Jornada Tech – Oficina “Desenvolvimento WEB”

1. Internet e Comunicação Digital

1.1. Conceito de Internet – Definição e importância

2. Protocolo HTTP/HTTPS

2.1. O que é protocolo e por que é necessário
2.2. Funcionamento de uma requisição e resposta HTTP
2.3. Métodos HTTP principais: GET, POST, PUT, DELETE
2.4. HTTPS e segurança (SSL/TLS)

3. Funcionamento de um Site

3.1. Cliente vs. Servidor
3.2. DNS: como nomes de domínio viram endereços IP
3.3. Ciclo de vida de uma requisição web
3.4. Componentes de um servidor web (Apache, Nginx)

4. Navegadores Web

4.1. O que é e para que serve um navegador
4.2. Principais exemplos (Chrome, Firefox, Edge, Safari)
4.3. Motor de renderização (Blink, Gecko, WebKit)
4.4. Ferramentas de desenvolvedor (DevTools)

5. HTML: Estrutura de Conteúdo

5.1. Introdução ao HTML e sua semântica
5.2. Tags, atributos e elementos fundamentais
5.3. Documento HTML mínimo
5.4. Exemplo comentado de página simples

6. CSS: Estilo e Layout

6.1. Papel do CSS na Web
6.2. Sintaxe: seletores, propriedades e valores
6.3. Modelos de inclusão: inline vs. internal vs. external
6.4. Box model e posicionamento básico
6.5. Exemplos de regras de estilo

7. JavaScript – Interatividade e Comportamento

7.1. O que é JavaScript no contexto web
7.2. Inserção de scripts em HTML
7.3. Manipulação do DOM (Document Object Model)
7.4. Eventos básicos: click, load, submit
7.5. Exemplo de função simples de interação

8. Resumo dos Papéis das Três Tecnologias

8.1. Comparativo de responsabilidades
8.2. Fluxo de trabalho integrado: HTML → CSS → JavaScript

9. Conclusão

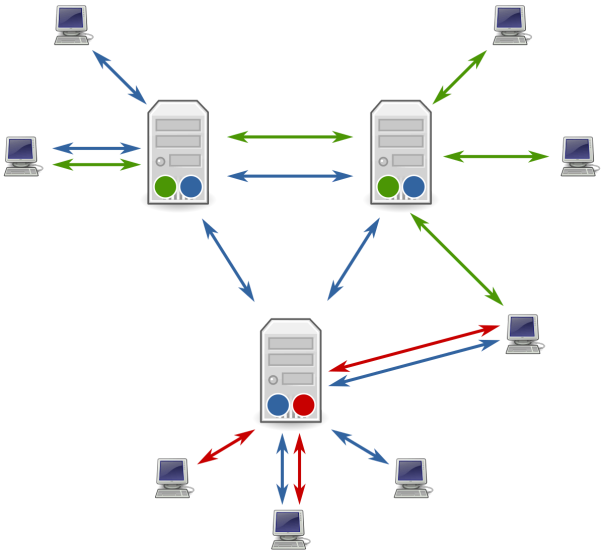
9.1. Por que é importante separar estrutura, estilo e comportamento?
9.2. Como a segurança (HTTPS) impacta a experiência do usuário?
9.3. Quais são os desafios de renderização em diferentes navegadores?

Referências Bibliográficas

1.1 Conceito de Internet – Definição e importância

A Internet revolucionou a maneira como a sociedade se conecta, comunica e acessa informações, tornando-se essencial para quase todas as atividades modernas. Trata-se de um sistema complexo e globalizado que impacta profundamente o cotidiano das pessoas, empresas e governos.

A Internet é uma rede global formada por milhões de dispositivos digitais interconectados que possibilitam a comunicação e a troca de informações por meio de protocolos padrão, como o TCP/IP. A estrutura da Internet inclui desde computadores pessoais e smartphones até grandes servidores de dados interligados por meio de redes locais (LAN) e redes remotas (WAN) (Kurose & Ross, 2021).



Esquema de conexão de servidores e clientes na Internet.

Entre os inúmeros benefícios da Internet, destacam-se três razões fundamentais:

1. Comunicação instantânea: A Internet permite comunicação imediata através de e-mails, mensagens instantâneas, chamadas de voz e videoconferências, facilitando interações pessoais e profissionais independentemente da distância física (Cisco, 2022).
2. Democratização do conhecimento: Oferece amplo acesso a informações educacionais e científicas, permitindo que usuários em todo o mundo tenham acesso a cursos online, pesquisas acadêmicas e plataformas de aprendizado gratuito, como Khan Academy e Coursera (Pew Research Center, 2023).
3. Impulso econômico e inovação: Atua como base para o comércio eletrônico, impulsionando novas economias digitais e startups tecnológicas. Estima-se que a economia digital represente hoje cerca de 16% do PIB mundial, tornando-se um motor vital para a inovação e desenvolvimento econômico (UNCTAD, 2023).

2. Protocolo HTTP/HTTPS

2.1 O que é protocolo e por que é necessário

Um protocolo pode ser definido como um conjunto padronizado de regras que permitem que dispositivos digitais comuniquem-se de forma eficiente e compreensível. É fundamental porque garante que equipamentos diferentes, como computadores, smartphones e servidores, consigam trocar informações corretamente, mesmo possuindo arquiteturas e sistemas operacionais distintos (Tanenbaum & Wetherall, 2022).

Os protocolos estabelecem formatos específicos para mensagens, controlam sequências de troca de dados e gerenciam erros e interrupções. Dessa forma, são essenciais para garantir a integridade e confiabilidade das comunicações digitais (Kurose & Ross, 2021).

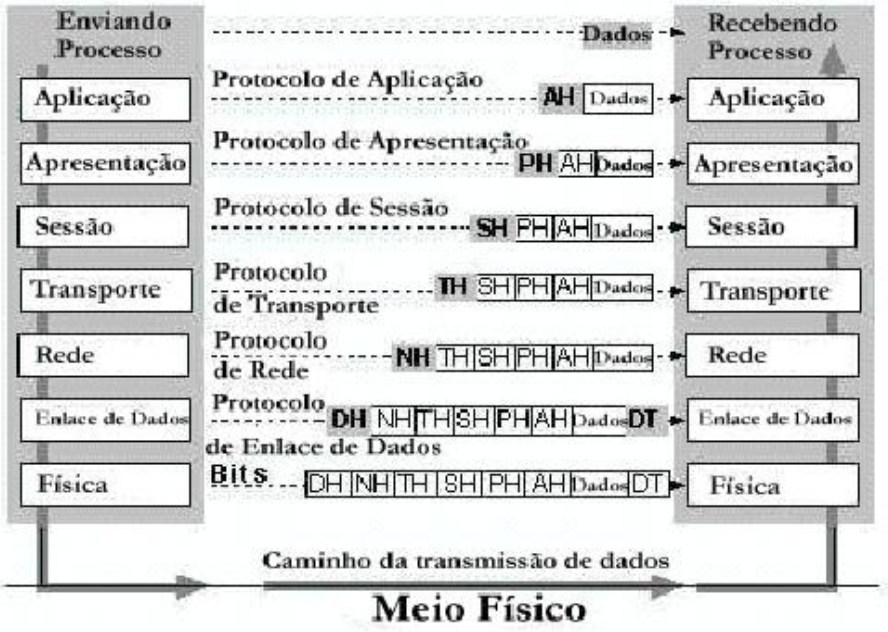
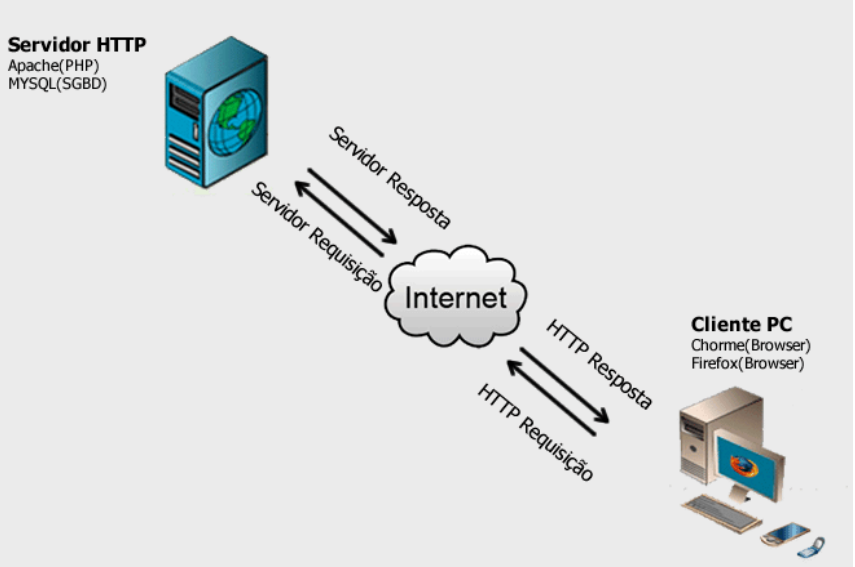


Diagrama de camadas de protocolo, mostrando onde o HTTP opera.

2.2 Funcionamento de uma requisição e resposta HTTP

O HTTP (Hypertext Transfer Protocol) opera no modelo cliente-servidor, utilizando o TCP/IP como base. O processo típico ocorre em etapas:

- Requisição: O cliente (geralmente um navegador) estabelece uma conexão TCP com o servidor e envia uma requisição HTTP contendo método, URL e informações adicionais.
- Processamento: O servidor recebe a requisição, processa a informação solicitada e prepara uma resposta.
- Resposta: O servidor envia uma mensagem HTTP contendo um código de status, cabeçalhos informativos e os dados requisitados (como uma página HTML ou JSON).
- Finalização: Após a resposta, a conexão pode ser fechada ou mantida aberta para novas requisições (Fielding & Reschke, 2014).



Fluxograma de requisição e resposta HTTP.

2.3 Métodos HTTP principais: GET, POST, PUT, DELETE

Métodos HTTP definem ações específicas solicitadas ao servidor:

1. **GET**: Utilizado para requisitar recursos ou dados específicos do servidor, sem alterá-los.
   Exemplo: Carregar uma página web ou buscar um registro de usuário.
2. **POST**: Utilizado para enviar dados ao servidor, frequentemente criando novos recursos.
   Exemplo: Submissão de formulários ou envio de informações pessoais em cadastros.
3. **PUT**: Atualiza completamente um recurso existente no servidor com os dados enviados pelo cliente.
   Exemplo: Atualização total dos dados de um usuário.
4. **DELETE**: Remove um recurso específico do servidor.
   Exemplo: Exclusão de uma conta ou registro específico (Mozilla Developer Network, 2024).

Esses métodos permitem operações CRUD (Create, Read, Update, Delete) padronizadas em aplicações web modernas.

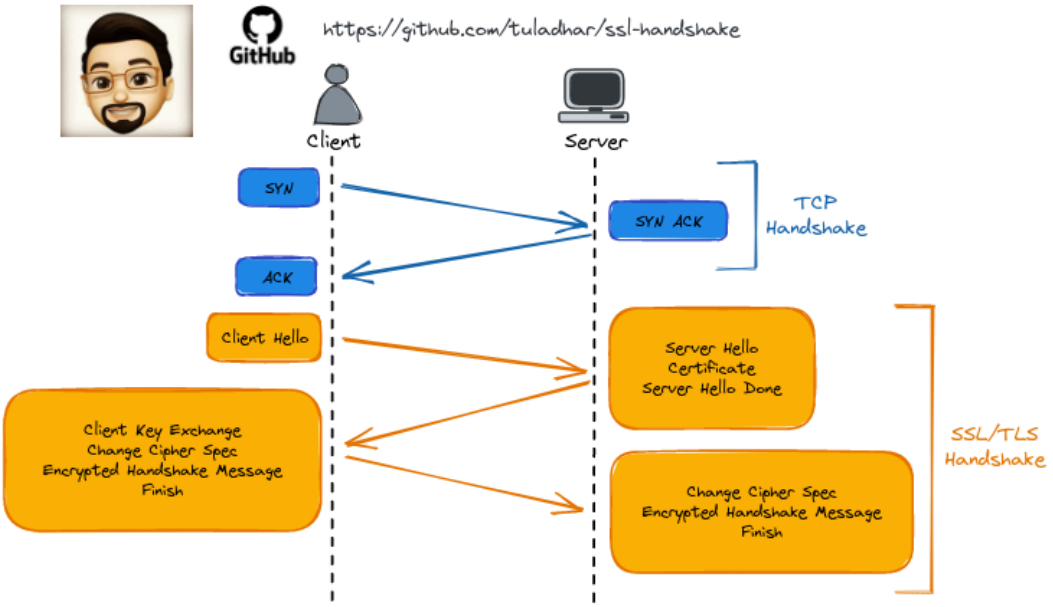
2.4 HTTPS e segurança (SSL/TLS)

HTTPS (Hypertext Transfer Protocol Secure) é a versão segura do HTTP. Ele utiliza protocolos criptográficos SSL (Secure Sockets Layer) ou TLS (Transport Layer Security) para proteger os dados durante sua transmissão, prevenindo interceptação e modificações não autorizadas (Rescorla, 2018).

- O processo seguro envolve o chamado handshake SSL/TLS:
- O cliente inicia a conexão com o servidor HTTPS.

- O servidor responde com seu certificado digital.
- O cliente verifica o certificado e gera uma chave criptográfica compartilhada.

Ambos cliente e servidor estabelecem uma conexão segura criptografada, permitindo troca segura de informações sensíveis, como senhas e dados financeiros (Dierks & Rescorla, 2008).

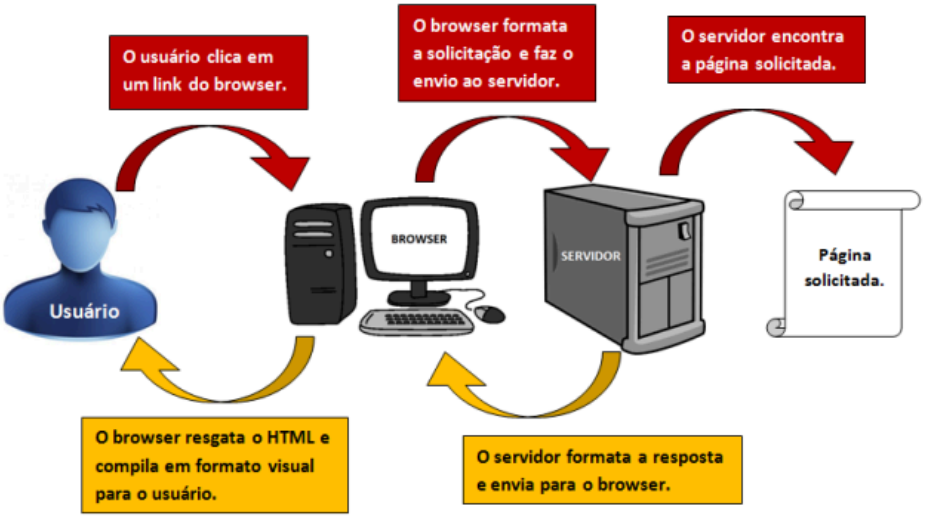


Esquema de handshake SSL/TLS em HTTPS (fonte: P

3. Funcionamento de um Site

3.1 Cliente vs. Servidor

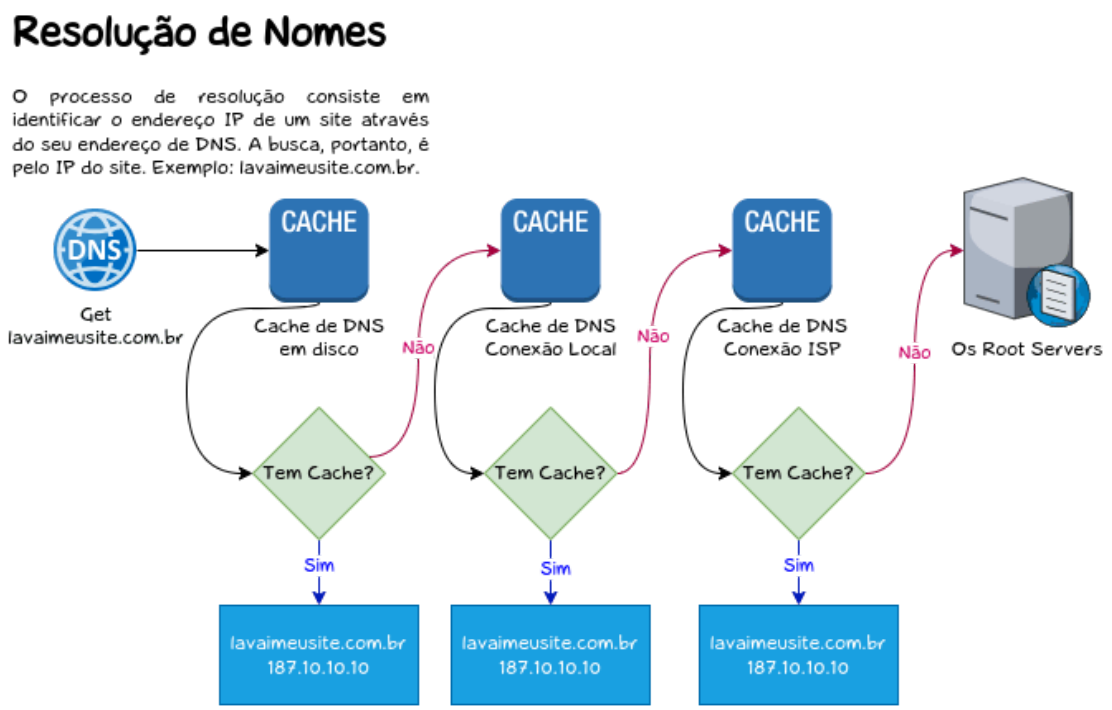
O cliente é normalmente o navegador do usuário; ele envia requisições HTTP/HTTPS para obter recursos como HTML, CSS, JavaScript e imagens. O servidor armazena esses arquivos, processa a lógica de aplicação (quando necessário) e responde com o conteúdo solicitado, acompanhado de cabeçalhos que descrevem tipo de mídia, cache e status (Kurose & Ross, 2021). Essa troca ocorre sobre o protocolo TCP/IP, garantindo entrega confiável dos dados.



3.2 DNS: como nomes de domínio viram endereços IP

Quando o usuário digita www.exemplo.com, o navegador consulta o DNS (Domain Name System) para obter o endereço IP correspondente. O processo segue esta ordem (Mockapetris, 1987):

1. Verifica o cache local do sistema operacional.
2. Consulta o resolver configurado (normalmente o servidor DNS do provedor).
3. Se não houver resposta em cache, o resolver pergunta a um servidor raiz, que devolve os servidores de top-level domain (.com, .org).
4. O resolver pergunta ao TLD, que aponta para o servidor autoritativo do domínio.
5. O autoritativo retorna um registro A (IPv4) ou AAAA (IPv6) contendo o IP; alternativamente, pode devolver um CNAME, que redireciona a nova busca.



3.3 Ciclo de vida de uma requisição web

1. URL digitada: o navegador identifica protocolo, domínio e caminho.
2. Resolução DNS: converte o domínio em IP, conforme passo 3.2.
3. Handshake TCP: cliente e servidor trocam pacotes SYN/SYN-ACK/ACK para estabelecer conexão confiável (Stevens, 2011).
4. TLS handshake (apenas em HTTPS): negociação de chaves e certificados.
5. Requisição HTTP: navegador envia método (GET, POST), cabeçalhos e, se necessário, corpo.
6. Processamento no servidor: leitura de arquivos estáticos ou execução de código (PHP, Node, Python).
7. Resposta HTTP: servidor devolve cabeçalhos (ex.: 200 OK, Content-Type: text/html) e corpo.
8. Renderização: o navegador constrói a árvore DOM, aplica CSS, executa JavaScript e exibe a página (Grigorik, 2013).

3.4 Componentes de um servidor web (Apache, Nginx)

Um servidor web aceita conexões, interpreta a requisição, decide como responder e envia os dados de volta (Buytaert, 2020). Entre os mais usados:

Aspecto	Apache HTTP Server	Nginx
Arquitetura	Processos/threads por conexão (prefork, worker, event)	Event-driven assíncrono
Desempenho static	Bom, pode consumir mais memória sob alta concorrência Excelente	Excelente, lida melhor com milhares de conexões simultâneas
Módulos	Ampla coleção (mod_php, mod_wsgi) recompilável	Módulos menores; muitas vezes requer recompilação
Proxy reverso	Disponível (mod_proxy), mas menos otimizado	Desempenho superior; muito usado para cache e load-balancing
Casos de uso	Hospedagem compartilhada, aplicações que dependem de .htaccess	Sites de alto tráfego, APIs, micro-serviços

Nginx geralmente atua como proxy reverso na frente de aplicações (ex.: Node.js, Django), enquanto o Apache continua popular em provedores com configuração flexível via .htaccess (Nginx Inc., 2022).

4. Navegadores Web

4.1 O que é e para que serve um navegador

Um navegador web é um aplicativo cliente que interpreta HTML, CSS, JavaScript e outros padrões da Web, permitindo ao usuário visualizar páginas, enviar formulários e executar aplicações on-line (MDN Web Docs, 2024). Ele recebe recursos do servidor via HTTP/HTTPS, renderiza a interface gráfica e isola processos para segurança dos dados (Kerrisk, 2023).

4.2 Principais exemplos (Chrome, Firefox, Edge, Safari)

1. Google Chrome – conhecido por alta performance, integração com serviços Google e enorme ecossistema de extensões; lidera com ≈66 % do mercado global em abril 2025 (Statcounter, 2025).
2. Apple Safari – otimizado para macOS/iOS, foco em eficiência energética e privacidade; ocupa ≈17 % (Statcounter, 2025).
3. Microsoft Edge – baseado no projeto Chromium, traz recursos corporativos e integração com Windows; detém ≈5 % (Statcounter, 2025).
4. Mozilla Firefox – engine independente, forte em padrões abertos e extensões; participação em torno de 2,5 % (Statcounter, 2025).

4.3 Motor de renderização (Blink, Gecko, WebKit)

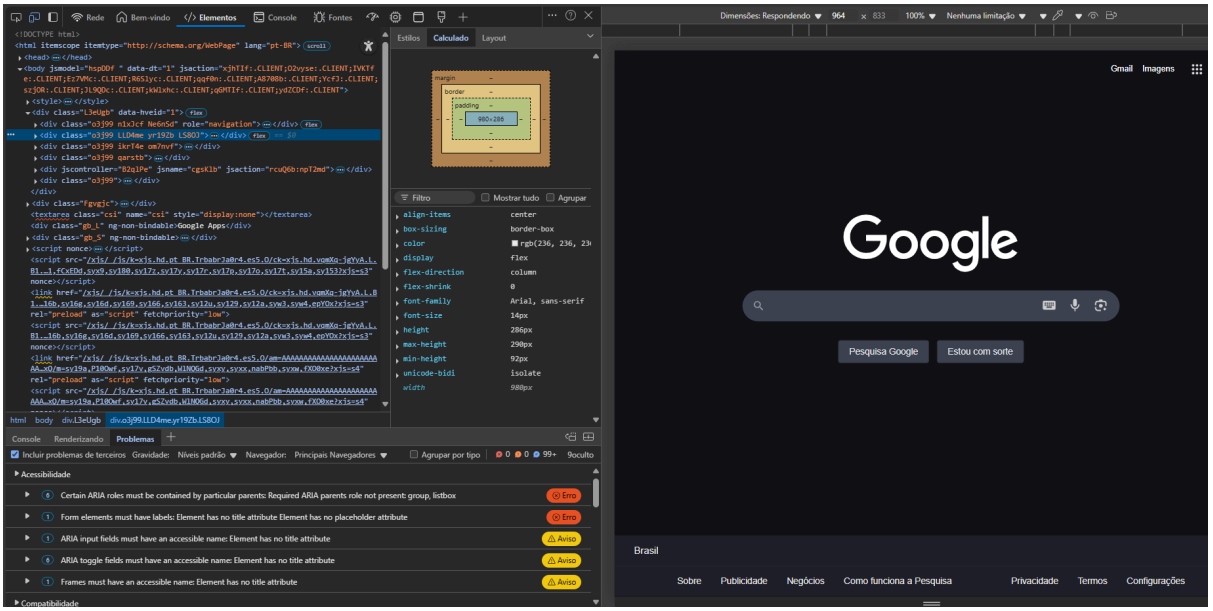
Um motor de renderização transforma código web em pixels na tela: ele constrói a árvore DOM, aplica CSSOM, executa JavaScript e compõe camadas para exibição (O'Donnell, 2022).

- Blink – deriva do WebKit, usa arquitetura "multi-process" para isolar abas; adotado por Chrome, Edge, Opera e Brave (Google Developers, 2024).
- Gecko – engine da Mozilla escrito em C++; introduziu o motor de CSS Servo-Stylo para paralelismo; usado no Firefox (Mozilla, 2024).
- WebKit – mantém pipeline compacto e APIs específicas do iOS; empregado no Safari (Apple, 2024).

4.4 Ferramentas de desenvolvedor (DevTools)

Todos os navegadores modernos incluem DevTools (Ferramentas DEV), um conjunto de painéis que auxiliam no desenvolvimento e na depuração (Google Chrome Docs, 2024):

1. Elements / Inspector – permite editar HTML e CSS em tempo real, facilitando ajustes de layout.
2. Console – executa comandos JavaScript, loga erros e exibe avisos.
3. Network – monitora requisições, tempos de resposta, tamanho de recursos e cabeçalhos HTTP.
4. Performance – grava e analisa uso de CPU, FPS e gargalos de renderização, ajudando a otimizar carregamento.



Tela da DevTools em inspeção de elemento destacando o painel de estilos.

3. HTML: Estrutura de Conteúdo

3.1 Introdução ao HTML e sua semântica

HTML (Hypertext Markup Language) é a linguagem base para a criação de páginas web. Trata-se de uma linguagem de marcação que define a estrutura e o conteúdo de documentos exibidos na web, permitindo indicar claramente a função de cada elemento presente na página (Duckett, 2022).

A semântica em HTML refere-se à utilização correta das tags para indicar o significado do conteúdo, ajudando não apenas navegadores web, mas também tecnologias assistivas e mecanismos de busca (SEO) a interpretar corretamente o documento. Uma boa estrutura semântica melhora a acessibilidade e visibilidade online (MDN Web Docs, 2024).

3.2 Tags, atributos e elementos fundamentais

Em HTML, uma tag é utilizada para marcar e delimitar conteúdos, enquanto os atributos fornecem informações adicionais sobre esses conteúdos. Um elemento é formado por uma tag, seus atributos e o conteúdo interno que ela engloba (W3Schools, 2024).

Exemplos de tags essenciais:

<h1>: Define o título principal da página ou seção.

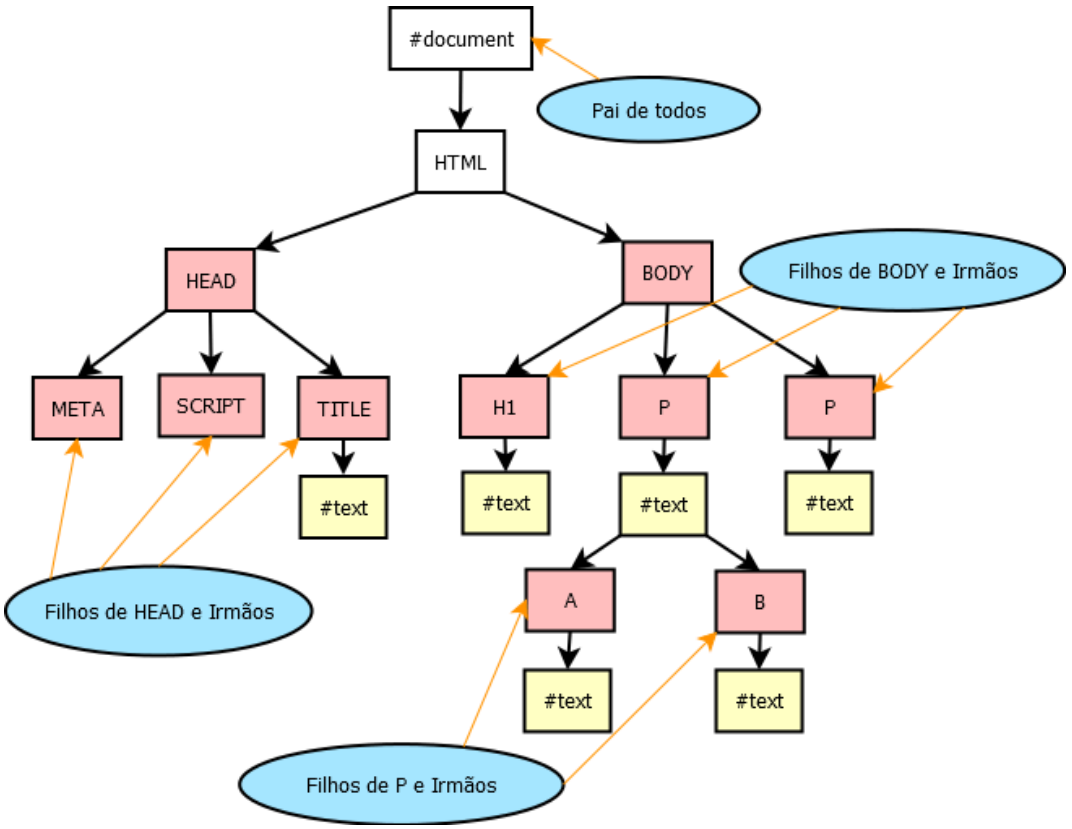
<p>: Marca um parágrafo de texto.

<a>: Cria links para outras páginas ou conteúdos.

<img>: Exibe imagens no documento.

<ul>: Representa listas não ordenadas, com itens marcados por bullets.

Essas tags são a base para criar conteúdo bem estruturado e de fácil compreensão por usuários e ferramentas digitais (Duckett, 2022).



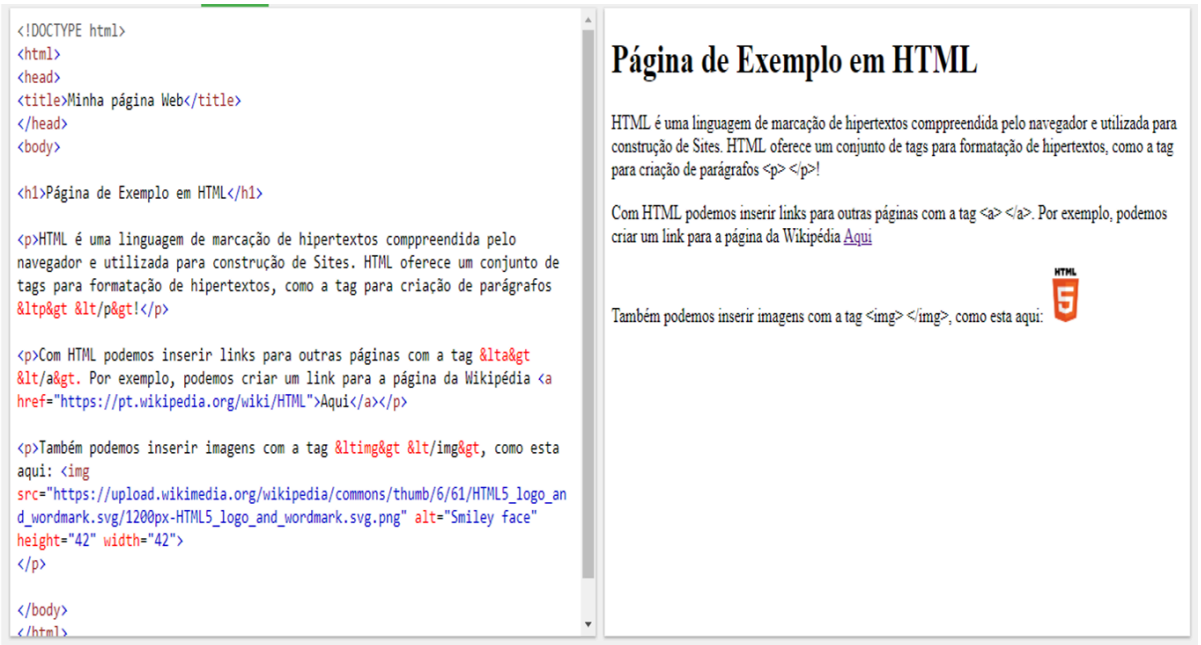
Exemplo de diagrama de árvore DOM mostrando elementos HTML

3.3 Documento HTML mínimo

Um documento HTML básico possui a seguinte estrutura essencial (MDN Web Docs, 2024):

1. <!DOCTYPE html>: Indica ao navegador a versão do HTML utilizada (HTML5 atualmente).
2. <html>: Engloba todo o conteúdo do documento, contendo dois elementos principais:
3. <head>: Inclui informações sobre o documento, como título, metadados e links para folhas de estilo.
4. <body>: Abriga todo o conteúdo visível para o usuário, como textos, imagens, vídeos e links.

Cada seção cumpre um papel claro e importante, garantindo organização e clareza no documento web.



3.4 Exemplo comentado de página simples

Abaixo, temos um exemplo prático de uma página HTML básica:

```
Unset
<!DOCTYPE html> <!-- Declara que este documento é HTML5 -->
<html lang="pt-br"> <!-- Define o idioma da página como português do Brasil
-->

<head>
  <meta charset="UTF-8"> <!-- Define a codificação de caracteres como UTF-8
-->
```

```
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
<!-- Torna a página responsiva -->
  <title>Minha Primeira Página</title> <!-- Define o título exibido na aba
do navegador -->
</head>

<body>
  <h1>Bem-vindo à minha página!</h1> <!-- Título principal -->
  <p>Este é um parágrafo de texto em HTML.</p> <!-- Parágrafo simples -->

  <img src="imagem.jpg" alt="Descrição da imagem"> <!-- Inserção de imagem
com texto alternativo -->

  <ul> <!-- Lista não ordenada -->
    <li>Item 1</li>
    <li>Item 2</li>
    <li>Item 3</li>
  </ul>

  <a href="https://www.exemplo.com">Visite este site</a> <!-- Link externo
-->
</body>

</html>
```

Cada comentário (que começa com ‘<!--’) explica o propósito das linhas, auxiliando na compreensão clara do documento HTML (Duckett, 2022).

4. CSS: Estilo e Layout

4.1 Papel do CSS na Web

O CSS (Cascading Style Sheets) é uma linguagem utilizada para controlar o estilo e a apresentação visual de páginas web. Sua principal função é separar o conteúdo estrutural (HTML) do aspecto visual (cores, fontes, layout), permitindo manutenção simplificada e design consistente (Duckett, 2022).

Essa separação entre conteúdo e estilo facilita o desenvolvimento de layouts responsivos, garantindo que páginas se adaptem a diferentes dispositivos e tamanhos de tela, melhorando a experiência do usuário e a acessibilidade (MDN Web Docs, 2024).

4.2 Sintaxe: seletores, propriedades e valores

Uma regra CSS possui três partes principais: seletor, propriedade e valor, estruturados assim:

```
seletor {
  propriedade: valor;
}
```

Exemplos de seletores simples:

1. Seletor de elemento: p { color: blue; } altera a cor de todos os parágrafos.
2. Seletor de classe: .titulo { font-size: 20px; } aplica tamanho de fonte aos elementos com class="titulo".
3. Seletor de ID: #cabecalho { background-color: gray; } estiliza o elemento único com id="cabecalho" (W3Schools, 2024).

4.3 Modelos de inclusão: inline vs. internal vs. external

CSS pode ser aplicado de três maneiras principais:

1. **Inline** (style): diretamente na tag HTML. Rápido, porém menos escalável e difícil de manter.

```
<p style="color:red;">Texto vermelho</p>
```

2. **Internal** (<style>): no cabeçalho do documento HTML. Prático para pequenas alterações em uma única página.

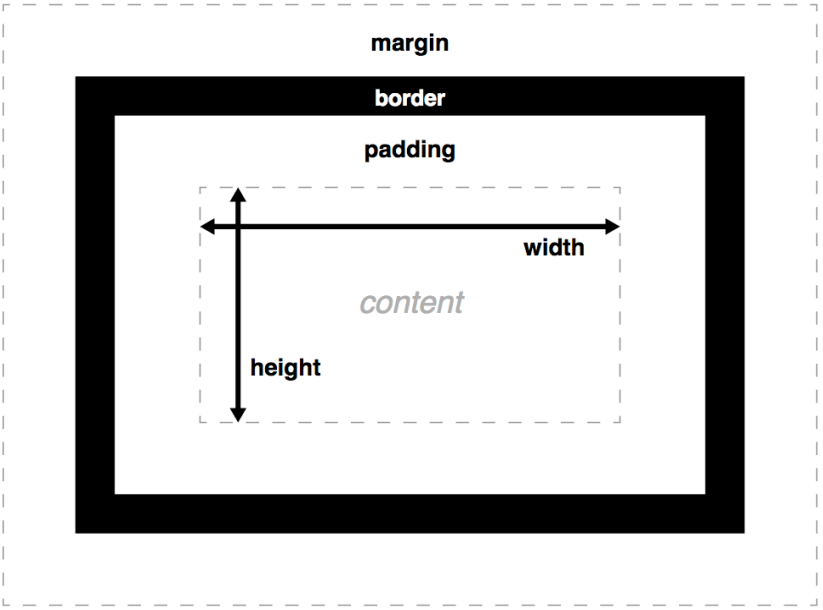
```
<head>
  <style>
    p { color: green; }
  </style>
</head>
```

3. **External** (.css externo): arquivo separado referenciado no HTML. Ideal para manutenção, reutilização e eficiência, principalmente em grandes projetos (Duckett, 2022).

```
<head>
  <link rel="stylesheet" href="styles.css">
</head>
```

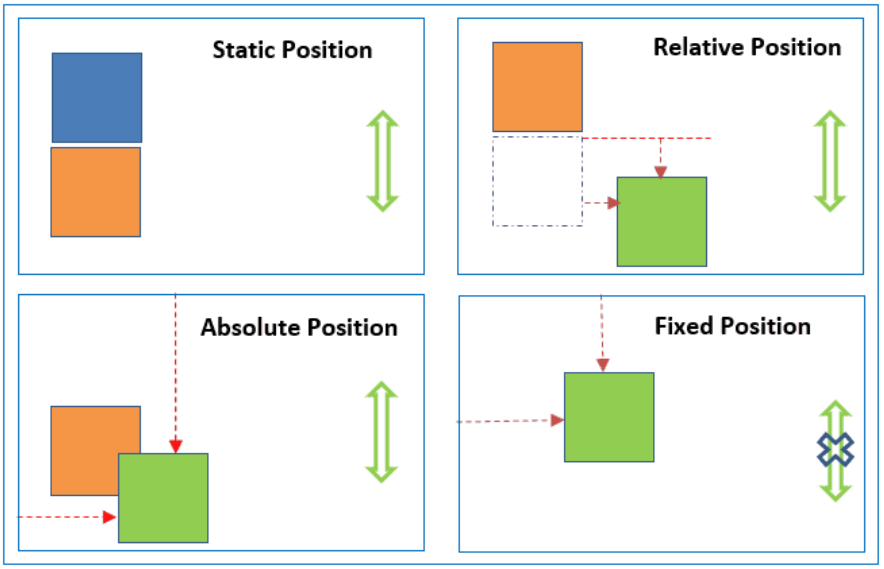
4.4 Box model e posicionamento básico

O modelo de caixa (box model) é a base para o layout de elementos em CSS. Cada elemento é formado por quatro camadas: conteúdo (content), preenchimento (padding), borda (border) e margem (margin) (MDN Web Docs, 2024).



O posicionamento básico de elementos pode ser controlado por:

- static: posicionamento padrão, o elemento segue o fluxo normal.
- relative: elemento ajustado em relação à sua posição inicial.
- absolute: posicionado em relação ao ancestral mais próximo posicionado.
- fixed: elemento fixo em relação à janela do navegador, útil para menus fixos



4.5 Exemplos de regras de estilo

1. Alteração de cor de fundo:

```
body {
  background-color: #f2f2f2; /* Define o fundo cinza claro da página */
}
```

2. Layout simples com Flexbox

```
.container {
  display: flex; /* ativa layout flexível */
  justify-content: center; /* alinha horizontalmente */
  align-items: center; /* alinha verticalmente */
}
```

3. Estilização básica de links:

```
a {
  color: #007bff; /* cor do link padrão */
  text-decoration: none; /* remove sublinhado */
}

a:hover {
  text-decoration: underline; /* sublinha ao passar o mouse */
}
```

Essas regras demonstram claramente o poder e a flexibilidade do CSS para controlar o estilo visual de páginas web (Duckett, 2022).

7. JavaScript – Interatividade e Comportamento

7.1 O que é JavaScript no contexto web

JavaScript é uma linguagem de programação interpretada pelos navegadores que permite adicionar interatividade, validações, animações e dinamismo às páginas web. Diferente do HTML e CSS, que lidam com estrutura e estilo, o JavaScript atua no comportamento da aplicação, rodando diretamente no cliente (navegador), sem necessidade de recarregar a página (Flanagan, 2020). Isso torna as interfaces mais responsivas e modernas.

7.2 Inserção de scripts em HTML

O JavaScript pode ser incluído em uma página HTML de três formas:

1. Inline:

```
<button onclick="alert('Olá!')">Clique</button>
```

2. Internal:

```
<script>
  console.log("Rodando JavaScript interno");
</script>
```

3. External:

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

As boas práticas recomendam o uso de scripts externos e o atributo defer para carregar o código após o parsing do HTML, sem bloquear o carregamento da página. O async também carrega de forma assíncrona, mas pode executar fora de ordem (MDN Web Docs, 2024).

7.3 Manipulação do DOM (Document Object Model)

O DOM representa a estrutura da página como uma árvore de objetos que pode ser acessada e modificada pelo JavaScript. Algumas APIs comuns para manipulação incluem:

```
Unset
const titulo = document.querySelector('h1');       // Seleciona o primeiro
<h1>
const paragrafo = document.getElementById('info'); // Seleciona pelo ID
titulo.innerHTML = 'Novo título';                  // Altera o conteúdo
paragrafo.classList.add('destaque');               // Adiciona uma classe
CSS
```

Esses métodos permitem atualizar o conteúdo da página dinamicamente, sem recarregar (Resig & Bibeault, 2013).

7.4 Eventos básicos: click, load, submit

Eventos representam ações que ocorrem na página, como cliques, carregamento e envio de formulários. Podemos escutá-los usando addEventListener:

```javascript
window.addEventListener('load', () => {
  console.log("Página carregada");
});

document.querySelector('#btn').addEventListener('click', () => {
  alert("Botão clicado!");
});

document.querySelector('form').addEventListener('submit', (e) => {
  e.preventDefault(); // Evita recarregar a página
  console.log("Formulário enviado");
});
```

Esses eventos tornam a aplicação mais interativa e permitem capturar a intenção do usuário (Flanagan, 2020).

7.5 Exemplo de função simples de interação

```unset
<!-- HTML: botão e texto -->
<button id="botao">Clique aqui</button>
<p id="mensagem">Texto original</p>

<script>
  // Seleciona o botão e o parágrafo
  const botao = document.getElementById('botao');
  const mensagem = document.getElementById('mensagem');

  // Adiciona evento de clique
  botao.addEventListener('click', () => {
    mensagem.innerText = "Você clicou no botão!"; // Altera o texto do
  parágrafo
  });
</script>
```

Esse exemplo mostra como capturar um clique e modificar o conteúdo da página sem recarregá-la, usando a API DOM e eventos JavaScript.

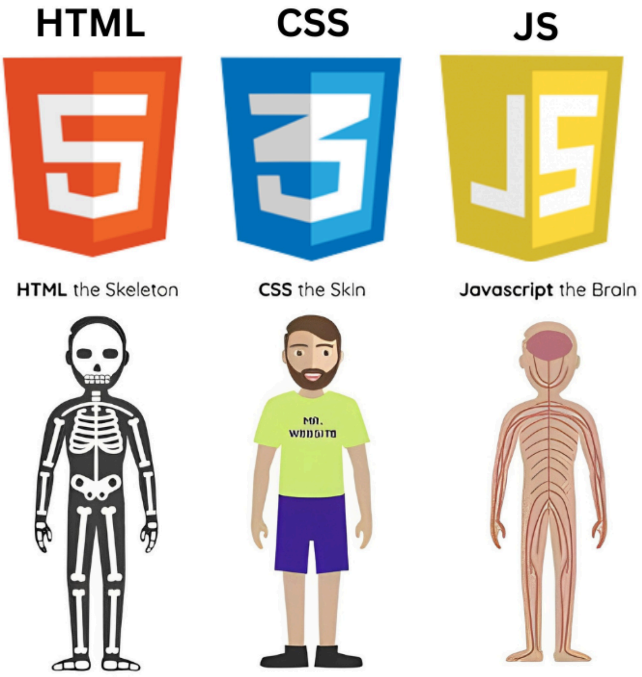
8. Resumo dos Papéis das Três Tecnologias

8.1 Comparativo de responsabilidades

As três principais tecnologias da web – HTML, CSS e JavaScript – possuem papéis distintos, mas complementares. A tabela abaixo resume suas responsabilidades, escopo, limitações e exemplos práticos de uso (Duckett, 2022; MDN Web Docs, 2024).

Tecnologia	Papel principal	Escopo	Limitações	Exemplos de uso prático
HTML	Marcar e estruturar conteúdo	Define elementos e semântica	Não define aparência nem comportamento	Títulos, listas, formulários, imagens
CSS	Estilizar e organizar visualmente	Controla layout e estilo	Não pode criar ou alterar conteúdo dinâmico	Cor de fundo, grid/flex, responsividade
JavaScript	Adicionar interatividade e lógica	Manipula DOM, eventos e dados	Depende de HTML bem estruturado e CSS	Validação de formulários, sliders, modais

Cada camada atua sobre a anterior: HTML fornece a estrutura, CSS estiliza os elementos e JavaScript adiciona comportamento dinâmico (Resig & Bibeault, 2013).



Analogia do uso de cada ferramenta dentro do desenvolvimento web

8.2 Fluxo de trabalho integrado: HTML → CSS → JavaScript

Ao carregar uma página web, o navegador segue uma sequência específica para interpretar e renderizar as três camadas tecnológicas:

1. HTML é carregado primeiro, criando a árvore DOM com os elementos da página (W3C, 2023).
2. CSS é carregado em seguida; os estilos são aplicados ao DOM, criando a CSSOM (CSS Object Model).
3. O navegador computa o layout, combinando DOM e CSSOM para desenhar os elementos na tela.
4. JavaScript é então executado (respeitando a ordem e uso de defer ou async), podendo manipular o DOM/CSSOM, adicionar eventos e gerar conteúdos dinâmicos.

Esse fluxo garante uma construção progressiva da interface, do conteúdo bruto à interação refinada. A ordem correta de carregamento evita erros de renderização e melhora a performance (Flanagan, 2020).

9. Conclusão

9.1 Por que é importante separar estrutura, estilo e comportamento?

A separação entre estrutura (HTML), estilo (CSS) e comportamento (JavaScript) é um dos pilares da arquitetura moderna da web. Essa divisão proporciona diversos benefícios:

- Manutenção: permite que desenvolvedores atuem de forma especializada e organizada. Alterar o estilo visual de um site não exige mudanças no HTML, por exemplo, o que reduz o risco de erros (Silva, 2021).
- Performance: navegadores podem fazer cache de arquivos CSS e JavaScript, melhorando o tempo de carregamento. O HTML permanece enxuto, carregando rapidamente o conteúdo básico (Silva, 2021).
- Acessibilidade: ao manter a semântica HTML limpa e clara, tecnologias assistivas como leitores de tela conseguem interpretar melhor o conteúdo, favorecendo a inclusão (W3C, 2023).
- Escalabilidade: projetos maiores exigem organização em camadas. Separar responsabilidades evita acoplamento excessivo e facilita a evolução contínua do sistema (Silva, 2021).

9.2 Como a segurança (HTTPS) impacta a experiência do usuário?

O uso de HTTPS é essencial tanto do ponto de vista técnico quanto da experiência do usuário:

- Confiança: navegadores indicam quando uma conexão é segura, aumentando a credibilidade do site e a sensação de segurança dos usuários (Rocha, 2022).
- Integridade: garante que os dados transmitidos entre cliente e servidor não sejam modificados durante o tráfego por terceiros, protegendo formulários, autenticações e APIs (Rocha, 2022).

- Desempenho: apesar do custo inicial do handshake TLS, tecnologias como HTTP/2 e TLS 1.3 tornam o carregamento mais eficiente do que conexões HTTP tradicionais, com múltiplas otimizações (Rocha, 2022).
- Compatibilidade e SEO: o Google prioriza sites com HTTPS em seu ranking e muitos recursos modernos da web (como geolocalização) exigem conexões seguras (Google Developers, 2024).

9.3 Quais são os desafios de renderização em diferentes navegadores?

Desenvolver para a web exige atenção a diferenças entre navegadores:

- Engines de renderização: cada navegador utiliza um motor diferente (por exemplo, Chrome usa Blink; Firefox, Gecko; Safari, WebKit), o que pode resultar em variações de comportamento no CSS ou suporte parcial a APIs JavaScript (Andrade, 2023).
- Inconsistências de layout: propriedades como flex, grid ou pseudo-elementos podem ser renderizadas de formas distintas, especialmente em navegadores desatualizados ou mobile (MDN Web Docs, 2024).
- Necessidade de testes cruzados: é fundamental validar a aplicação em múltiplos ambientes e dispositivos usando ferramentas como BrowserStack, DevTools e técnicas de progressive enhancement (Andrade, 2023).
- Fallbacks e polyfills: muitas vezes é necessário oferecer estilos alternativos ou incluir scripts que simulem funcionalidades não suportadas nativamente por certos navegadores.

Referências Bibliográficas:

Cisco. (2022). Cisco Annual Internet Report (2018–2023). Recuperado em 4 de maio de 2025, de https://www.cisco.com/c/en/us/solutions/executive-perspectives/annual-internet-report/index.html

Kurose, J. F., & Ross, K. W. (2021). Redes de Computadores e a Internet: uma abordagem top-down (8ª ed.). Pearson Education.

Pew Research Center. (2023). The Internet and the Pandemic. Recuperado em 4 de maio de 2025, de https://www.pewresearch.org/internet/2023/02/07/the-internet-and-the-pandemic/

UNCTAD. (2023). Digital Economy Report 2023. United Nations Conference on Trade and Development. Recuperado em 4 de maio de 2025, de https://unctad.org/topic/ecommerce-and-digital-economy/digital-economy-report-2023

Dierks, T., & Rescorla, E. (2008). The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246. Internet Engineering Task Force (IETF). Recuperado de https://datatracker.ietf.org/doc/html/rfc5246

Fielding, R., & Reschke, J. (2014). Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. RFC 7230. Internet Engineering Task Force (IETF). Recuperado de https://datatracker.ietf.org/doc/html/rfc7230

Kurose, J. F., & Ross, K. W. (2021). Redes de Computadores e a Internet: Uma Abordagem Top-Down (8ª ed.). Pearson Education.

Mozilla Developer Network (MDN). (2024). HTTP request methods. Recuperado em 4 de maio de 2025, de https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

Rescorla, E. (2018). The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446. Internet Engineering Task Force (IETF). Recuperado de https://datatracker.ietf.org/doc/html/rfc8446

Tanenbaum, A. S., & Wetherall, D. J. (2022). Computer Networks (6ª ed.). Pearson Education.

Buytaert, D. (2020). *Pro Apache HTTP Server*. Apress. https://doi.org/10.1007/978-1-4842-5731-2

Grigorik, I. (2013). *High Performance Browser Networking*. O'Reilly Media. https://hpbn.co

Kurose, J. F., & Ross, K. W. (2021). *Computer Networking: A Top-Down Approach* (8th ed.). Pearson.

Mockapetris, P. (1987). *Domain Names—Concepts and Facilities* (RFC 1034). Internet Engineering Task Force. https://www.rfc-editor.org/rfc/rfc1034

Nginx Inc. (2022). *Nginx Reference Architecture*. https://docs.nginx.com

Stevens, W. R. (2011). *TCP/IP Illustrated, Vol. 1* (2nd ed.). Addison-Wesley.

Flanagan, D. (2020). JavaScript: The Definitive Guide (7ª ed.). O'Reilly Media. https://www.oreilly.com/library/view/javascript-the-definitive/9781491952023/

MDN Web Docs. (2024). JavaScript Guide. Mozilla. https://developer.mozilla.org/en-US/docs/Web/JavaScript

Resig, J., & Bibeault, B. (2013). Secrets of the JavaScript Ninja (2ª ed.). Manning Publications. https://www.manning.com/books/secrets-of-the-javascript-ninja-second-edition

Duckett, J. (2022). HTML & CSS: Design and Build Websites (2ª ed.). John Wiley & Sons. https://www.wiley.com

Flanagan, D. (2020). JavaScript: The Definitive Guide (7ª ed.). O'Reilly Media. https://www.oreilly.com/library/view/javascript-the-definitive/9781491952023/

MDN Web Docs. (2024). HTML, CSS, and JavaScript Guide. Mozilla. https://developer.mozilla.org/en-US/docs/Learn

Resig, J., & Bibeault, B. (2013). Secrets of the JavaScript Ninja (2ª ed.). Manning Publications. https://www.manning.com/books/secrets-of-the-javascript-ninja-second-edition

W3C. (2023). HTML Living Standard. https://html.spec.whatwg.org/

Silva, M. (2021). *Arquitetura Front-End: Princípios e Boas Práticas*. São Paulo: DevPress. https://devpress.com/arquitetura-frontend

Rocha, T. (2022). *Segurança Web Moderna: HTTPS, TLS e Certificados Digitais*. Rio de Janeiro: WebSecure. https://websecure.org/https-guia

Andrade, L. (2023). *Compatibilidade entre Navegadores: Desafios e Estratégias*. Florianópolis: BrowserTest Lab. https://browsertestlab.org/guias/compatibilidade-css-js

W3C. (2023). *Accessibility Guidelines (WCAG 2.1)*. World Wide Web Consortium. https://www.w3.org/WAI/WCAG21

MDN Web Docs. (2024). *Cross-browser testing and compatibility*. Mozilla. https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Cross_browser_testing

Google Developers. (2024). *Why HTTPS matters*. https://developers.google.com/web/fundamentals/security/encrypt-in-transit/why-https