Web Accessibility

Version WCAG 2.2 · verified 2026-11-02

WCAG 2.2 AA for frontend engineers — semantics, ARIA, keyboard and focus, forms, contrast, what 2.2 added, and the EAA in one table.

Contents

Mental model 2

Semantic HTML first 2

ARIA only when needed 3

Keyboard and focus 4

Forms and errors 6

Images, color and motion 7

What WCAG 2.2 added 8

The EAA and EN 301 549 9

Testing 10

Further reading 11

Mental model

Assistive technology never sees your pixels; it reads the accessibility tree, which the browser builds from your HTML: every node has a role, a name and a state. Native elements fill that tree correctly for free, so most accessibility work is choosing the right element and keeping its state true. ARIA only patches the tree, and a wrong patch is worse than none.

Semantic HTML first

Use	Not	You get free
`<button>`	`<div onclick>`	Focus, Enter/Space, role
`<a href>`	`<span onclick>`	Focus, Enter, “link” role
`<label for>`	Placeholder text	Accessible name, bigger target
`<nav>`, `<main>`, `<header>`	`<div class>`	Landmarks to jump between
`<h1>`–`<h6>` in order	Bold text	Heading navigation
`<dialog>` + `showModal()`	Custom overlay	Focus move, Escape, inert page

Screen-reader users navigate by headings and landmarks first, so the outline is the page’s table of contents. A button acts, a link navigates: pick by behaviour, not by look.

```
<header><nav aria-label="Main">…</nav></header>
<main>
```

```
  <h1>Orders</h1>
  <section aria-labelledby="open">
    <h2 id="open">Open orders</h2>
    <button type="button">Export CSV</button>
  </section>
</main>
```

> **Gotcha:** A `<div>` with a click handler is invisible to keyboard users: it takes no focus and ignores Enter and Space. Adding `role="button"` fixes neither; you would also need `tabindex="0"` and key handlers. Use `<button>`.

ARIA only when needed

Attribute	Use for
`aria-label`	Name when no visible text exists
`aria-labelledby`	Name from visible text elsewhere
`aria-describedby`	Extra description (hints, errors)
`aria-expanded`	Disclosure open/closed state
`aria-current="page"`	Current item in a set
`aria-live="polite"`	Announce updates in place

The first rule of ARIA is not to use it when a native element does the job; the APG puts it as “no ARIA is better than bad ARIA”. ARIA changes what is announced, never behaviour: `role="tab"` adds no arrow keys.

```
<button aria-expanded="false" aria-controls="faq1">
  Shipping
</button>
<div id="faq1" hidden>Ships in 2 days.</div>

<button aria-label="Close">
  <svg aria-hidden="true">…</svg>
</button>
```

> **Gotcha:** `aria-hidden="true"` on an element that holds focus leaves keyboard users on an element the screen reader cannot see. Use `inert` to hide and disable a whole region.

Keyboard and focus

Key or attribute	Effect
`tabindex="0"`	Adds to tab order, DOM position
`tabindex="-1"`	Focusable by script only
`tabindex="1"` +	Never: breaks tab order
Arrows in a widget	Move within tabs, menus, grids
Escape	Closes dialog or popup
`:focus-visible`	Style the keyboard focus ring

Tab moves between widgets; arrow keys move inside one (a tablist, a menu, a listbox). The APG calls this roving `tabindex`: only the active item has `tabindex="0"`. When content opens, move focus into it; when it

closes, return focus to the opener. A modal `<dialog>` does both for you; a custom popup must do it by hand.

```
const tabs = [
  ...document.querySelectorAll('[role="tab"]'),
];
const step = { ArrowRight: 1, ArrowLeft: -1 };
tabs.forEach((tab, i) => {
  tab.addEventListener("keydown", (e) => {
    const d = step[e.key];
    if (!d) return;
    const n = tabs.length;
    const next = tabs[(i + d + n) % n];
    tab.tabIndex = -1;
    next.tabIndex = 0;
    next.focus(); // wraps last -> first
  });
});
```

> **Warning:** `outline: none` without a replacement removes the only cue a keyboard user has. Restyle `:focus-visible` instead, with at least 3:1 contrast against the background.

Forms and errors

Need	Do
Name every field	`<label for>` or wrapping `<label>`
Group radios	`<fieldset>` + `<legend>`
Hint or error text	`aria-describedby`
Mark the invalid field	`aria-invalid="true"`
Autofill personal data	`autocomplete="email"` etc.
Announce form errors	Focus a summary or `role="alert"`

An error must say what went wrong in text, next to the field, and be programmatically tied to it so a screen reader reads it on focus. `autocomplete` tokens satisfy SC 1.3.5 and spare users retyping.

```
<label for="email">Email</label>
<input id="email" type="email"
  autocomplete="email" required
  aria-invalid="true"
  aria-describedby="email-err">
<p id="email-err">Enter an address like a@b.co</p>
```

> **Gotcha:** Placeholder text is not a label. It disappears as soon as the user types, and its default gray usually fails contrast.

Images, color and motion

Requirement	Threshold
Body text contrast (1.4.3)	4.5:1
Large text: 18pt, or 14pt bold	3:1
UI parts and graphics (1.4.11)	3:1
Meaning by color alone (1.4.1)	Never
Decorative image	`alt=""`

Alt text says what the image means in context, not what it looks like; a decorative image gets an empty `alt` so it is skipped. Pair every color cue with text, an icon or a pattern. Honour the user’s motion setting for animation that is not essential.

```
.error { color: #b00020; }
.error::before { content: "Error: "; }

@media (prefers-reduced-motion: reduce) {
  *, *::before, *::after {
    animation-duration: 0.01ms !important;
    transition-duration: 0.01ms !important;
    scroll-behavior: auto !important;
  }
}
```

> **Gotcha:** A missing `alt` is not the same as `alt=""`. Without the attribute, many screen readers read the file name instead.

What WCAG 2.2 added

Criterion	Level	Requires
2.4.11 Focus Not Obscured (Min)	AA	Focus never fully hidden
2.5.7 Dragging Movements	AA	Single-pointer alternative
2.5.8 Target Size (Minimum)	AA	24×24 CSS px, or spacing
3.3.8 Accessible Authentication	AA	No memory or puzzle test
3.2.6 Consistent Help	A	Help in the same place
3.3.7 Redundant Entry	A	Don’t ask twice in a flow

WCAG 2.2 keeps every 2.1 criterion except 4.1.1 Parsing, which is now obsolete and removed. 2.2 also adds three AAA criteria (2.4.12, 2.4.13, 3.3.9). W3C states that content conforming to 2.2 also conforms to 2.1 and 2.0, though a policy that cites 2.1 may still expect 4.1.1 tested.

```
/* 2.4.11: sticky header must not cover focus */
html { scroll-padding-top: 4rem; }
header { position: sticky; top: 0; height: 4rem; }

/* 2.5.8: icon buttons hit 24x24 at minimum */
.icon-btn { min-width: 24px; min-height: 24px; }
```

> **Tip:** 3.3.8 accepts allowing paste and password-manager autofill in login fields as the mechanism. Blocking paste fails it.

The EAA and EN 301 549

Item	Status on 2 Nov 2026
EAA (Directive 2019/882)	Applies since 28 Jun 2025
Products already used in services	Transition until 28 Jun 2030
Microenterprise services	Exempt (<10 staff, ≤ €2M)
EN 301 549 V3.2.1 (2021)	Maps to WCAG 2.1 AA
EN 301 549 V4.1.1 (2026-09)	Maps to WCAG 2.2 AA

The EAA covers consumer products and services sold in the EU, including e-commerce, banking, e-books, transport ticketing and electronic communications. EN 301 549 is the technical standard that turns it into testable clauses. V4.1.1 adds an annex mapping to the EAA but gives a presumption of conformity only once it is cited in the Official Journal; until then V3.2.1 remains the cited reference.

```
Target today:   WCAG 2.2 Level AA
Satisfies:      WCAG 2.1 AA (EN 301 549 V3.2.1)
Also satisfies: EN 301 549 V4.1.1 web clauses
Document it:    accessibility statement
```

> **Note:** The EAA binds the business that offers the product or service in the EU, wherever its developers sit.

Testing

Layer	Tool	Catches
Automated	axe-core, Lighthouse	Missing names, contrast, ARIA misuse
Keyboard	Tab, Shift+Tab, Esc	Traps, lost focus, order
Screen reader	NVDA, JAWS, VoiceOver	Names, states, announcements
Zoom	200% text, 320px width	Reflow (1.4.10), clipping

Automated rules catch only the machine-checkable subset; whether alt text is meaningful or focus order makes sense needs a human. Run axe in CI on every key page and do a keyboard-only pass on every new component.

```
import { test, expect } from "@playwright/test";
import { AxeBuilder } from "@axe-core/playwright";

const tags = [
  "wcag2a", "wcag2aa", "wcag21a",
  "wcag21aa", "wcag22aa",
];
test("home: WCAG 2.2 AA", async ({ page }) => {
  await page.goto("/");
  const res = await new AxeBuilder({ page })
    .withTags(tags)
```

```
    .analyze();
  expect(res.violations).toEqual([]);
});
```

> **Gotcha:** axe tags are not cumulative. `wcag22aa` alone runs only the rules new in 2.2; list every level you need.

Further reading

- W3C — WCAG 2.2
- W3C — What's new in WCAG 2.2
- W3C — ARIA Authoring Practices Guide
- EUR-Lex — Directive (EU) 2019/882
- ETSI — EN 301 549 V4.1.1
- axe-core — rule tags