The 2025 State of the Web Report: Architectural Standards, Design Paradigms, and Security Protocols for World-Class Digital Platforms

Executive Summary

The digital ecosystem of 2025 represents a convergence of high-fidelity interaction, rigorous security posturing, and the decentralization of intelligence. The era of the static web is definitively over, replaced by an environment where websites function as sophisticated software platforms capable of offline operation, local artificial intelligence inference, and cinematic user experiences. This report serves as a comprehensive operational blueprint for architects, developers, and product stakeholders tasked with building world-class websites.

Our analysis of the current landscape reveals three critical pivots. First, the **Architectural Pivot**: the shift from monolithic client-side hydration to server-first rendering and "island" architectures, driven by the need to optimize Interaction to Next Paint (INP). Second, the **Aesthetic Pivot**: the rejection of flat design in favor of spatial interfaces—Glassmorphism and Neumorphism—that leverage depth and physics to guide user attention. Third, the **Intelligence Pivot**: the migration of AI workloads from the cloud to the client via WebLLM and WebGPU, enabling privacy-preserving, zero-latency intelligence.

However, these advancements introduce complex liabilities. The attack surface has expanded with the rise of Server Actions and increasingly complex supply chains. Simultaneously, the introduction of WCAG 2.2 standards has transformed accessibility from a compliance checklist into a fundamental design constraint.

This document synthesizes data from over 200 industry sources, technical documentations, and trend analyses to provide an exhaustive guide to the "Do's and Don'ts" of modern web development. It is structured to facilitate high-level strategic decision-making while providing granular technical directives for implementation.

---

1. The Visual Language of 2025: Immersion, Depth, and Physics

The visual identity of the modern web has graduated from the utilitarian flatness of the previous decade into a rich, tactile environment. World-class websites in 2025 are defined not just by how they look, but by how they *feel*. The integration of physics-based motion, variable typography, and dimensional layering creates interfaces that mimic the organic properties of the physical world, thereby reducing cognitive load and increasing intuitive navigability.

1.1 The New Materialism: Glassmorphism and Neumorphism

After years of dominance by "Flat Design," 2025 has seen the solidification of "The New Materialism," characterized primarily by Glassmorphism and Neumorphism. These styles are not merely decorative trends; when executed correctly, they serve vital functional roles in establishing visual hierarchy and context.[1]

Glassmorphism creates a sense of depth by emulating frosted glass. It uses background blur, transparency, and subtle white borders to separate UI layers. This technique allows users to maintain context of what lies "behind" an active element, such as a modal or a floating navigation bar, without visual clutter. The "frosted" effect effectively creates a mental model of elevation.[2]

- **Mechanism:** The effect relies on the CSS backdrop-filter: blur() property combined with semi-transparent backgrounds (e.g., rgba(255, 255, 255, 0.2)).
- **The Trap (Don'ts):** A critical failure mode in Glassmorphism is accessibility. If the background blur is insufficient or the transparency too high, text contrast ratios often fall below the WCAG 2.2 AA standard of 4.5:1. Furthermore, excessive use of backdrop-filter is computationally expensive, particularly on mobile GPUs, leading to frame drops during scrolling.[1]

Neumorphism (Soft UI) has evolved from its initial experimental phase into a refined tool for interactive affordance. By using light and shadow to create elements that appear extruded from the background (rather than floating above it), Neumorphism provides a tactile quality to buttons and toggles. In 2025, this is rarely applied to an entire interface but is reserved for "touchable" controls to signal interactivity.[2]

- **Strategic Application:** World-class sites use Neumorphism for micro-interactions—toggles, sliders, and primary action buttons—while retaining high-contrast flat design for typography and data display. This hybrid approach resolves the early readability issues associated with the style.[3]

1.2 The Bento Grid Revolution

The rigid, single-column layouts of the past have given way to **Bento Grids**, a layout paradigm inspired by the compartmentalized Japanese lunch box. This trend dominates 2025 web design because it solves the "content variety" problem. Modern websites must display a heterogeneous mix of content—videos, 3D models, text, interactive widgets—and the Bento Grid provides a unified yet flexible structure to house them.[4]

The Bento Grid is inherently modular. It organizes content into distinct, rectangular blocks of varying sizes that lock together. This modularity is not just aesthetic; it is deeply tied to the component-based architecture of modern frameworks like React and Vue. Each "cell" in the grid functions as an independent component.[5]

Do's for Bento Grids:

- **Do** leverage **CSS Grid Layout** for implementation. The two-dimensional nature of CSS Grid is the only robust way to handle the complex row and column spanning required by Bento layouts.
- **Do** utilize **CSS Container Queries** (@container). This allows individual grid cards to adapt their internal layout based on the size of their container, rather than the viewport. A "weather widget" card, for instance, can display full details when spanning two columns but switch to a minimal icon view when compressed to one.[4]
- **Do** maintain consistent gap spacing and border-radius (typically 16px-24px) to ensure the grid feels cohesive rather than chaotic.

Don'ts for Bento Grids:

- **Don't** simply stack the blocks vertically on mobile without consideration. A 12-block grid becomes an endless scroll on a phone. World-class mobile implementations use horizontal swiping carousels for lower-priority grid rows to save vertical space.[4]
- **Don't** use Bento Grids for text-heavy content like articles. The compartmentalization fragments the reading experience. This layout is strictly for dashboards, landing pages, and portfolios.

1.3 Scrollytelling: The Narrative Web

Scrollytelling—the practice of using scroll interaction to drive narrative progression—has matured into a standard for high-end digital storytelling. In 2025, this is no longer about simple parallax; it involves complex sequencing where scrolling triggers video playback, data visualization updates, and 3D model rotations.[6]

The technical enabler for this is the **CSS Scroll-driven Animations** specification (and the View Timeline API). Historically, scroll animations required JavaScript listeners running on the main thread, which often caused "jank" (stuttering) because the scroll event and the render loop were fighting for resources. The new CSS-based approach offloads this work to the compositor thread, ensuring buttery smooth 60fps performance even on mid-range devices.[7]

Immersive Implementation Strategies:

- **The "Sticky" Canvas:** A common pattern involves a "sticky" background element (like a 3D product model or a map) that transforms as the user scrolls through text sections overlaying it.
- **Video Scrubbing:** Linking the currentTime of a video to the scroll position allows users to "scrub" through a video by scrolling, providing frame-accurate control over the

narrative.[9]

Critical Don'ts:

- **Don't Scrolljack:** Never hijack the browser's native scroll physics. Overriding the user's scroll speed or damping creates a disconnect between the physical input (mouse wheel/finger swipe) and the screen output, leading to "motion sickness" and frustration.[10]
- **Don't** ignore **Reduced Motion** preferences. Scrollytelling can be disorienting for users with vestibular disorders. A robust implementation must detect @media (prefers-reduced-motion: reduce) and flatten the experience into a static, readable layout.[11]

1.4 Fluid Typography and Variable Fonts

Typography in 2025 is dynamic. The days of setting static breakpoints for font sizes (e.g., 16px on mobile, 18px on tablet) are over. **Fluid Typography** utilizes the CSS clamp() function to scale text smoothly across *every* pixel of viewport width.

- **Equation:** font-size: clamp(1.5rem, 4vw + 1rem, 3rem);
  - This ensures the font never shrinks below 1.5rem, never exceeds 3rem, but scales fluidly at a rate of 4vw in between. This eliminates "orphan" words and awkward line breaks at intermediate screen sizes.[13]

Variable Fonts act as the perfect partner to fluid sizing. A single font file now contains axes for weight, width, slant, and optical sizing. This allows for:

1. **Performance:** Loading one binary (e.g., Inter-Variable.woff2) instead of 12 separate files for every weight/style combination.
2. **Animation:** Weights can be animated smoothly on hover (e.g., transitioning from weight 400 to 600) without the text "jumping," as the variable font engine handles the interpolation.[15]

1.5 Motion Physics and Micro-interactions

Static interfaces feel broken in 2025. Users expect **Micro-interactions**—subtle animations that acknowledge inputs. The gold standard is **Physics-based Motion**, where animations follow the laws of mass and friction rather than linear time.

The Golden Rules of Motion Timing [16]:

Interaction Type	Duration	Easing	Purpose
Micro (Hover/Click)	100ms - 150ms	Ease-out	Instant feedback; prevents "sluggish"

			feel.
Transition (Card Open)	200ms - 300ms	Spring / Ease-in-out	Context change; allows eye to track movement.
Large (Page Load)	300ms - 500ms	Decelerate	Established mood; smooth entry.

Don't:

- **Don't** animate purely for decoration. Every motion must serve a purpose: guiding the eye, confirming an action, or revealing hierarchy.[17]
- **Don't** use linear easing. It looks robotic and unnatural.

---

2. Architectural Paradigms: The Battle for the Server

Building a "world-class" website requires selecting an architecture that balances Developer Experience (DX) with User Experience (UX). In 2025, the industry has largely bifurcated into two camps: the **Full-Stack Application** (Next.js/Remix) and the **Content-First Island Architecture** (Astro).

2.1 The Framework Ecosystem: Next.js vs. Remix vs. Astro

The choice of framework is the single most significant technical decision in the project lifecycle.

Next.js: The Enterprise Monolith
Next.js remains the default for large-scale applications. Its adoption of React Server Components (RSC) has fundamentally changed how React is written. By rendering components on the server and streaming HTML to the client, Next.js minimizes the JavaScript bundle sent to the browser.18

- *Use Case:* Complex E-commerce, Enterprise Dashboards, Social Networks.
- *Pro:* Tight integration with Vercel infrastructure, massive ecosystem, "Zero-Bundle" server components.
- *Con:* High complexity. The mental model of "Client vs. Server" components is a common source of bugs. Vulnerabilities in Server Actions (detailed in Section 4) require vigilance.[19]

Remix: The Web Standards Purist
Remix has carved out a niche for developers who prioritize web fundamentals. Unlike Next.js, which creates its own abstraction layers, Remix relies heavily on native HTTP caching, HTML

forms, and standard Request/Response objects.

- *Use Case:* SaaS Applications, dynamic data-heavy tools.
- *Pro:* Exceptional data mutation story (Actions), no "waterfall" data fetching (loaders run in parallel on the server), superior handling of dynamic content.[21]
- *Con:* Smaller ecosystem than Next.js.

Astro: The Performance King
For content-driven websites, Astro is unrivaled in 2025. Its Island Architecture allows the page to be static HTML by default. Interactive components (React, Vue, Svelte) are "hydrated" in isolation only when needed.

- *Use Case:* Marketing sites, Portfolios, Blogs, Documentation, News Sites.
- *Pro:* Zero JavaScript by default. Scores 100 on Core Web Vitals almost effortlessly. Agnostic to UI frameworks (can mix React and Svelte).
- *Con:* Not designed for complex, state-heavy application logic (though possible).[22]

2.2 Hydration Strategies and Anti-Patterns

Hydration—the process of attaching JavaScript event listeners to server-rendered HTML—is the bottleneck of the modern web. The 2025 standard is **Partial Hydration** (Astro) or **Selective Hydration** (React Concurrent Mode).

The "Uncanny Valley" of Hydration:
A common "Don't" is shipping massive hydration bundles for static content. If a footer is just a list of links, it should never be hydrated. "Over-hydration" leads to high TBT (Total Blocking Time) and poor INP scores.18
React Anti-Patterns in 2025 [24]:

1. **useEffect for Derived State:** This is a major performance killer.
   - *Bad:* useEffect(() => { setFullName(first + last) }, [first, last])
   - *Good:* const fullName = first + last (Calculated during render).
2. **Prop Drilling:** Passing data through 10 layers of components.
   - *Fix:* Use Composition (passing children) or Server Components to fetch data directly where it is needed.
3. **Large Client Components:** Marking a top-level page as 'use client' de-optimizes the entire tree.
   - *Fix:* Push the client boundary down to the "leaves" (buttons, inputs), keeping the layout as a Server Component.

2.3 PWA Evolution: The Web as an OS

Progressive Web Apps have gained superpowers in 2025. They are no longer just "offline websites" but fully capable applications.

- **Background Sync API:** Allows the app to defer actions (like sending a message) until connectivity is restored, even if the user closes the tab.[27]

- **File System Access API:** Enables PWAs to read/write files directly to the user's hard drive (e.g., a photo editor PWA saving directly to the "Pictures" folder).[29]
- **Window Controls Overlay:** Allows PWAs to customize the title bar area, making them look indistinguishable from native desktop apps.[31]

Do: Implement a Service Worker for asset caching (Strategy: Stale-While-Revalidate).
Do: Use the Web App Manifest to define "standalone" display mode for a native feel.

---

3. The Performance Standard: Core Web Vitals and Beyond

In 2025, performance is a function of user perception, not just network speed. Google's Core Web Vitals (CWV) are the non-negotiable metrics for SEO and UX success.

3.1 Interaction to Next Paint (INP)

INP has replaced FID (First Input Delay) as the primary responsiveness metric. It measures the latency of *all* interactions (clicks, taps, key presses) throughout the page's lifecycle. A good INP is **<200ms**.

Why INP Fails:
High INP is usually caused by the main thread being blocked by heavy JavaScript execution immediately after a user interacts. If the browser is busy parsing a massive JSON object or re-rendering a complex React tree, it cannot paint the next frame to acknowledge the user's click.32
Do's for INP Optimization:

- **Do** use scheduler.yield() or setTimeout to break long tasks into smaller chunks, giving the main thread breathing room to process inputs.
- **Do** use **Web Workers** to offload heavy logic (e.g., image compression, data sorting) off the main thread entirely.
- **Do** show immediate visual feedback (e.g., a button active state) *before* starting the heavy logic.[32]

3.2 Largest Contentful Paint (LCP)

LCP measures loading speed (Target: **<2.5s**). The 2025 silver bullet for LCP is **Fetch Priority**.

- **Technique:** Add <img src="hero.jpg" fetchpriority="high"> to the LCP element. This tells the browser to load this image before stylesheets or non-critical scripts.
- **Don't** lazy-load the LCP element (loading="lazy"). This delays the load until the layout is calculated, destroying LCP scores.[32]

3.3 CSS Performance Features

- **content-visibility: auto:** This property allows the browser to skip rendering layout and painting for elements that are off-screen. For a long page with complex Bento Grids, applying this to lower sections can improve initial load performance by 30-50%.[35]
- **CSS Layers (@layer):** Allows developers to control the "cascade" explicitly, preventing specificity wars and reducing the size of CSS bundles by enabling better tree-shaking logic.[37]

---

4. Security & Governance: The Zero-Trust Web

The modern web is a hostile environment. World-class websites operate on a **Zero Trust** model, assuming that the network is compromised and that malicious inputs will occur.

4.1 Content Security Policy (CSP)

A robust CSP is the single most effective defense against Cross-Site Scripting (XSS). In 2025, the standard is **Nonce-based Strict CSP**. Whitelisting domains (e.g., script-src google.com) is considered insecure because open redirects on trusted domains can be exploited.

Do: Implement the following header:

```http
Content-Security-Policy:
 default-src 'self';
 script-src 'nonce-{RANDOM_STRING}' 'strict-dynamic';
 object-src 'none';
 base-uri 'none';
```

This ensures that *only* scripts with the correct cryptographic nonce (generated per request) can execute. The 'strict-dynamic' directive allows trusted scripts to load their own dependencies.[38]

Don't:

- **Don't** use 'unsafe-inline' or 'unsafe-eval'.
- **Don't** rely on XSS auditors (which are deprecated). CSP is the standard.[41]

4.2 Application-Level Security

- **Server Actions & RPC:** In frameworks like Next.js, Server Actions are public endpoints.
  - **Do:** Implement authentication and authorization checks *inside* every Server Action.
  - **Do:** Use schema validation (Zod/Valibot) to sanitize inputs. Attackers can bypass the UI and send malformed JSON directly to the action URL.[42]
  - **Alert:** CVE-2025-66478 revealed vulnerabilities in React Server Components related to unauthenticated remote code execution. Immediate patching and dependency monitoring (Snyk/Dependabot) are mandatory.[20]
- **HTTP Headers:**
  - **HSTS:** Strict-Transport-Security: max-age=63072000; includeSubDomains; preload. Mandatory to prevent SSL stripping.
  - **X-Content-Type-Options:** nosniff. Prevents MIME-type confusion attacks.[44]

---

5. The Intelligent Browser: Local-First AI

We are witnessing the "Intelligence Pivot." Instead of sending user data to the cloud for processing (expensive, slow, non-private), 2025 websites run AI models directly in the browser.

5.1 WebLLM and Transformers.js

Libraries like **WebLLM** and **Transformers.js** utilize **WebAssembly** and **WebGPU** to run Large Language Models (LLMs) and computer vision models on the client's hardware.[46]

Use Cases:

- **Real-time Grammar Correction:** Running a small BERT model locally in a text editor.
- **Image Analysis:** Generating alt text for user-uploaded images before they are even sent to the server.
- **Privacy-First Chat:** An onboarding chatbot that runs entirely on the device, ensuring no user queries are logged.[48]

5.2 Implementation Best Practices

- **Do** use the **Cache API** to store model weights (which can be 50MB-500MB). Download them once, and they persist like a native app.
- **Do** use **Web Workers**. AI inference is CPU/GPU intensive. Running it on the main thread will freeze the UI, causing an INP failure.
- **Don't** assume every user has a GPU. Implement feature detection for WebGPU and fallback to a smaller, quantized model (WASM/CPU) or server-side inference if the hardware is insufficient.[48]

5.3 Ethics and Transparency

- **Transparency:** Clearly label AI-generated content or interactions.

- **Privacy:** If using local AI, explicitly market the privacy benefits. "Your data never leaves this device" is a powerful value proposition in 2025.[50]

---

6. Accessibility & Quality Assurance: No Exceptions

6.1 WCAG 2.2 Compliance

Web Content Accessibility Guidelines (WCAG) 2.2 introduced critical new success criteria that world-class sites must meet:

- **Focus Appearance (2.4.13):** Focus indicators must have a contrast ratio of at least 3:1 against the background and be of a minimum size. The browser default is often insufficient. **Do** create custom, high-visibility focus rings.[52]
- **Target Size (2.5.8):** All interactive targets must be at least 24x24 CSS pixels.
- **Dragging Movements (2.5.7):** Anything that requires dragging (e.g., a Kanban board) must have a single-pointer alternative (e.g., "Move Left/Right" buttons).[52]

Motion and Vestibular Disorders:
For sites using Scrollytelling or Parallax:

- **Do** respect @media (prefers-reduced-motion: reduce).
- **Don't** just slow down the animation. Stop it entirely or replace it with a simple fade. Parallax can cause severe nausea for some users.[12]

6.2 Automated Quality Assurance

Manual testing is dead. The complexity of modern apps requires automated governance.

Visual Regression Testing (VRT):
Tools like Percy and Chromatic are essential for preventing "CSS drift." They take screenshots of every component in every browser (Chrome, Firefox, Safari) on every commit and compare them pixel-by-pixel.

- *Chromatic:* Best for component libraries (Storybook users).
- *Percy:* Best for full-page integration testing.[54]

E2E Testing with Playwright:
Playwright is the 2025 standard for End-to-End testing, surpassing Cypress due to its speed, parallelism, and native WebKit support.

- **Do:** Use Playwright to test critical user flows (Login, Checkout) across all three browser engines.
- **Do:** Use codegen to generate tests by recording your interactions, then refine the code.[56]

---

7. Comprehensive Do's and Don'ts Matrix

The following matrix summarizes the report’s findings into actionable directives for the development team.

Category	DO	DON'T
Design	**Do** use Bento Grids with Container Queries for modular layouts.	**Don't** use Scrolljacking to override native scroll physics.
	Do use Glassmorphism sparingly with strict contrast checks (WCAG AA).	**Don't** rely on color alone for status indicators (Accessibility).
	Do use Fluid Typography (clamp()) for seamless scaling.	**Don't** use Neumorphism for text or complex data (readability risk).
Architecture	**Do** use Astro for content sites (Zero JS default).	**Don't** over-hydrate static content (Footer/Header).
	Do implement PWA Background Sync for form reliability.	**Don't** fetch data in a "waterfall" sequence; use parallel loaders.
Performance	**Do** optimize INP by yielding to the main thread (scheduler.yield).	**Don't** lazy-load the LCP image (loading="lazy").
	Do use content-visibility: auto for off-screen render skipping.	**Don't** use @import in CSS (blocks parallel downloading).
Security	**Do** use Nonce-based Strict CSP headers.	**Don't** use dangerouslySetInnerHTML without sanitization.
	Do validate all Server Action inputs with	**Don't** commit .env files or expose API secrets to the

	Zod/Valibot.	client.
AI & Edge	**Do** use Web Workers for client-side AI inference.	**Don't** run AI models on the main thread (freezes UI).
	Do cache AI models using the Cache API.	**Don't** download large models on cellular data without consent.
QA	**Do** use Playwright for cross-browser E2E testing.	**Don't** rely on manual QA for visual regression detection.

Conclusion

The "world-class" website of 2025 is a paradox: it is incredibly complex in its engineering yet effortless in its presentation. It achieves beauty through physics-based motion and spatial design, speed through edge computing and island architectures, and trust through zero-trust security and ethical AI implementation.

The differentiator is no longer just "responsive design" or "fast load times"—those are table stakes. The new frontier is the **intelligent, resilient, and inclusive web**. By adhering to the standards outlined in this report—specifically the rigorous application of Core Web Vitals, the adoption of local-first AI, and the non-negotiable compliance with accessibility standards—organizations can build digital platforms that are not merely modern, but enduringly excellent.

Works cited

1. Design Trends 2025: Glassmorphism, Neumorphism & Styles You Need to Know by Randall Carter - Contra, accessed on December 28, 2025, https://contra.com/p/PYkeMOc7-design-trends-2025-glassmorphism-neumorphism-and-styles-you-need-to-know
2. What Are the Top Web Design Trends of 2025? | TBH Creative, accessed on December 28, 2025, https://www.tbhcreative.com/blog/web-design-trends-of-2025/
3. “Neumorphism vs. Glassmorphism: The Future of UI Design Trends in 2025” | by Geetha, accessed on December 28, 2025, https://medium.com/design-bootcamp/neumorphism-vs-glassmorphism-the-future-of-ui-design-trends-in-2025-be8d44a97c36
4. The Best Web Design Trends of 2025 - Paddle Creative, accessed on December 28, 2025,

https://www.paddlecreative.co.uk/blog/the-best-web-design-trends-of-2025

5. 25 Top Web Design Trends 2025: From Neubrutalism to Dynamic UI - DepositPhotos Blog, accessed on December 28, 2025, https://blog.depositphotos.com/web-design-trends-2025.html
6. 9 Scrollytelling Examples in Business (+ Templates) - Storydoc, accessed on December 28, 2025, https://www.storydoc.com/blog/scrollytelling-examples
7. Scroll-driven animation timelines - CSS - MDN Web Docs, accessed on December 28, 2025, https://developer.mozilla.org/en-US/docs/Web/CSS/Guides/Scroll-driven_animations/Timelines
8. Animate elements on scroll with Scroll-driven animations | CSS and UI, accessed on December 28, 2025, https://developer.chrome.com/docs/css-ui/scroll-driven-animations
9. 12 engaging scrollytelling examples to inspire your content - Shorthand, accessed on December 28, 2025, https://shorthand.com/the-craft/engaging-scrollytelling-examples-to-inspire-your-content/index.html
10. A Beginner's Guide to Scrollytelling Website Design - Mockplus, accessed on December 28, 2025, https://www.mockplus.com/blog/post/scrollytelling-website-design
11. Web Animation 2025: Complete Guide To Motion Design Increasing User Engagement, accessed on December 28, 2025, https://mmcommunications.vn/en/web-animation-motion-design-guide-n607
12. Creating Accessible Parallax Websites - DubBot, accessed on December 28, 2025, https://dubbot.com/dubblog/2024/creating-accessible-parallax-websites.html
13. Responsive Typography: Best Practices for 2025, accessed on December 28, 2025, https://remtopx.com/blog/responsive-typography-best-practices/
14. Optimal Typography For Web Design In 2025 - Elegant Themes, accessed on December 28, 2025, https://www.elegantthemes.com/blog/design/optimal-typography-for-web-design
15. Variable fonts - CSS - MDN Web Docs, accessed on December 28, 2025, https://developer.mozilla.org/en-US/docs/Web/CSS/Guides/Fonts/Variable_fonts
16. Beyond Stillness: Navigating Motion Design with Fundamental Principles | Clay, accessed on December 28, 2025, https://clay.global/blog/web-design-guide/motion-design-principles
17. Motion | Apple Developer Documentation, accessed on December 28, 2025, https://developer.apple.com/design/human-interface-guidelines/motion
18. React Performance Optimization: Best Techniques for Faster, Smoother Apps in 2025, accessed on December 28, 2025, https://www.growin.com/blog/react-performance-optimization-2025/
19. Next.js vs Astro vs Remix: Choosing the Right Front-end Framework - Strapi, accessed on December 28, 2025, https://strapi.io/blog/nextjs-vs-astro-vs-remix-choosing-the-right-frontend-fram

ework
20. React Next.js Security Best Practices Guide 2025 - Kinplus Technologies Blog, accessed on December 28, 2025, https://blog.kinplusgroup.com/react-nextjs-security-best-practices-guide-2025/
21. Remix vs NextJS 2025 comparison - Merge Rocks, accessed on December 28, 2025, https://merge.rocks/blog/remix-vs-nextjs-2025-comparison
22. Remix vs Next.js vs Astro: Framework Comparison 2026 - Index.dev, accessed on December 28, 2025, https://www.index.dev/skill-vs-skill/remix-vs-nextjs-vs-astro
23. Next.js vs Astro vs Remix: Choosing the Right Front-end Framework | by Solomon Eseme, accessed on December 28, 2025, https://medium.com/strapi/next-js-vs-astro-vs-remix-choosing-the-right-front-end-framework-59f0e74c9d8e
24. Top 20 Mistakes React Native Developers Still Make in 2025 (And How to Fix Them), accessed on December 28, 2025, https://javascript.plainenglish.io/top-20-mistakes-react-native-developers-still-make-in-2025-and-how-to-fix-them-604f3fc1dbf9
25. Common useEffect anti-patterns I see in code reviews (and how to fix them) - Reddit, accessed on December 28, 2025, https://www.reddit.com/r/reactjs/comments/1pnkm1c/common_useeffect_antipatterns_i_see_in_code/
26. 15 React Anti-Patterns (and Fixes) You'll Actually Use - JsDev Space, accessed on December 28, 2025, https://jsdev.space/react-anti-patterns-2025/
27. Synchronize and update a PWA in the background - Microsoft Edge Developer documentation, accessed on December 28, 2025, https://learn.microsoft.com/en-us/microsoft-edge/progressive-web-apps/how-to/background-syncs
28. Background Sync in PWAs: Service Worker Guide - Zee Palm, accessed on December 28, 2025, https://www.zeepalm.com/blog/background-sync-in-pwas-service-worker-guide
29. Progressive Web App samples - Microsoft Edge Developer documentation, accessed on December 28, 2025, https://learn.microsoft.com/en-us/microsoft-edge/progressive-web-apps/samples/
30. Progressive Web Apps in 2025: Essential Features Every Website Needs - The Ad Firm, accessed on December 28, 2025, https://www.theadfirm.net/progressive-web-apps-in-2025-essential-features-every-website-needs/
31. Best practices for PWAs - Progressive web apps | MDN, accessed on December 28, 2025, https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Guides/Best_practices
32. How to improve Core Web Vitals in 2025: A complete guide - OWDT, accessed on December 28, 2025, https://owdt.com/insight/how-to-improve-core-web-vitals/
33. Introducing INP to Core Web Vitals | Google Search Central Blog, accessed on December 28, 2025,

https://developers.google.com/search/blog/2023/05/introducing-inp
34. 10+ New Optimizations For Your 2025 Core Web Vitals Strategy - NitroPack, accessed on December 28, 2025, https://nitropack.io/blog/core-web-vitals-strategy/
35. Faster Rendering with the content-visibility CSS Property - DebugBear, accessed on December 28, 2025, https://www.debugbear.com/blog/content-visibility-api
36. content-visibility - CSS - MDN Web Docs - Mozilla, accessed on December 28, 2025, https://developer.mozilla.org/en-US/docs/Web/CSS/Reference/Properties/content-visibility
37. 7 Simple but Powerful New CSS Features for 2025 | by Enigma of the Stack | Medium, accessed on December 28, 2025, https://medium.com/@cannon_circuit/7-simple-but-powerful-new-css-features-for-2025-ec48bac206a4
38. Content Security Policy (CSP) - HTTP - MDN Web Docs, accessed on December 28, 2025, https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/CSP
39. Content Security Policy - OWASP Cheat Sheet Series, accessed on December 28, 2025, https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html
40. What is Content Security Policy (CSP) | Header Examples - Imperva, accessed on December 28, 2025, https://www.imperva.com/learn/application-security/content-security-policy-csp-header/
41. Your Ultimate Website Security Checklist: 8 Steps For 2025 - Bruce & Eddy, accessed on December 28, 2025, https://www.bruceandeddy.com/website-security-checklist/
42. Guides: Data Security - Next.js, accessed on December 28, 2025, https://nextjs.org/docs/app/guides/data-security
43. Critical Security Vulnerability in React Server Components, accessed on December 28, 2025, https://react.dev/blog/2025/12/03/critical-security-vulnerability-in-react-server-components
44. HTTP Headers - OWASP Cheat Sheet Series, accessed on December 28, 2025, https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Headers_Cheat_Sheet.html
45. Top 7 Web Security Practices Every Developer Should Follow in 2025 - Zalvice, accessed on December 28, 2025, https://zalvice.com/blog/top-7-web-security-practices-2025
46. Platform and environment | TensorFlow.js, accessed on December 28, 2025, https://www.tensorflow.org/js/guide/platform_environment
47. A Guide to In-Browser LLMs - Intel, accessed on December 28, 2025, https://www.intel.com/content/www/us/en/developer/articles/technical/web-developers-guide-to-in-browser-llms.html
48. Client-side AI for Cost-Effective, Secure Web Apps - Grid Dynamics, accessed on

December 28, 2025, https://www.griddynamics.com/blog/client-side-ai
49. Improve performance and UX for client-side AI - web.dev, accessed on December 28, 2025, https://web.dev/articles/client-side-ai-performance
50. Ethical AI Content Creation: NP Digital's Guide 2025 - Neil Patel, accessed on December 28, 2025, https://neilpatel.com/blog/ethical-ai-content-creation/
51. AI, Ethics, and the Future of UI/UX in 2025 | by Hriday Checker - Medium, accessed on December 28, 2025, https://waffledesigns.medium.com/ai-ethics-and-the-future-of-ui-ux-in-2025-e9195a2f78de
52. Web Content Accessibility Guidelines (WCAG) 2.2 - W3C, accessed on December 28, 2025, https://www.w3.org/TR/WCAG22/
53. All WCAG 2.2 Techniques | WAI - W3C, accessed on December 28, 2025, https://www.w3.org/WAI/WCAG22/Techniques/
54. Percy vs Chromatic: Which visual regression testing tool to use? | by Crissy Joshua, accessed on December 28, 2025, https://medium.com/@crissyjoshua/percy-vs-chromatic-which-visual-regression-testing-tool-to-use-6cdce77238dc
55. 15 Best Visual Testing Tools: Complete Overview - Functionize, accessed on December 28, 2025, https://www.functionize.com/automated-testing/visual-testing-tools
56. Playwright vs. Cypress: The Ultimate 2025 E2E Testing Showdown - Frugal Testing, accessed on December 28, 2025, https://www.frugaltesting.com/blog/playwright-vs-cypress-the-ultimate-2025-e2e-testing-showdown
57. Playwright vs Cypress: Key Differences, and When to Use Each | LambdaTest, accessed on December 28, 2025, https://www.lambdatest.com/blog/cypress-vs-playwright/