL.

2. Alternativa C.

As afirmações I e II estão corretas, pois a minificação e a compactação de imagens reduzem o tamanho dos arquivos, melhorando o desempenho.

A afirmação III está incorreta, pois aumentar requisições HTTP pode, na verdade, piorar o desempenho.

3. Alternativa C.

A divisão permite que apenas os scripts necessários sejam carregados, quando necessários.

A alternativa A está incorreta, pois remover os arquivos Javascript remove a parte de interação dinâmica.

A alternativa B está incorreta, pois o *cache* ajuda no desempenho, mas não tem a ver com os arquivos Javascript.

A alternativa D está incorreta, pois aumentar as bibliotecas pode aumentar o número de arquivos carregados.

A alternativa E está incorreta, pois a intenção é aumentar o número de usuários simultâneos e não diminuir.

MEU ESPAÇO

MEU ESPAÇO



TEMA DE APRENDIZAGEM 9

TESTES E DEPURAÇÃO EM DESENVOLVIMENTO FRONT-END

MINHAS METAS

- Entender os conceitos de depuração.
- Conhecer as ferramentas de depuração.
- Realizar a depuração no navegador.
- Apresentar os testes automatizados.
- Aprender a instalar os pacotes npm.
- Criar testes unitários.
- Entender os diferentes tipos de testes.

INICIE SUA JORNADA

Imagine-se desenvolvendo uma aplicação web, com várias funcionalidades e uma interface bonita, interativa. O projeto parece impecável até você abrir o navegador e encontrar um erro que trava completamente o site. Agora, imagine o usuário tentando acessar o mesmo sistema: a experiência frustrante faz com que ele desista e nunca mais retorne.

Para evitar (ou tentar evitar) surpresas desagradáveis, os testes e a depuração não são apenas ferramentas técnicas, mas também práticas fundamentais para transformar o trabalho do desenvolvedor em algo mais profissional. Garantir que o código funcione corretamente e atenda às expectativas do usuário final vai além de cumprir um requisito funcional, é preciso oferecer qualidade, construir confiança e criar valor real para quem usa o produto.

Resolver problemas no desenvolvimento exige um olhar investigativo, como se você fosse um detetive à procura de pistas. É preciso testar hipóteses, explorar soluções e validar cenários, aprendendo algo novo a cada tentativa. É um processo dinâmico que não apenas resolve falhas, mas também ajuda a compreender como sua aplicação comporta-se em diferentes situações. Algumas ferramentas tornam essa jornada mais prática e eficiente, lhe permitindo simular, testar e aperfeiçoar suas ideias.

Ao realizar testes e depuração, você percebe que erros fazem parte do processo, mas também que é possível aprender com eles e, assim, melhorar continuamente. Esse aprendizado não se limita à aplicação em si, ele aprimora suas habilidades profissionais, aumenta sua capacidade de prever problemas, planejar soluções e entregar sistemas mais robustos. Testar e depurar são, além de tarefas técnicas, momentos de amadurecimento profissional.



PLAY NO CONHECIMENTO

Realizar testes e depuração é algo que dura por toda a carreira de desenvolvimento, independentemente do tamanho do projeto ou da empresa. Grandes organizações também têm ótimos exemplos na área de testes e depuração. Conheça um pouco mais delas nesse episódio do podcast. **Recursos de mídia disponíveis no conteúdo digital do ambiente virtual de aprendizagem.**

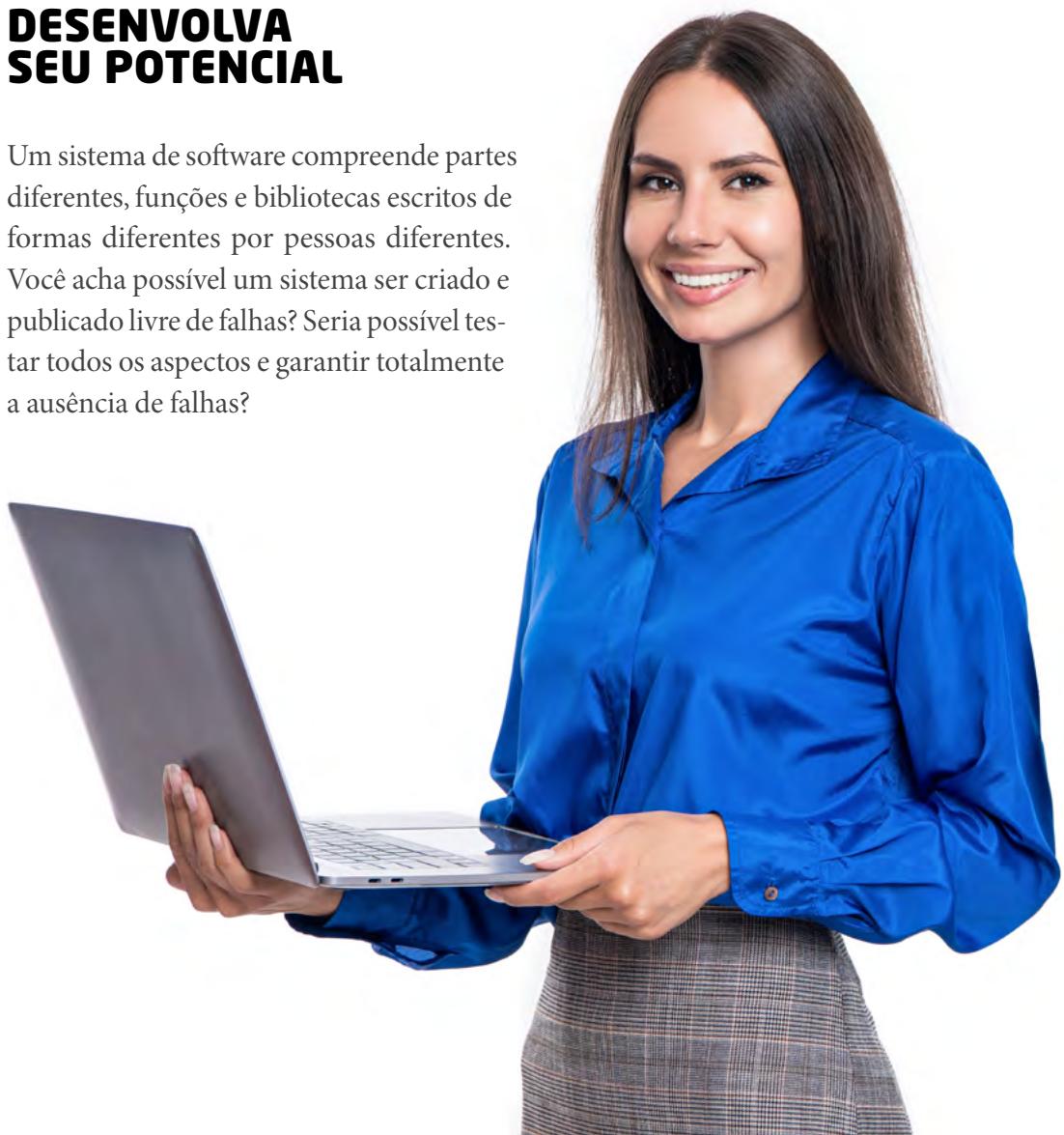
VAMOS RECORDAR?

Ao programar em Javascript, as ferramentas para desenvolvedores de um navegador são muito importantes, principalmente o console, onde aparecem as mensagens de erro e o qual é possível ser utilizado de forma interativa. Aprenda a dominar o console, principalmente nos primeiros 20 minutos do vídeo, a seguir.

Acesse: <https://www.youtube.com/watch?v=x5a4HyDMoNM>

DESENVOLVA SEU POTENCIAL

Um sistema de software compreende partes diferentes, funções e bibliotecas escritos de formas diferentes por pessoas diferentes. Você acha possível um sistema ser criado e publicado livre de falhas? Seria possível testar todos os aspectos e garantir totalmente a ausência de falhas?



De forma bem resumida, um programa pode ser garantidamente livre de erros, caso ele seja tão simples que não tenha tanta utilidade, como um programa que apenas exibe “Olá, mundo!” na tela, no entanto, para cobrir todos os detalhes e garantir a ausência de erros, necessitariíamos de um formalismo matemático inviável, ou, como dizem hoje em dia, “intankável”, a sistemas complexos.

Isso não quer dizer que não haja solução ou que não valha a pena. Aqui, discutiremos **formas viáveis e importantes** para identificar bem como corrigir erros de software.

DEPURAÇÃO: O PROCESSO DE DETETIVE DO SOFTWARE

Você sabe como funciona o fluxo de execução de um software? Em linhas gerais, um sistema inicia sua execução pelo começo, como esperado, e segue uma sequência de instruções. Essa sequência pode ser alterada por estruturas de controle (*if/switch*), por estruturas de repetição (*for/while/do*), por chamadas de funções, por eventos ou outras situações, mas há uma sequência lógica na execução.

O processo de **depuração**, do inglês *debugging*, consiste em encontrar e remover erros existentes no código de programas. Para isso, podemos executar o programa e seguir as mensagens de erro, ou utilizar um *debugger* (ou **depurador**).

Encontrar e remover erros existentes no código de programas

ZOOM NO CONHECIMENTO

O termo “*debug*” vem do inglês, com a ideia de “retirar *bugs*”. Em inglês, “*bug*” significa “inseto”. Esse termo é associado a falhas em sistemas devido a insetos que entravam nos primeiros computadores, atraídos pelo calor, e que queimavam componentes, causando as falhas em sistemas. A primeira falha por inseto documentada foi realizada por Grace Hopper, a qual liderava uma equipe quando trabalhava no Mark II, em 1947, na Universidade de Harvard. Uma mariposa ficou presa no sistema elétrico, interrompendo a transmissão de energia. A mariposa em questão foi anexada ao relato (Selim, 2024, on-line).

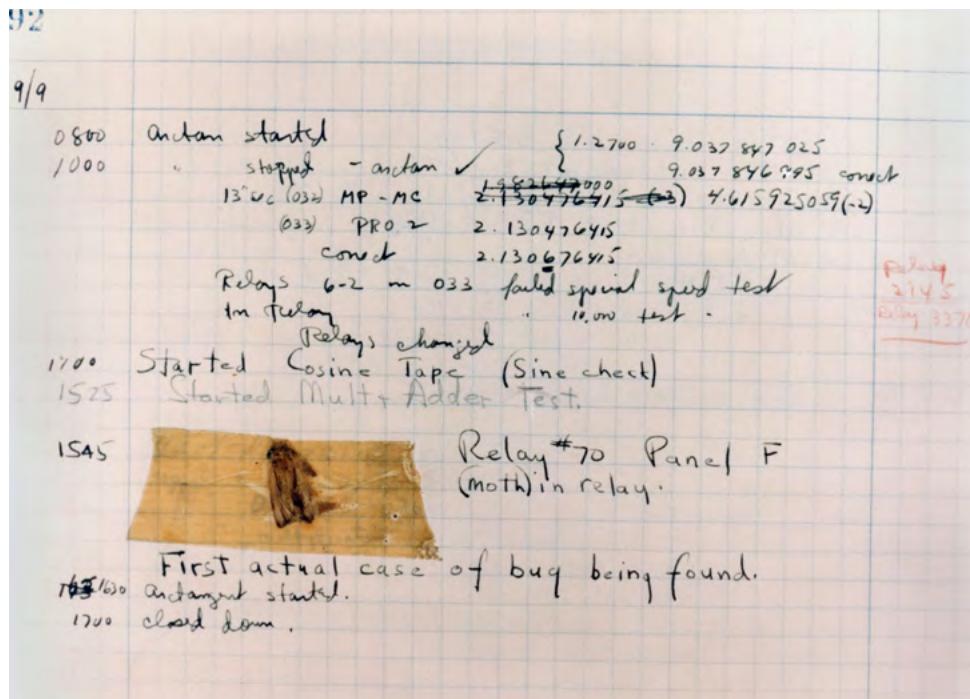


Figura 1 - Documento com registro do primeiro bug de computador real

Fonte: https://commons.wikimedia.org/wiki/File:First_Computer_Bug,_1945.jpg. Acesso em: 7 fev. 2025.

Descrição da Imagem: fotografia colorida de uma página de um diário de laboratório da equipe de computação de Harvard em 1947. O documento registra a descoberta do primeiro bug de computador real, um inseto (mariposa) preso em um relé do computador Mark II. A mariposa está colada na página com uma anotação sobre o evento. Esse incidente popularizou o termo bug para erros em sistemas computacionais. O texto é escrito à mão sobre papel quadriculado. Fim da descrição.

O processo de identificar e corrigir erros é uma parte importante no processo de desenvolvimento de software. Saber encontrar erros é uma habilidade adquirida que melhora com a experiência.

Todo mundo erra: a programação orientada a erros

Alguma vez você, com muita confiança, escreveu um código e, quando o executou, ele retornou um erro “cabuloso”? Erros acontecem com frequência durante o desenvolvimento, e o nosso objetivo é resolvê-los antes de entregar o programa aos usuários, ou tão logo sejam identificados.

É importante entender o que causou o erro. Por isso, analisaremos os três tipos de erros que podem acontecer em seu código, segundo Aho *et al.* (2008):

ERRO LÉXICO

Erra na escrita de alguma palavra reservada, função, operador ou variável. É o que acontece quando digitamos uma palavra errada, como digitar *fr* em vez de *for*.

ERRO SINTÁTICO

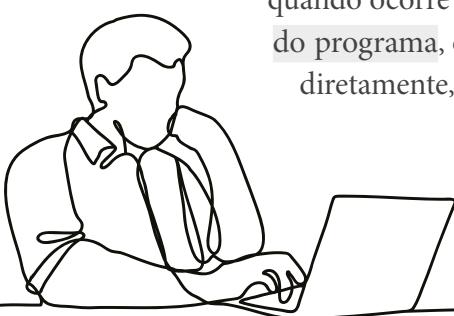
É um erro na gramática da linguagem. As palavras foram todas escritas corretamente, mas podem estar em ordem errada ou faltando partes. O famoso caso de esquecer de fechar os parênteses ou esquecer o ponto e vírgula no final da linha, por exemplo.

ERRO SEMÂNTICO

Erro na ideia ou no significado do código. As palavras estão corretas e a gramática também, mas o sentido está incorreto. É o erro mais difícil de detectar, pois envolve entender o que aquele código deve fazer. Um exemplo seria escrever uma função para somar dois números em uma operação interna que fosse uma subtração.

Os **erros léxicos** e os **erros sintáticos** são, geralmente, detectados pelo compilador (ou interpretador) do código, ao tentar gerar um executável. Nesse caso, uma mensagem de erro é gerada, portanto, é importante aprender a ler essas mensagens, pois elas dão o caminho para a correção, indicando o tipo de erro e onde esse erro ocorreu no código (linha e coluna).

No caso de códigos em Javascript executados no navegador, quando ocorre um **erro fatal**, o qual impede a continuação do programa, o navegador não exibe esse erro ao usuário diretamente, ele coloca no console, nas ferramentas para desenvolvedores (acessado pela tecla F12 ou pelo menu “Inspecionar”).



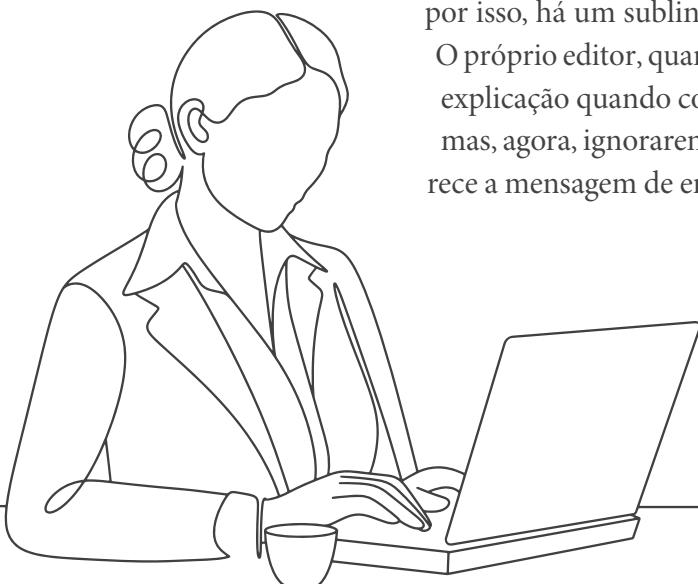
```
trivia > JS extras.js > ...
1  for (let index = 0; index < 10) {
2      console.log(index);
3  }
4
5  console.log("teste!");
6
7
```

Figura 2 - Trecho de código com erro sintático / Fonte: o autor.

Descrição da Imagem: captura de tela de um trecho de código em Javascript, no arquivo extras.js. Na linha 1: for (let index = 0; index < 10) { (nessa linha há um sublinhado vermelho abaixo do fechamento do parênteses); na linha 2: console.log(index); na linha 3: }; a linha 4 está vazia; na linha 5: console.log("teste!"); as linhas 6 e 7 estão vazias. Fim da descrição.

Vemos, na Figura 2, um trecho de código com um erro sintático. Os editores de código (como o Visual Studio Code, usado na figura) costumam indicar possíveis erros léxicos e sintáticos enquanto digitamos, por isso, há um sublinhado vermelho na imagem.

O próprio editor, quando indica algo, fornece uma explicação quando colocamos o mouse por cima, mas, agora, ignoraremos isso, para ver como aparece a mensagem de erro no navegador.



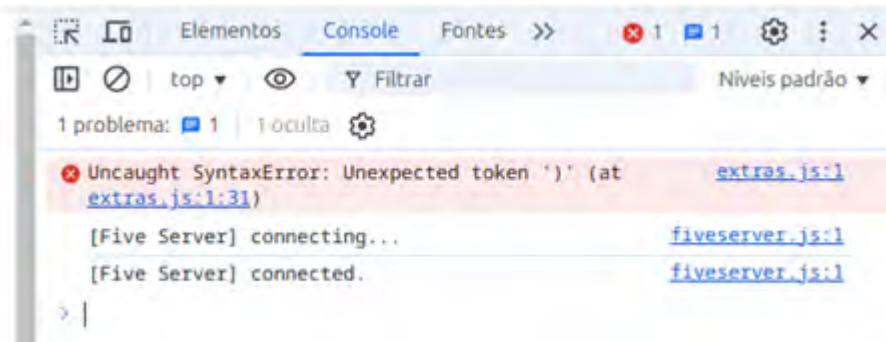


Figura 3 - Mensagem de erro no console do navegador / Fonte: o autor.

Descrição da Imagem: captura de tela do console do navegador. Nas abas acima, a aba "Console" está selecionada. Sobre um fundo vermelho há a mensagem: "Uncaught SyntaxError: Unexpected token ')' (at extras.js:1:31)", à direita da mensagem, um link escrito "extras.js:1". Abaixo, a mensagem "[Five Server] connecting..." e "[Five Server] connected.". Em seguida, o símbolo ">", indicando o prompt do console. Fim da descrição.

Na Figura 3, podemos ver o erro gerado no navegador ao executar o código que consta na Figura 1. Em geral, essas mensagens aparecem em inglês, portanto, é importante aprender, pelo menos o básico, para entender melhor as situações (ou utilizar uma ferramenta para traduzir).

As ferramentas de LLM (como ChatGPT, Copilot ou DeepSeek) também ajudam bastante a entender as mensagens de erro. A seguir, compreenderemos a mensagem de erro retornada:

```
Uncaught SyntaxError: Unexpected token ')' (at extras.js:1:31)
```

A mensagem começa dizendo que houve um erro sintático não tratado, depois diz que o fechamento de parênteses foi inesperado. Por fim, indica que aconteceu no arquivo `extras.js`, na linha 1 e na coluna 31, exatamente a posição que nosso editor havia marcado com o sublinhado.

A mensagem de erro já nos mostra todo o caminho para resolver o problema. Se o fechamento de parênteses foi inesperado, significa que o interpretador esperava outra coisa. Qual? Então, olhamos para o comando `for` e nos lembramos de que o `for` espera três parâmetros, mas só informamos dois. Faltou colocar o incremento "`index++`" ou "`index = 1`".

Agora, repare que à esquerda da mensagem de erro há um ícone vermelho com um “x”, e que o fundo da mensagem é um **erro fatal**. Nesse caso, o processamento daquele script é interrompido, e veja que não foi executada a linha 5, a qual faz a palavra “teste” ser escrita no console.

Como o código é bem curto, copie-o na sua máquina e execute-o. Corrigindo esse erro, você verá a palavra “teste” ser exibida ao final.

Além do erro fatal, existem os **avisos** (em inglês, *warnings*), mensagens que costumam aparecer em amarelo no console, com um ícone também amarelo, e uma exclamação. Esses avisos merecem atenção para haver melhora do código, mas não impedem o processamento do script.



Diferentemente de erros lógicos e sintáticos, os **erros semânticos** são os mais difíceis de serem identificados, porque eles são indetectáveis por ferramentas, em sua maioria. Acontecem porque a ideia está incorreta, mesmo não havendo erros léxicos ou sintáticos (se houvesse, o processo de interpretação já teria sido interrompido).

Alguns poucos indícios de erros semânticos são detectados, como as variáveis não inicializadas, as variáveis declaradas e não utilizadas, as quais podem lhe ajudar a perceber que você esqueceu de algo, ou utilizou uma variável diferente da planejada, mas o computador não consegue saber qual era a sua intenção e o que pensou na hora de escrever o código. Imagine a seguinte função em seu código:

```
function somaTodosOsNumerosPares (inicio, fim) {  
    let soma = 0;  
  
    for (let index = inicio; index <= fim; index += 1) {  
  
        if (index % 2 === 1) {  
  
            soma += index;  
        }  
  
    }  
  
    return soma;  
}
```

Essa função recebe dois números: o início e o fim. O que ela faz é somar todos os números pares entre início e fim (incluindo ambos) e retornar o resultado. Esse código é interpretado e executado sem problemas. Há algum erro?

Na quarta linha do código, em vez de testar se o número é par, está sendo testado se o número é ímpar, com isso, a lógica fica toda invertida, e a função soma os números ímpares em vez de somar os pares. O processamento não será interrompido, o programa será executado, e o erro só será notado quando a soma errada incomodar alguém ou gerar um prejuízo ao usuário. Um erro de cálculo já ficou famoso por fazer um foguete carregando satélites caríssimos explodir no ar (Baraniuk, 2015, on-line).

Para evitar erros desse tipo, existem os testes automatizados de software, que veremos mais adiante, nos quais você pode escrever uma bateria de testes com o valor esperado para diferentes situações, tentando cobrir as diferentes situações possíveis.

Utilizando o *debugger* no navegador

Quando precisamos encontrar um erro, principalmente semântico, podemos executar o código com detalhes, para tentar identificar onde acontece o comportamento inesperado. É muito comum mandar exibir mensagens de texto em diferentes pontos para descobrir onde parou a execução, ou para descobrir o va-

lor de variáveis (o famoso `console.log`, no Javascript), mas existe uma ferramenta mais profissional a esse objetivo, o **debugger** (traduzido como “depurador”, mas usaremos *debugger*, por ser mais popular).

Os navegadores modernos possuem as ferramentas para desenvolvedores embutidas, como o console, que já vimos. Uma das ferramentas é o *debugger*, colocado na aba “Fonte”, onde temos acesso a todo o código-fonte sendo executado.

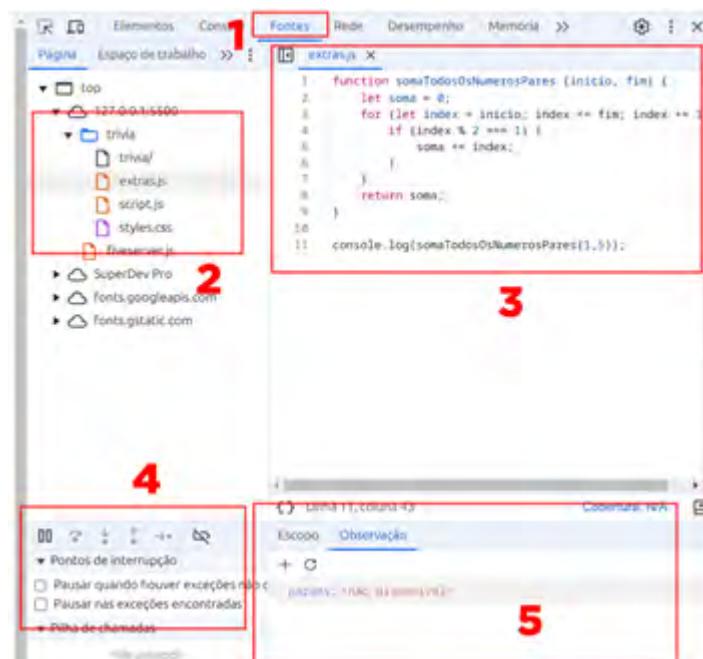
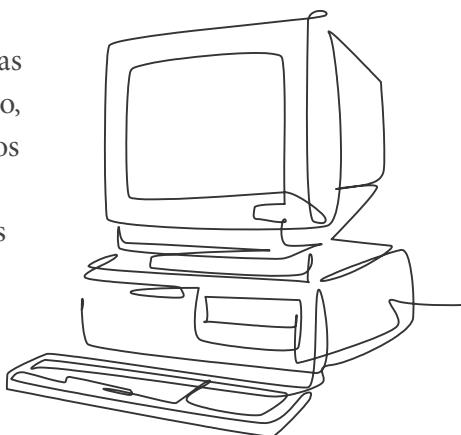


Figura 4 - Área do *debugger* no navegador / Fonte: o autor.

Descrição da Imagem: captura de tela que mostra a aba “Fontes” (1) do *DevTools* no navegador, com cinco seções destacadas por meio de números em cima de cada captura. As seções são: Fontes (1): aba selecionada no *DevTools*, usada para inspecionar e depurar arquivos do projeto. Explorador de Arquivos (2): lista os arquivos do projeto, destacando o arquivo `extras.js`. Editor de Código (3): exibe o código Javascript do arquivo `extras.js`, com uma função chamada `somaTodosOsNumerosPares` que possui um erro lógico ao somar apenas números ímpares. Pontos de Interrupção (4): opções para gerenciar *breakpoints* e pausar a execução em exceções. Painel de Observação (5): exibe variáveis e parâmetros atualmente indisponíveis para inspeção. Fim da descrição.

Na Figura 4, podemos ver como é a interface dessa área de depuração, nesse caso, no Google Chrome. Na aba Fontes (1), vemos, à esquerda, a lista de arquivos do projeto (2) com o código-fonte do arquivo selecionado à direita (3). Abaixo, à esquerda, há opções de gerenciamento da execução (4) com botões para executar, parar e outras opções. No lado direito, um painel de observação (5), onde podemos ver variáveis ou expressões para acompanhar os valores durante a execução.

Ao executar o código, tudo é processado de forma muito rápida, em segundos ou milissegundos. Para conseguir enxergar as alterações durante a execução, precisamos adicionar ***breakpoints***, que são pontos de parada onde a execução segue normal até esses pontos, mas fica suspensa para observarmos o comportamento. Nesse momento, controlamos a execução passo a passo, ou a seguimos normalmente até o final.

Vamos encontrar o erro em nosso código (apesar de já sabermos o que é)? Imagine comigo que você suspeita de que o problema do código esteja no teste condicional. Você, então, se faz as seguintes perguntas:

- Será que o erro está no teste (if)?
- Para quais valores de index a soma está sendo realizada?
- Qual será o valor da soma a cada iteração?

Para resolver essas três questões, você pode efetuar os seguintes passos (acompanhe na Figura 5):

- Coloque um *breakpoint* no corpo do teste condicional, clicando com o mouse à esquerda do número da linha 5.
- Na área “Observação” (ou “*Watcher*”), clique no sinal de “+” e adicione a variável “index”.
- No mesmo sinal de “+”, adicione a variável “soma”.

Não existe um único jeito de achar o erro, você coloca o *breakpoint* onde quiser. Colocando na linha 5, pausa a execução apenas quando o código entrar no bloco do *if*. Agora que tudo está preparado, pode recarregar a página, para que o script seja executado novamente.

Na Figura 5, vemos a situação logo que o script é executado, e o sistema fica suspenso ao executar a linha 5.

```

function somaTodosOsNumerosPares (inicio, fim) {
    let soma = 0;
    for (let index = inicio; index <= fim; index += 1) {
        if (index % 2 === 1) { index = 1
            soma += index;
        }
    }
    return soma;
}

console.log(somaTodosOsNumerosPares(1,5));

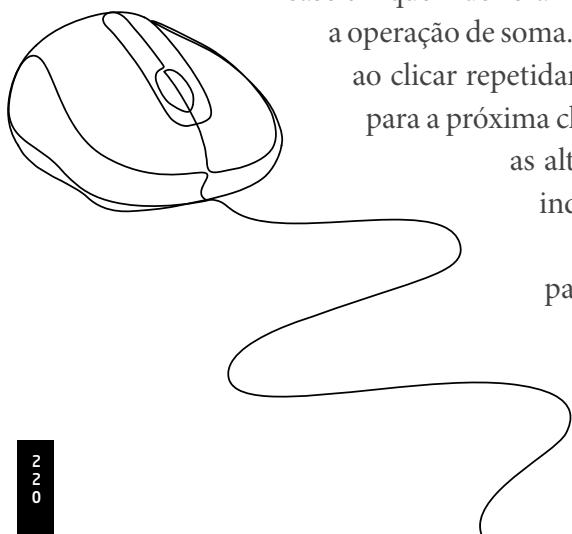
```

Figura 5 - Debugger do navegador em execução com um *breakpoint* / Fonte: o autor.

Descrição da Imagem: captura de tela mostra a aba “Fontes” do DevTools com um ponto de interrupção ativado no código Javascript. As seções principais são: Explorador de Arquivos: lista os arquivos do projeto, destacando extras.js. Editor de Código: exibe a função somaTodosOsNumerosPares, com a linha soma += index destacada, indicando que a execução foi pausada nesse ponto. Painel de Depuração: indica que a execução está pausada no ponto de interrupção. Variáveis em Tempo de Execução: mostra os valores das variáveis index = 1 e soma = 0. Fim da descrição.

Note que o *debugger* já mostra que o valor de index é 1, ou seja, entrou no caso em que index era ímpar, e a soma é 0 porque ainda não efetuou a operação de soma. Você pode seguir a execução passo a passo ao clicar repetidamente no segundo botão do painel (pular para a próxima chamada), no qual verá a soma ser efetuada, as alterações do index e para quais valores de index a soma é efetuada.

Assim, perceba que a soma é feita apenas para os números ímpares. Você pode retirar o nosso *breakpoint* e retornar ao editor, para fazer a correção.



TESTES AUTOMATIZADOS DE SOFTWARE

Sempre que escrever um programa, inevitavelmente, você pode realizar alguns mínimos testes manualmente para ver se ele funcionou ou não.

Ao criar o usuário “Testivaldo”, é possível colocar valores financeiros fictícios e, assim, ver se está ou não gravando os dados.

Você que criou o programa, porém, está enviesado a testar o funcionamento que dá certo, como se o usuário não fosse tentar digitar letras em um CPF, ou que “nunca o usuário comprará uma quantidade negativa de camisetas”.

Além de ser uma boa prática colocar outra pessoa para testar o seu sistema, os **testes automatizados** permitem realizar uma bateria de testes a qualquer momento, podendo ser executados a cada alteração do código, por exemplo, algo que você não faria toda vez, manualmente.

Você pode escrever os seus testes após criar os seus códigos, ou então, em uma abordagem que ficou popular no desenvolvimento de software, escrever os testes antes de escrever o código, o que é conhecido como **TDD (Test Driven Development)**. Nessa abordagem, descreva o comportamento desejado do software em forma de testes, então faça o desenvolvimento até que todos os testes sejam satisfeitos (Aniche, 2015).

Testes de unidade

Uma tendência natural de sistemas é que eles cresçam em complexidade e em quantidade de código, porém é recomendado escrever o código em pequenas partes, com menos complexidade, então, conectar essas partes umas às outras. A partir daí, é possível realizar testes mais simples em pequenas partes, pois, caso elas estejam realmente funcionando, é mais fácil garantir que o todo funcione bem.

Os **testes de unidade** são testes realizados nas menores unidades de um código, geralmente, as funções que escrevemos. Esses são os testes mais baratos de serem realizados, seja em esforço, seja em processamento ou complexidade.

Imagine um sistema de uma calculadora. Utilizando testes de unidade, é possível testar cada operação da calculadora individualmente. Isso é feito escrevendo uma série de operações, como “1 + 1”, “20 - 3” e “12/4”, comparando os resultados aos resultados esperados “2”, “17” e “3”, respectivamente. Caso todos os testes retornem o resultado esperado, não significa que o sistema funciona para todos os casos possíveis, mas já indica que realiza o esperado dentro da bateria de testes.

Jest para testes de unidade em Javascript

Um dos sistemas mais conhecidos e utilizados de testes de unidade em Javascript é o **Jest**. Para utilizá-lo, passaremos rapidamente por alguns outros conceitos.

As principais linguagens de programação modernas têm sistemas de **gerenciamento de dependências**, que são catálogos de bibliotecas escritas para a linguagem, de onde é possível reutilizar inúmeros sistemas prontos em sua programação.

Para Javascript, há a possibilidade de usar o **npm** (*Node Package Manager*), que é o sistema de repositório de pacotes padrão do **Node.js**, um ambiente de execução Javascript em linha de comando, baseado no motor V8 do Chrome.

Para isso, utilize um pouco do terminal do seu sistema operacional (pode usar o PowerShell no Windows, ou o terminal do MacOS, ou Linux). Com o npm instalado na máquina, abra o terminal na pasta de seu projeto e digite:

```
npm init
```

De forma interativa, o sistema lhe pedirá informações, como nome do pacote, autor etc. Nesse momento, não importam muito esses detalhes, é possível seguir com as opções padrão. Depois, digite o seguinte:

```
npm install -D jest
```

Ao fazer isso, o Jest será instalado em seu projeto, e você verá que, após os dois comandos, foram criados os arquivos “package.json”, “package.lock” e a pasta “node_modules”, onde ele baixa as dependências.

EU INDICO

Para sistemas mais completos, o ideal é iniciar o gerenciamento de pacotes com “npm init”, onde há uma configuração interativa de seu projeto, definindo nome, versão etc., porém isso também pode ser feito manualmente, editando diretamente o arquivo “package.json”. Não será detalhado o processo neste material, mas você pode aprender mais a partir do material em: <https://www.youtube.com/watch?v=GnXkpTWobF8>

Abra o arquivo “package.json” em seu editor e altere a linha de scripts > test, deixando da seguinte forma:

```
"scripts": {  
  "test": "jest"  
},
```

Agora, escreva seus testes em Jest. Como você está usando um arquivo chamado **extras.js**, crie um arquivo com o nome **extras.test.js**. No arquivo extras.js, exporte a função criada, colocando uma última linha com o conteúdo:

```
module.exports = somaTodosOsNumerosPares;
```

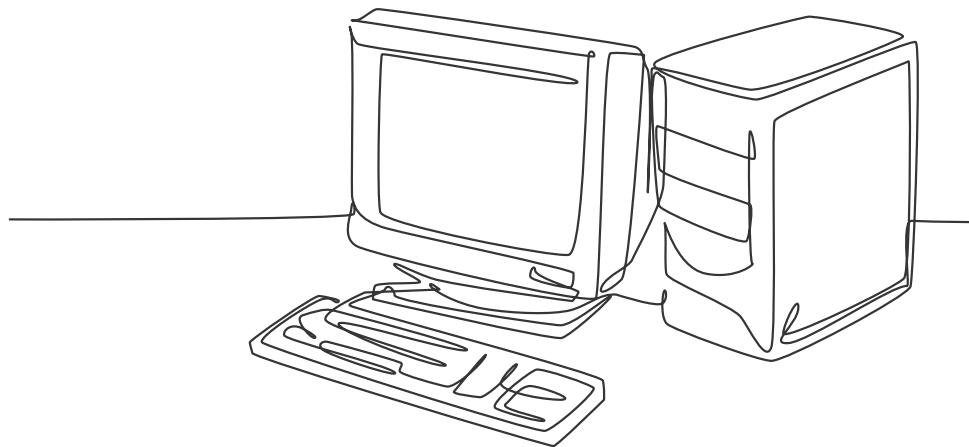
Então, crie o arquivo de testes. Estes são escritos a partir de uma execução da função, comparando com o resultado esperado.

Veja um arquivo completo, com um único teste. Em seguida, será feita a análise linha a linha, para entender o funcionamento.

```
const somaTodosOsNumerosPares = require('./extras');

describe("Testa a soma de todos os números pares", () => {
    it("Soma os pares dado um início e um fim", () => {
        expect(somaTodosOsNumerosPares(1, 5)).toBe(6);
    });
});
```

Na primeira linha, é importada a função “somaTodosOsNumerosPares” que foi escrita no outro arquivo, indicado na função require. Note que não foi colocada a extensão “js” no nome do arquivo.



Depois, o comando “describe” serve apenas para escrever um rótulo na tela e agrupar um conjunto de testes. Isso é muito útil quando são escritos diversos testes para um arquivo, fica mais organizado e ajuda a entender a execução. Repare que o segundo parâmetro é uma função, e os testes vêm dentro dela.

Em seguida, no comando “it” (que também pode ser escrito como “test”, eles são sinônimos aqui) é onde a mágica acontece. Esse comando define os testes propriamente ditos. Você escreve um comando “it” para cada teste, porém pode

testar diversas coisas em um mesmo teste (ficará mais nítido no exemplo mais completo). Também tem uma função como segundo parâmetro.

Dentro dessa função, é chamada a execução da função a ser testada dentro do “expect”, então, é comparada ao resultado esperado usando *matchers*, nesse caso, o “toBe” (Usando [...], 2021, on-line). Consulte sempre a referência para conhecer os *matchers* apropriados para cada situação.

Então, execute o teste, com o comando “npm test”, no terminal. A saída será similar ao que mostra a Figura 6.

```

> trivia@1:0:0 test
> jest
FAIL  extras.test.js
  Testa a soma de todos os números pares
    * Soma os pares dado um inicio e um fim (3 ms)

    * Testa a soma de todos os números pares → Soma os pares dado um inicio e um fim
      expect(received).toEqual(expected) // Object.is equality

      Expected: 6
      Received: 9

        ||| describe("Testa a soma de todos os números pares", () => {
          it('Soma os pares dado um inicio e um fim', () => {
            expect(somaTodosOsNumerosPares(1, 5)).toEqual(6)
          })
        });
      at Object. (extras.test.js:5:47)

Test Suites: 1 Failed, 1 total
Tests:       1 Failed, 1 total
Snapshots:  0 total
Time:        0.276 s, estimated 1 s
 Ran all test suites

```

Figura 6 – Tela de execução do Jest / Fonte: o autor.

Descrição da Imagem: captura de tela que mostra a saída do Jest executando um teste unitário no arquivo extras.test.js, onde um teste falhou. As principais informações são: Indicação de falha: o Jest marca o teste como FAIL. Descrição do teste: verifica a função somaTodosOsNumerosPares, esperando que a soma dos pares entre 1 e 5 retorne 6. Erro na expectativa: o valor esperado era 6, mas o recebido foi 9, indicando um erro na implementação da função. Código do teste: usa expect(somaTodosOsNumerosPares(1, 5)).toEqual(6);, mas o resultado não corresponde ao esperado. Resumo da execução: test Suites: 1 falhou, 1 no total. Tests: 1 falhou, 1 no total. Tempo de execução: 0.276s. A falha sugere um erro lógico na função testada, possivelmente, na filtragem dos números pares. Fim da descrição.

Percebe-se, na figura, que o teste falhou, pois ele esperava que a função retornasse 6 e ela retornou 9. Mais uma vez, nota-se que a função está somando os números ímpares em vez de somar os números pares, portanto, é preciso voltar ao código e corrigir o teste para “if (index % 2 === 0)”. Assim, ao realizar novamente o teste, será recebida uma mensagem apontando que ele passou com sucesso.

Há a possibilidade, então, de estender o teste da seguinte forma:

```
const somaTodosOsNumerosPares = require('./extras');

describe("Testa a soma de todos os números pares", () => {
    it("Soma os pares dado um início e um fim", () => {
        expect(somaTodosOsNumerosPares(1, 5)).toBe(6);
        expect(somaTodosOsNumerosPares(1, 10)).toBe(30);
        expect(somaTodosOsNumerosPares(10, 13)).toBe(22);
        expect(somaTodosOsNumerosPares(-1, 1)).toBe(0);
        expect(somaTodosOsNumerosPares(-1, 5)).toBe(6);
        expect(somaTodosOsNumerosPares(-5, 0)).toBe(-6);
    });
    it("Retorna zero caso não tenha dois parâmetros", () => {
        expect(somaTodosOsNumerosPares()).toBe(0)
        expect(somaTodosOsNumerosPares(1)).toBe(0);
    });
});
```

Execute esse teste para observar como ele é avaliado. Ele representa dois testes, mesmo executando várias operações. Se alguma operação falhar, o teste correspondente também falhará, indicando o erro. Você pode inserir valores incorretos para verificar o comportamento.

Outros tipos de testes automatizados

Após realizar os testes de unidade, que são os mais simples, existem diversos testes que podem ser realizados. Após escrever diferentes módulos e eles serem testados em suas unidades, é possível realizar **testes de integração**, para verificar

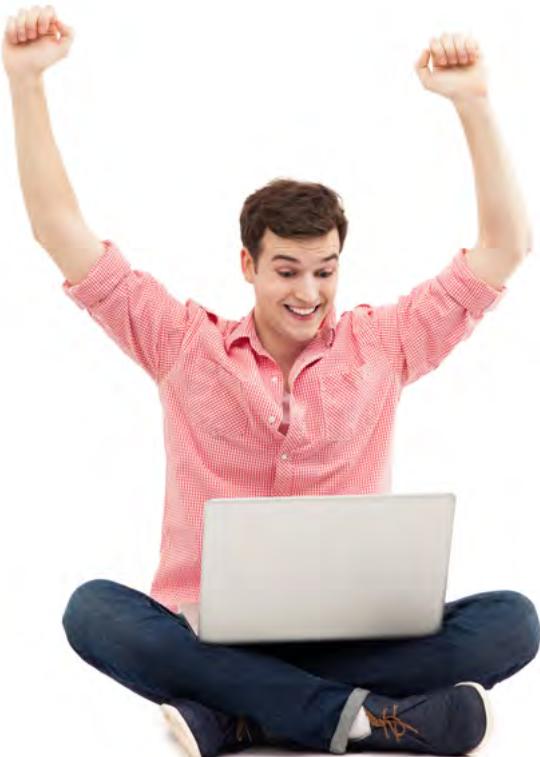
se as partes funcionam conforme o esperado quando colocadas para funcionar em conjunto. Isso é importante, porque as partes precisam ser integradas de forma coerente.

Há a chance de simular o comportamento do usuário com um **teste end-to-end (E2E)**, também chamado de teste ponta a ponta, no qual é possível simular a experiência na interface, no preenchimento de formulários, login etc. Tanto para o teste end-to-end, quanto para os testes de integração, é comum o uso do **cypress** em Javascript.

Ainda, há a possibilidade de **testes de acessibilidade**, para garantir que pessoas com as mais diferentes necessidades possam ter acesso ao sistema. Uma ferramenta bastante usada é o **Lighthouse**, do Google, que também serve para realizar **testes de performance**, visando à otimização do processamento.

Por fim, existem os **testes de aceitação**, que podem ser realizados manualmente por pessoas, para captar a recepção dos sistemas por usuários reais, ou de formas automatizadas, aproveitando os testes E2E.

Quando falamos em testes de software, vários cenários precisam ser levados em consideração, visando um produto com corretude, além de uma boa usabilidade para ter, também, boa aceitação pelos usuários.



EM FOCO

Assista à videoaula para mais informações e conhecimentos sobre este tema de aprendizagem. Recursos de mídia disponíveis no conteúdo digital do ambiente virtual de aprendizagem.

NOVOS DESAFIOS

No universo do desenvolvimento front-end, a ponte entre teoria e prática é o que transforma conhecimento em competência profissional. Ao dominar conceitos de depuração e ferramentas do navegador, você não apenas resolve erros de código, mas adquire a agilidade necessária para atuar em ambientes ágeis, onde otimizar tempo e garantir eficiência são prioridades. A prática com testes automatizados, especialmente a criação de testes unitários, vai além do exercício técnico: é uma resposta direta às demandas do mercado por aplicações robustas e escaláveis, que exigem confiabilidade em cada etapa de desenvolvimento.

O gerenciamento de pacotes via npm e a compreensão dos diferentes tipos de testes (unitários, de integração, E2E) refletem habilidades essenciais para integrar-se a fluxos de trabalho modernos, como *pipelines* de CI/CD e equipes multidisciplinares. Essas competências não apenas atendem às exigências atuais, mas também preparam você para tendências emergentes, como a automação avançada e a colaboração em projetos distribuídos.

No ambiente profissional, a capacidade de unir fundamentos teóricos à aplicação prática é o que diferencia um desenvolvedor adaptável. Cada linha de código testada, cada erro depurado e cada pacote gerenciado são passos para enfrentar desafios complexos, desde a manutenção de sistemas legados até a construção de soluções inovadoras.

O futuro do front-end pede profissionais que não apenas acompanhem as mudanças, mas as antecipem. É nesse equilíbrio entre conhecimento técnico e visão estratégica que suas próximas oportunidades surgirão. Prepare-se: suas habilidades, agora, são a base para transformar desafios em conquistas.



VAMOS PRATICAR

1. "Testes de software são o processo de avaliar e verificar se um produto ou aplicativo de software faz o que deveria fazer. Os benefícios de bons testes incluem a prevenção de bugs e a melhoria do desempenho" (O que [...], 2024, on-line).

Qual ferramenta é amplamente utilizada para testes end-to-end (E2E) em front-end?

- a) Jest
- b) Cypress
- c) ESLint
- d) Webpack
- e) Babel

2. Os testes unitários são apenas uma parte de uma estratégia de teste mais ampla, que também pode incluir testes de integração, de sistema, de aceitação do usuário e outros. Eles são, no entanto, uma ferramenta valiosa para garantir a qualidade do código e reduzir a quantidade de *bugs*.

Considerando o texto apresentado, avalie as afirmações, a seguir, a respeito de testes de software:

- I - Testes unitários verificam componentes isoladamente.
- II - Testes de integração validam a comunicação entre módulos.
- III - Testes end-to-end (E2E) simulam interações do usuário em fluxos completos.

É correto o que se afirma em:

- a) I, apenas.
- b) III, apenas.
- c) I e II, apenas.
- d) II e III, apenas.
- e) I, II e III.

VAMOS PRATICAR

3. "O principal objetivo dos testes unitários é garantir que cada componente do software funcione conforme o esperado, validando seu comportamento em diferentes cenários e detectando erros de forma precoce no ciclo de desenvolvimento" (Normando, 2024, on-line).

Durante a execução de um teste com Jest, um desenvolvedor encontra a mensagem "Expected value to be 5, but received 4". O que isso indica?

- a) O teste está verificando um valor errado.
- b) A função testada retornou um valor inesperado.
- c) O Jest não suporta esse tipo de teste.
- d) O erro ocorre porque Jest não permite valores numéricos nos testes.
- e) A configuração do Jest está incorreta.

REFERÊNCIAS

AHO, A. V. et al. **Compiladores**: princípios, técnicas e ferramentas. 2. ed. São Paulo: Pearson Addison-Wesley, 2008.

ANICHE, M. **Testes automatizados de software**: um guia prático. São Paulo: Casa do Código, 2015.

BARANIUK, C. As falhas numéricas que podem causar desastres. **BBC News Brasil**, [s. l.], 14 maio 2015. Disponível em: https://www.bbc.com/portuguese/noticias/2015/05/150513_vert_fut_bug_digital_ml. Acesso em: 10 fev. 2025.

NORMANDO, C. Testes unitários. **Medium**, [s. l.], 4 mar. 2024. Disponível em: <https://medium.com/@celionormando/testes-unit%C3%A1rios-5bb55a9b4e83>. Acesso em: 10 fev. 2025.

O QUE é teste de software? **IBM**, [s. l.], 17 jan. 2024. Disponível em: <https://www.ibm.com/br-pt/topics/software-testing>. Acesso em: 10 fev. 2025.

SELIM, R. O primeiro “bug” de computador: uma história curiosa da tecnologia. **Folha de Pernambuco**, Recife, 24 abr. 2024. Disponível em: <https://www.folhape.com.br/colunistas/tecnologia-e-games/o-primeiro-bug-de-computador-uma-historia-curiosa-da-tecnologia/43431/>. Acesso em: 7 fev. 2025.

USANDO Matchers. **Jest**, [s. l.], 2021. Disponível em: <https://jestjs.io/pt-BR/docs/using-matchers>. Acesso em: 10 fev. 2025.

CONFIRA SUAS RESPOSTAS

1. Alternativa B.

Cypress é especializado em testes E2E, simulando interações reais do usuário.

A alternativa A está incorreta, pois Jest é focado em testes unitários.

A alternativa C está incorreta, pois ESLint é uma ferramenta de Lint.

A alternativa D está incorreta, pois Webpack não é ferramenta de testes E2E.

A alternativa E está incorreta, pois Babel não é ferramenta de testes E2E.

2. Alternativa E.

A afirmação I está correta, pois testes unitários verificam componentes isoladamente.

A afirmação II está correta, pois testes de integração validam a comunicação entre módulos.

A afirmação III está correta, pois testes end-to-end (E2E) simulam interações do usuário em fluxos completos.

3. Alternativa B.

Esse erro ocorre quando a saída real não corresponde ao valor esperado pelo teste.

A alternativa A está incorreta, pois a mensagem indica que o teste encontrou um resultado diferente do esperado, mas isso não significa automaticamente que o teste esteja errado.

A alternativa C está incorreta, pois o teste é suportado, apenas o valor está diferente.

A alternativa D está incorreta, pois o Jest permite valores numéricos.

A alternativa E está incorreta, pois a mensagem não tem a ver com a configuração.