ChatGPT

Checklist completo para construção de sites de alta qualidade

Este checklist reúne boas práticas universais para criar sites rápidos, seguros, acessíveis e prontos para auditorias como o **PageSpeed Insights**. As orientações são baseadas nos problemas identificados no diagnóstico do site *advanced-amino-formula.semprenamoda.com.br* e em pesquisa de fontes especializadas.

1. Desempenho e Core Web Vitals

1.1 Defina um orçamento de desempenho

A velocidade de um site tende a piorar ao longo do tempo. O Google recomenda definir um *performance budget*: limites para métricas como **First Contentful Paint (FCP)** e **Time to Interactive (TTI)**, tamanho do JavaScript e pontuação do Lighthouse, e monitorá-los continuamente. Por exemplo, um orçamento pode estabelecer que o FCP em 3G lento deve ser < 2 s, o TTI < 5 s, a pontuação de desempenho do Lighthouse > 80 e o total de JavaScript < 170 KB. Automatize esses limites em seu pipeline de CI/CD para que builds falhem caso ultrapassem o orçamento.

1.2 Reduza recursos bloqueadores de renderização

- **CSS crítico**: carregue apenas o CSS essencial no `<head>` e adie o restante. Minifique seus arquivos e elimine regras não usadas. Prefira `rel="preload" as="style"` com `onload` para tornar o CSS não bloqueante.
- **JavaScript**: scripts podem bloquear o parser HTML. Marque scripts externos com `async` ou `defer` para evitar travar a renderização. Divida código em módulos menores, carregando apenas o que for necessário em cada página. Remova scripts não utilizados ou adie a execução de funcionalidades secundárias.
- **Solicitações em cadeia**: evite dependências profundas onde cada recurso precisa baixar outro. O PageSpeed apontou uma árvore de dependência longa; revise a arquitetura para paralelizar downloads.

1.3 Otimize CSS

- **Minifique e agrupe**: remova espaços, comentários e quebras de linha para reduzir o tamanho dos arquivos.
- **CSS crítico inline**: para estilos essenciais, inclua as regras dentro de `<style>` no `<head>` para exibir rapidamente o conteúdo acima da dobra.
- **Evite CSS inline no** `<body>`: mantê-lo em arquivos externos facilita cache, manutenção e melhora a performance.
- **Simplifique e elimine regras não usadas** com ferramentas como PurgeCSS/Stylelint.

1.4 Otimize JavaScript

- **Minifique**: remova comentários e espaço em branco para reduzir tamanho.

- **Divida e carregue sob demanda**: use *code splitting* para que páginas carreguem apenas o código necessário. Agrupe funcionalidades secundárias em arquivos carregados posteriormente.
- **Marque scripts com** `async` **ou** `defer` para torná-los não bloqueantes.
- **Reduza o trabalho na thread principal**: o diagnóstico apontou ~2,3 s de processamento na thread principal. Simplifique cálculos, elimine bibliotecas pesadas e mova lógicas demoradas para Web Workers.

1.5 Otimize imagens

- **Escolha o formato correto**: o WebP ou AVIF oferecem compressão superior e devem ser usados sempre que possível. Use PNG/WebP lossless para imagens com muitos detalhes e JPEG/AVIF/WebP para fotos.
- **Dimensione corretamente**: forneça imagens na resolução exata em que serão exibidas e utilize `srcset` / `sizes` para carregar a versão mais adequada. O PageSpeed apontou imagens baixadas maiores que o necessário.
- **Comprima arquivos**: use ferramentas (e.g., TinyPNG, ImageOptim) para remover metadados e reduzir o tamanho sem comprometer a qualidade.
- **Substitua GIFs por vídeos** para animações, pois vídeos são mais eficientes.
- **Acelere a LCP**: identifique a imagem de **Largest Contentful Paint** no HTML e atribua `fetchpriority="high"` para priorizar seu carregamento; evite lazily loading desse recurso.

1.6 Otimize fontes

- **Evite embedar fontes grandes** diretamente em CSS/JS. O web.dev recomenda não embutir arquivos de fonte para evitar atrasos no discovery.
- **Pré-conecte domínios de fontes** com `<link rel="preconnect" href="https://fonts.example.com" crossorigin>` para reduzir latência.
- **Use WOFF2** sempre que possível, pois comprime ~30 % melhor que WOFF.
- **Subconjunto e limite pesos**: carregue apenas os caracteres e pesos necessários. Fontes excessivas aumentam o tempo de download.

1.7 Latência do servidor, cache e compressão

- **Minimize o Time To First Byte (TTFB)**: reduza redirecionamentos, optimize consultas ao banco de dados e use cache de página. O guia de desempenho recomenda manter TTFB < 1,3 s.
- **Habilite compressão Brotli ou Gzip** para HTML, CSS e JS para diminuir o tamanho transferido.
- **Implemente cache eficiente**: configure cabeçalhos de cache com prazos longos para recursos estáticos. O PageSpeed notou TTL de apenas 10 minutos em alguns ativos; use pelo menos 1 ano para ativos versionados.
- **Use CDN** para distribuir assets globalmente; isso reduz latência e protege contra picos de tráfego.

1.8 Evite reflow e layout thrashing

JavaScript que lê e escreve propriedades de layout repetidamente desencadeia *reflows* forçados, atrasando a renderização. Evite acessar propriedades como `offsetWidth` após alterações no DOM; agrupe leituras e gravações. Use `transform` e `opacity` para animações, pois não provocam recalculo de layout.

1.9 `fetchpriority` e descoberta de requisições

O atributo `fetchpriority` ajuda o navegador a priorizar recursos sem bloquear a renderização. Aplique `fetchpriority="high"` na imagem de LCP e scripts essenciais, e `fetchpriority="low"` em scripts ou imagens não críticos. Assegure-se de que o recurso LCP seja encontrado diretamente no HTML, não através de JavaScript; evite carregamento preguiçoso desse recurso.

1.10 Árvore de dependências de rede

Evite longas cadeias de solicitações (por exemplo, script A importa B que importa C). Quanto mais profunda a árvore, mais tempo o navegador leva para iniciar downloads; reorganize as dependências para paralelizar o carregamento.

1.11 Outras recomendações de desempenho

- **Minifique e agrupe HTML** para remover comentários e espaços redundantes.
- **Use *lazy loading*** em imagens e iframes não essenciais para adiar downloads.
- **Evite tarefas longas** (> 50 ms) na thread principal; divida funções pesadas ou use Web Workers.
- **Monitore de forma contínua**: use ferramentas como PageSpeed Insights, Lighthouse CI e Chrome UX Report para medir o impacto das mudanças.

2. Acessibilidade

A acessibilidade garante que qualquer pessoa, inclusive pessoas com deficiências, possa utilizar seu site. O guia da WebAbility resume dez práticas essenciais e fornece detalhes de implementação. Principais recomendações:

2.1 Estrutura semântica

- Use elementos HTML5 corretos (`<nav>`, `<main>`, `<article>`, `<header>`, `<footer>`) para fornecer significado e melhorar a navegação de leitores de tela. Só um `<h1>` por página; siga a ordem hierárquica dos cabeçalhos (H1, H2, H3…) e evite pular níveis.
- Defina landmarks (`<main>`, `<aside>`) para permitir que usuários saltem diretamente para a seção desejada.
- Valide seu HTML com a ferramenta de validação do W3C para detectar erros estruturais.

2.2 Texto alternativo para imagens

- Adicione **atributos** `alt` **descritivos** a todas as imagens significativas para que leitores de tela possam transmitir o conteúdo.
- Mantenha o texto alternativo conciso (< 125 caracteres) e evite frases como “imagem de…”.
- Imagens puramente decorativas devem usar `alt=""` para serem ignoradas pelos leitores.
- Descreva a **função** da imagem quando ela for um link ou botão.

2.3 Navegação por teclado

Certifique-se de que todos os elementos interativos (links, botões, formulários) possam ser acessados e utilizados apenas com o teclado. Gerencie o foco de forma previsível e visível. Use o atributo `tabindex` de maneira criteriosa para definir ordem lógica.

2.4 Contraste de cores

Garanta contraste mínimo de 4,5:1 para texto normal e 3:1 para textos grandes. Use ferramentas como WebAIM Contrast Checker para testar combinações e evite cores como única forma de transmitir informação.

2.5 ARIA e componentes dinâmicos

Quando elementos dinâmicos (menus, carrosséis) não podem ser descritos apenas com HTML, utilize atributos ARIA apropriados (`role`, `aria-labelledby`, `aria-expanded`). Não abuse de ARIA; prefira elementos nativos sempre que possível.

2.6 Formulários acessíveis

Associe rótulos (`<label>`) a cada campo usando `for`/`id` e informe claramente o propósito do campo. Forneça mensagens de erro específicas e sugestões para correção.

2.7 Design responsivo e mobile-first

Garanta que layouts se adaptem a diferentes tamanhos de tela e orientações. Utilize unidades relativas (%, `rem`, `vw`/`vh`), media queries e flexbox/grid.

2.8 Estrutura de conteúdo e legibilidade

Organize o conteúdo em seções claras com cabeçalhos lógicos e utilize linguagem simples. Evite jargões e textos longos sem divisão.

2.9 Tratamento de erros

Mostre mensagens de erro claras e instrutivas, indicando o que deu errado e como corrigir. Nunca dependa apenas de cor para sinalizar erros.

2.10 Gestão de foco

Mantenha o foco visível (por exemplo, com `outline`) e faça-o percorrer os elementos na ordem esperada. Após ações como fechamento de modais ou envio de formulários, retorne o foco ao elemento mais apropriado.

2.11 Testes de acessibilidade

- Use ferramentas automáticas (Lighthouse, axe, WAVE) para detectar problemas comuns.
- Realize testes manuais com navegação por teclado e leitores de tela (NVDA, VoiceOver).
- Inclua pessoas com deficiência no processo de validação para identificar barreiras reais.

3. Segurança

A segurança deve ser parte integrante do desenvolvimento. A pesquisa de 2024/2025 destaca os cabeçalhos de segurança e medidas de proteção como críticos.

3.1 HTTPS e TLS

- **Use TLS 1.3** para conexões seguras. Forneça certificados válidos e automatize sua renovação.
- **Habilite HSTS (HTTP Strict Transport Security)** com `max-age` alto (≥ 31536000), `includeSubDomains` e `preload` para forçar o uso de HTTPS.
- **Elimine conteúdo misto** redirecionando todas as URLs HTTP para HTTPS.

3.2 Cabeçalhos de segurança

- **Content-Security-Policy (CSP)**: limita as origens de scripts, estilos e recursos, prevenindo ataques XSS. Utilize `nonce` ou `hash` e a diretiva `strict-dynamic` para permitir scripts confiáveis sem listas longas.
- **X-Content-Type-Options: nosniff**: impede que o navegador faça *sniffing* de tipos de arquivos, reduzindo o risco de download malicioso.
- **X-Frame-Options: SAMEORIGIN**: protege contra *clickjacking* controlando quem pode incorporar sua página em um iframe.
- **Referrer-Policy** e **Permissions-Policy**: defina políticas para privacidade e recursos sensíveis (microfone, câmera).

3.3 Firewall de Aplicação Web (WAF)

Implemente um WAF para filtrar e monitorar solicitações maliciosas. Configure regras personalizadas, habilite proteção contra bots e ataques DDoS e monitore os logs regularmente.

3.4 Atualizações e gerenciamento de patches

Mantenha todos os componentes (CMS, bibliotecas, plugins) atualizados. Automatize atualizações sempre que possível, teste em ambiente de staging e siga um cronograma regular.

3.5 Backups e recuperação de desastres

Adote a regra 3-2-1: três cópias dos dados, em duas mídias diferentes e uma delas off-site. Automatize backups, verifique a integridade e teste a restauração periodicamente. Criptografe os backups em trânsito e em repouso.

3.6 Controle de acesso e autenticação

- Aplique o princípio do **menor privilégio**: conceda a cada usuário apenas as permissões necessárias.
- Use **autenticação multifatorial (MFA)** para acessos administrativos.
- Faça revisão periódica de contas e revogue permissões não utilizadas.
- Aplique políticas de senha fortes e armazenamento seguro (hash com salt).

3.7 Monitoramento e auditoria

Implemente sistemas de **SIEM** e configure alertas em tempo real para atividades suspeitas. Realize avaliações de vulnerabilidade, testes de penetração e monitore componentes de terceiros.

3.8 Plano de resposta a incidentes

Prepare um plano que inclua **preparação, detecção, contenção, erradicação, recuperação e lições aprendidas**. Documente responsabilidades, fluxos de comunicação e procedimentos de escalonamento.

4. SEO e Metadados

Para que um site seja encontrado e ofereça um snippet atrativo nos resultados de pesquisa, siga estas orientações:

4.1 Otimização de títulos (title tags)

Trate cada título como um mini pitch de vendas: **frente carregue a palavra-chave principal**, mencione o benefício/resultado e qualifique o público. Em 2025, títulos eficazes têm 45–65 caracteres. Evite *keyword stuffing* e títulos vagos; alinhe o título com a intenção da página.

4.2 Metadescrições

Embora não influenciem diretamente o ranking, descrições atraentes aumentam o **CTR**. Escreva 120–155 caracteres que completem a ideia do título, descrevendo o conteúdo, o público e a próxima ação. Evite duplicar descrições ou deixá-las vazias.

4.3 Dados estruturados (Schema)

Implemente marcação *Schema.org* adequada (FAQ, HowTo, Produto, Avaliação) para permitir rich snippets e para que sistemas de IA entendam melhor entidades e relações. Use JSON-LD no `<script type="application/ld+json">`.

4.4 Meta tags sociais (Open Graph, Twitter Cards)

Defina `og:title`, `og:description` e `og:image` alinhados ao conteúdo da página. Isso controla como seu site aparece em compartilhamentos e no preview de aplicativos de mensagens.

4.5 Sitemap, robots.txt e canônicos

- Gere e envie sitemaps XML para o Google Search Console.
- Defina regras `robots.txt` para bloquear páginas que não devem ser indexadas.
- Use `<link rel="canonical">` para indicar o URL preferido de páginas duplicadas.
- Mantenha URLs amigáveis e consistentes.

4.6 Conteúdo de qualidade e intenção do usuário

Crie conteúdo original, útil e alinhado à intenção de pesquisa. Utilize uma hierarquia de cabeçalhos clara e inclua palavras-chave naturalmente. Atualize regularmente para refletir mudanças no produto ou nos serviços.

5. Compatibilidade de navegadores e responsividade

5.1 Testes cross-browser

Execute testes que verifiquem funcionalidade, aparência e desempenho em diferentes navegadores (Chrome, Safari, Firefox, Edge), versões e sistemas operacionais. O artigo da Virtuoso QA destaca que aplicações devem se comportar consistentemente em browsers e dispositivos variados para evitar frustração de usuários. Automatize esses testes para cobrir a combinação de navegadores, versões e tamanhos de tela.

5.2 Design responsivo e mobile-first

Adote abordagem mobile-first: comece o layout pelo menor viewport e escale para telas maiores. Use grades fluidas, `flexbox` / `grid`, `min()` / `max()` e unidades relativas. Teste em dispositivos reais e simuladores para garantir que elementos não fiquem sobrepostos e que as interações (toque, gestos) sejam funcionais.

5.3 Progressive enhancement e *graceful degradation*

Construa a experiência principal com tecnologias amplamente suportadas e adicione melhorias progressivas para navegadores modernos. Garanta que funcionalidades básicas permaneçam acessíveis mesmo que certos recursos (JavaScript, cookies) estejam desativados.

6. Boas práticas de codificação e manutenção

6.1 Arquitetura modular e limpa

- Separe responsabilidades: componentes reutilizáveis, serviços e utilitários isolados facilitam manutenção e testes.
- Use padrões MVC/MVVM ou frameworks modernos que incentivam modularização.

6.2 Documentação e comentários

- Documente funções, APIs e componentes. Comentários devem explicar *por quê* e não apenas *o que* o código faz.
- Mantenha uma wiki ou README atualizado.

6.3 Controle de versão e integração contínua (CI/CD)

- Use Git ou similar para versionar código e facilitar revisão.
- Configure pipelines de CI para executar testes automatizados (unitários, integração, acessibilidade, performance) em cada commit.
- Faça *deploy* automatizado para ambientes de staging e produção com rollback seguro.

6.4 Revisão humana e uso de IA

Ferramentas de IA podem gerar código rapidamente, mas é essencial revisão manual para garantir que o código seja legível, seguro e conforme os requisitos de performance e acessibilidade. Treine a IA com padrões de estilo e valide as saídas com testes automatizados.

Conclusão

Seguir este checklist ao construir sites – seja manualmente, seja com auxílio de inteligência artificial – ajuda a evitar os problemas identificados no relatório anterior e a atingir pontuações próximas de 100/100 em auditorias. A combinação de **performance**, **acessibilidade**, **segurança**, **SEO** e **manutenibilidade** não só melhora a experiência do usuário, mas também contribui para melhor posicionamento nos motores de busca e reduz riscos operacionais. Adote uma cultura de melhoria contínua: monitore métricas, atualize suas práticas e evolua seu site conforme novos padrões e tecnologias surgem.
