

What WCAG requires for **keyboard navigation**, and how to check it

For many sites, keyboard access is a legal requirement. The criteria, the common mistakes, their fixes and a manual check →

The people who navigate without a mouse

Keyboard navigation is using a website without a mouse: moving between links, buttons and form fields, and activating them, only with the keyboard.

Many people **depend on it:**

- People with motor impairments
- People with a temporary injury
- Screen reader users
- People who simply prefer the keyboard

The keyboard moves one focus through the page

The browser keeps one element **in focus** at a time, usually marked with an outline. The keyboard moves that focus from one interactive element to the next, **following the order in the HTML**, and acts on the focused element.

- **Tab / Shift+Tab:** next / previous interactive element
- **Enter:** follow a link or press a button
- **Space:** press a button, check a checkbox
- **Arrows:** move inside menus, selects, radio groups and tabs
- **Escape:** close menus, dropdowns and modals

How focus moves through a page



Each Tab moves the focus to the next stop, starting with the skip link. The down arrow opens the submenu and moves the focus into its first link.

Keyboard access is a legal requirement for many sites

Since the European Accessibility Act came into force in June 2025, meeting **WCAG AA** has been a legal requirement for many sites.

The EAA applies to specific services, such as e-commerce, banking, transport and e-books, and exempts microenterprises.

It references **EN 301 549**, which maps to **WCAG 2.1 AA**.

The main WCAG criteria for keyboard use

- **2.1.1 Keyboard (A):** everything works with the keyboard.
- **2.1.2 No Keyboard Trap (A):** focus can always move away from any element.
- **2.4.1 Bypass Blocks (A):** there is a way to skip repeated blocks like the header.
- **2.4.3 Focus Order (A):** focus moves in a logical order.
- **2.4.7 Focus Visible (AA):** the focused element is always visible.
- **2.4.11 Focus Not Obscured (AA, WCAG 2.2):** sticky headers, cookie banners or chat widgets never fully hide the focused element.

Other criteria also touch keyboard use, like dismissing content shown on focus or exposing the state of custom controls.

Open submenus add extra key presses on every page

A common source of keyboard problems: something that works fine with a mouse was **never tested with the keyboard.**

In the header menu, submenus open as soon as the parent item receives focus, or stay hidden only visually but remain in the tab order.

Every Tab walks through every child link, **on every page,** before reaching the main content.

More mistakes that block keyboard users

The result is extra Tab presses, a lost focus, or an element that **cannot be reached at all**.

- **Focus outline removed:** `outline: none` without a replacement leaves no way to know where focus is.
- **Modals:** focus does not move into the modal when it opens, or it cannot leave it.
- **No skip link:** every page starts with the whole header.
- **Fake buttons:** a `<div>` or `<span>` with a click handler is not focusable and does not respond to Enter or Space.

The technical fix for the header menu

Each fix is a change in the site's **HTML, CSS or JavaScript**.

- Submenus stay closed until opened with **Enter, Space or the down arrow**.
- A closed submenu is **out of the tab order** (`hidden` or `display: none`), so Tab jumps to the next top-level item.
- The toggle is a `<button>` with `aria-expanded="true|false"`, so screen readers announce the state.
- **Escape** closes the open submenu and returns focus to its toggle.

The technical fix for the other mistakes

- **Focus outline:** use `:focus-visible` to style the outline instead of removing it, so the focused element stays visible.
- **Modals:** focus goes inside on open, stays inside while open, and returns to the trigger on close. Escape closes it.
- **Skip link:** a "Skip to main content" link as the **first focusable element**.
- **Fake buttons:** use `<button>` and `<a>`, which are focusable and respond to the keyboard.

Checking a site with the mouse out of reach

Keyboard navigation is one of the easiest accessibility checks to run. It needs **no tools**.

1. Put the mouse away and press Tab from the address bar.
2. The first stop should be the skip link.
3. Focus should be visible on every stop.
4. Tab through the header: closed submenus should be skipped. Open one with Enter, close it with Escape.
5. Open a modal, close it with Escape, and check focus returns to the button that opened it.
6. Fill in and submit a form.

Automated tools like Lighthouse catch only part of this. **The manual check is the reliable one.**

A requirement for users and for WCAG compliance

Keyboard navigation is a requirement, both for **the people who depend on it** and for **WCAG compliance**.

Submenus in the tab order, a removed focus outline, broken modals, no skip link and fake buttons all block keyboard users, and none of them needs a tool to find.

The manual check shows where a site stands.

The fixes make the site quicker to use for everyone who relies on the keyboard, **whatever the reason**.



Can your website be used without a mouse?

At **Metadrop** we check and fix keyboard navigation as part of accessibility work. No jargon, no fluff.

Follow us for more on web accessibility and Drupal.



Save



Share



Comment

