A RESOURCE FOR DESIGN SYSTEM, QA AND ENGINEERING TEAMS

Keyboard-only QA checklist.

Put the mouse away. Ship what still works.

Walk the page. Script the widgets. Keep the order.

28	29	12
checks, each with its test	scripted widget checks, all passing on the clean page	planted bugs, all 12 found

If it cannot be done with a keyboard, it cannot be done.

The essay this guide accompanies ends with a ten-point keyboard checklist and one instruction: when an item fails, fix it and add it to your automated tests. This guide is that checklist grown up: 28 checks, each traced to a WCAG 2.2 success criterion or an ARIA Authoring Practices (APG) pattern, and most of them already automated.

Two tools do the automated part. A Tab walk presses Tab and Shift+Tab through a page and records every stop: the focus order, traps, elements no key reaches, positive tabindex, invisible focus and focus hidden under sticky content. Widget scripts, stored as JSON, press the keys the APG requires for dialogs, menu buttons, tabs, comboboxes, listboxes and grids, and assert where focus lands.

Both run against the Acme Team settings page from Field Guides 10 and 11, in a clean version and a broken one with 12 planted bugs. The walk finds 6 of them and the widget scripts the other 6; neither finds all 12 alone. A person still judges whether the recorded order makes sense.

Version 1.0 / Sources checked 24 September 2026

Field Guide 12 of the Design × AI series. Verified 24 September 2026 with Node 22.22 and Playwright 1.58.2 and 1.63.0 with their bundled Chromium (identical results), viewport 1280 by 720.

Practical guidance, not legal advice. WCAG references are to WCAG 2.2 (W3C Recommendation, 12 December 2024). APG patterns are recommendations, not requirements; the scripts assert only behaviour the APG lists as required and name the optional keys for manual testing. Prepared with AI assistance and edited by hand.

START HERE

Keys, criteria and the two tools.

Learn what each key should do, then pick the route. Every check below names the key it needs and the tool that proves it.

Key	What it should do	Where it applies
Tab / Shift+Tab	Move to the next or previous stop; never stick	The whole page
Enter	Follow a link; activate a button or menu item; accept an option	Links, buttons, menus, comboboxes
Space	Activate a button; toggle a checkbox; open a menu button	Buttons, checkboxes, menu buttons
Arrow keys	Move inside one widget	Tabs, menus, listboxes, grids, radio groups
Home / End	First or last item	Grids; optional in tabs, menus, listboxes
Escape	Close what just opened and return focus	Dialogs, menus, combobox popups

Testing a page by hand

Sections 01 and 03, in order. Ten minutes, keyboard only, with the recorded focus order from the Tab walk beside you.

Building a widget

Section 02 and `widgets.keyboard.json`. Bind the script to your markup and run it before the pull request.

Automating the pass

Section 04. Run `tab-walk.mjs` and `run-keyboard-scripts.mjs` on key routes in CI; review the focus order diff.

Triaging a bug report

Every check names its WCAG criterion, so a failure maps straight to a severity and an owner.

Label	Meaning
`TAB WALK`	Found by `tab-walk.mjs`: Tab and Shift+Tab through the page.
`SCRIPT`	Asserted by a step in `widgets.keyboard.json`, run by `run-keyboard-scripts.mjs`.
`APG`	Follows an ARIA Authoring Practices pattern.
`WCAG A / AA`	Traces to a WCAG 2.2 success criterion at that level.
`MANUAL`	A person has to judge it; the evidence line says what to record.
`PRACTICE`	A working method with a review signal rather than a hard gate.

SECTION 01

The page: reach everything, lose nothing.

Start at the address bar and press Tab until you are back where you began. Then do it with Shift+Tab. The Tab walk does the same and writes down every stop.

Suggested owners: QA lead + engineering lead

- [ ] **Every control can be reached with Tab**

  K01 TAB WALK WCAG A

  Anything that responds to a click must also take keyboard focus. A `<div onclick>` does not; a `<button>` does, with Enter and Space for free.

  Evidence: Walk `unreachable`: on the broken fixture, the "Export members" div. The walk flags elements with a click handler, an interactive role or a pointer cursor that no Tab reaches.

  SC 2.1.1 Keyboard [S01].

- [ ] **Tab never gets stuck**

  K02 TAB WALK WCAG A

  Focus can always leave a component with the keyboard alone. An editor that uses Tab for indentation must say how to leave (for example Escape, then Tab).

  Evidence: Walk `trap`: the broken fixture's Team notes editor swallows Tab and Shift+Tab. The walk reports it, jumps past it and carries on, so nothing after it goes unchecked.

  SC 2.1.2 No Keyboard Trap [S02].

- [ ] **The first Tab reaches a visible skip link**

  K03 SCRIPT WCAG A

  It appears on focus and moves focus to the main content. Hidden with `display: none`, it helps nobody.

  Evidence: Script steps skip.01 (focused and inside the viewport) and skip.02 (focus lands in `main`).

  SC 2.4.1 Bypass Blocks [S01].

- [ ] **Focus order follows the reading order**

  K04 TAB WALK MANUAL WCAG A

  Top to bottom, left to right in left-to-right languages. CSS that reorders visually (`row-reverse`, `order`, grid placement) leaves the Tab order behind.

  Evidence: Walk `order` warnings when focus moves up or leftwards: 2 on the broken fixture's reversed navigation. A person reads the recorded order and signs it off.

  SC 2.4.3 Focus Order [S03].

- [ ] **No positive tabindex**

  K05 TAB WALK WCAG A

  `tabindex="1"` and above pull an element to the front of the whole page's order. Use DOM order; `tabindex="0"` and `-1` only.

  Evidence: Walk `tabindex`: the broken fixture's Billing link comes first, before the logo.

  SC 2.4.3, failure F44 [S03].

- [ ] **Focus is always visible**

  K06 TAB WALK WCAG AA

  Every stop changes something on screen. How much it has to change is Field Guide 10's subject.

  Evidence: Walk `invisible`: focusing the element changes no pixel. 4 header links on the broken fixture.

  SC 2.4.7 Focus Visible [S14]; Field Guide 10.

- [ ] **Focus is never hidden under sticky content**

  K07 TAB WALK WCAG AA

  Shift+Tab scrolls the focused element to the top edge, straight under a sticky header. Reserve the header's height with `scroll-padding`.

  Evidence: Walk `obscured`: all sampled points covered. The three tabs on the broken fixture, found only by the Shift+Tab pass.

  SC 2.4.11 Focus Not Obscured (Minimum) [S04].

- [ ] **Composite widgets are one tab stop**

  K08 TAB WALK APG

  A tab list, listbox or grid takes one Tab; arrow keys move inside it. The clean fixture has 12 stops; the broken one has 22, because its 3 tabs and 9 grid cells are all stops.

  Evidence: Walk stop count and the recorded order; script steps tabs.06 and grid.01.

  APG tabs [S09] and grid [S12].

- [ ] **Nothing hidden takes focus**

  K09 TAB WALK WCAG AA

  Closed menus, collapsed panels and off-screen drawers are removed from the order (`hidden`, `inert`), not only moved off-screen.

  Evidence: A stop outside the viewport counts as covered, so the walk reports it as `obscured`.

  SC 2.4.3 [S03]; SC 2.4.11 [S04].

- [ ] **Focus never changes the context by itself**

  K10 MANUAL WCAG A

  Landing on a control does not submit, navigate or open a new window. Changing a select does not reload the page.

  Evidence: Manual: note any stop where focus alone changed the page.

  SC 3.2.1 On Focus [S01].

The recorded focus order is the page's keyboard map. Review it like a design.

SECTION 02

Widgets: the keys the APG requires.

Each check is one or more steps in `widgets.keyboard.json`. The runner stops a widget at its first failure, because every later step depends on where focus was left.

Suggested owners: Design-system lead + engineering lead

- [ ] **Dialog: opening moves focus inside**

  K11 SCRIPT APG

  To the first field, or to a static element at the top when the dialog opens on long content.

  Evidence: Step dialog.02. The broken fixture's dialog opens and leaves focus on the button.

  APG modal dialog [S06].

- [ ] **Dialog: the page behind is out of reach**

  K12 SCRIPT APG WCAG A

  Tab and Shift+Tab never reach content behind a modal dialog. The APG wraps focus inside; a native `<dialog>` opened with `showModal()` makes the page inert and may let focus reach the browser toolbar, which the script accepts.

  Evidence: Steps dialog.03 and dialog.04: six presses each, focus never inside `main` or `header`.

  APG [S06]; SC 2.1.2 allows confinement you can leave [S02].

- [ ] **Dialog: Escape closes it and focus goes back**

  K13 SCRIPT APG

  Focus returns to the control that opened the dialog, unless that control is gone.

  Evidence: Step dialog.05.

  APG [S06].

- [ ] **Menu button: Enter and Space open it**

  K14 SCRIPT APG

  Focus lands on the first item; Down Arrow moves to the next.

  Evidence: Steps menu.02, menu.03 and menu.05.

  APG menu button [S07]; menu [S08].

- [ ] **Menu: Escape and Tab close it**

  K15 SCRIPT APG

  Escape closes the menu, returns focus to the button and sets `aria-expanded="false"`. Tab leaves the menu and closes it.

  Evidence: Steps menu.04 and menu.06. The broken fixture's menu ignores Escape.

  APG menu [S08].

- [ ] **Tabs: arrows move and wrap**

  K16 SCRIPT APG

  Right and Left Arrow move between tabs and wrap at the ends; Tab leaves the tab list in one press.

  Evidence: Steps tabs.02 to tabs.06. On the broken fixture tabs.02 fails and the rest are blocked.

  APG tabs [S09].

- [ ] **Combobox: focus stays put, the option moves**

  K17 SCRIPT APG

  DOM focus stays on the combobox while `aria-activedescendant` points at the active option. Enter accepts it; Escape closes the popup without changing the value.

  Evidence: Steps combo.02 to combo.07, on the Role combobox in the Invite teammate dialog. The broken version ignores Enter.

  APG combobox [S10].

- [ ] **Listbox: arrows move the active option**

  K18 SCRIPT APG

  On focus, the selected option, or the first, is active. Down and Up Arrow move it.

  Evidence: Steps listbox.01 to listbox.03. Passes on both fixtures: the broken page is not broken everywhere.

  APG listbox [S11].

- [ ] **Grid: one stop, arrows, Home, Control+Home and End**

  K19 SCRIPT APG

  Arrow keys move one cell; Home goes to the start of the row; Control+Home and Control+End go to the first and last cell.

  Evidence: Steps grid.01 to grid.07. The broken grid has 9 tab stops and no arrow keys.

  APG grid [S12].

- [ ] **Every custom control has a written keyboard table**

  K20 PRACTICE WCAG A

  Sliders, date pickers and drag handles without an APG example still need one row per key, agreed before the build and turned into a script.

  Evidence: The component contract's keyboard table (Field Guide 02, item 05.03) and a script in `widgets.keyboard.json`.

  SC 2.1.1 [S01]; APG patterns [S06].

If the keyboard table is not written down, the widget is not finished.

SECTION 03

Beyond Tab: drags, shortcuts, disappearing content.

The failures that Tab never touches. Test them by hand, then script the ones you find.

Suggested owners: QA lead + product designer

- [ ] **Every drag has a keyboard path and a click path**

  K21 MANUAL WCAG A WCAG AA

  Reordering members by drag needs keyboard operation (SC 2.1.1) and a single-pointer way that is not a drag (SC 2.5.7): move up and down buttons do both. A keyboard alternative alone does not meet 2.5.7.

  Evidence: Manual: complete each drag task with keys only, then with single clicks only.

  SC 2.5.7 Dragging Movements, AA [S05]; SC 2.1.1 [S01].

- [ ] **Single-key shortcuts can be turned off**

  K22 MANUAL WCAG A

  A shortcut on a letter, number or symbol key alone can be turned off, remapped, or works only while its component has focus. Speech-input users trigger them by accident.

  Evidence: Manual: list every single-key shortcut and where it is turned off.

  SC 2.1.4 Character Key Shortcuts [S01].

- [ ] **Buttons and links do what their role promises**

  K23 MANUAL WCAG A

  Enter follows a link; Enter and Space activate a button. A link styled as a button that ignores Space, or a button that navigates, surprises everyone.

  Evidence: Manual: Enter and Space on every stop in the recorded order.

  SC 2.1.1 [S01]; SC 4.1.2 [S01].

- [ ] **Focus lands somewhere sensible when content disappears**

  K24 MANUAL APG

  After "Remove from team" deletes Ana's row, focus moves to the next row's actions, not to the top of the page. When the invoking element no longer exists, choose the next logical place.

  Evidence: Manual, then a script step asserting the new focus target.

  APG modal dialog, focus on close [S06]; SC 2.4.3 [S03].

A keyboard bug found by hand is a script step you have not written yet.

SECTION 04

Gates: make the pass repeatable.

A manual pass finds the bugs once. The kit keeps them found.

Suggested owners: Engineering lead + QA lead

- [ ] **Record the focus order of every key route**

  K25 TAB WALK PRACTICE

  `tab-walk.mjs --json` writes the order. Commit it; a pull request that changes it shows the diff, and a person approves the new order.

  Evidence: A `focus-order.json` per route in the repository.

  Kit script.

- [ ] **Run the widget scripts in CI**

  K26 SCRIPT PRACTICE

  One bindings file per page maps script targets to selectors; the same scripts cover every page that uses the widget.

  Evidence: `run-keyboard-scripts.mjs <url> --bindings <page>.json` exits 1 on any failure: 29 of 29 on the clean fixture; 7 pass, 6 fail and 16 blocked on the broken one.

  Kit script; Playwright keyboard API [S13].

- [ ] **A person does the pass once per release**

  K27 MANUAL PRACTICE

  Keyboard only, on the release candidate, with a screen reader for the dialog and the combobox. Automation checks what the APG says; a person checks whether it makes sense.

  Evidence: The review record at the end of this guide.

  Recommended practice.

- [ ] **Every keyboard bug becomes a test**

  K28 PRACTICE

  The essay's closing rule: file it, fix it, add it to the suite. Here that means a step in `widgets.keyboard.json` or a route in the walk.

  Evidence: Each closed keyboard bug links the step or route that now covers it.

  The essay's checklist.

Automate the keys. Keep the judgement human.

APPENDIX A

A widget script, and what it binds to.

Scripts use target names; the bindings file turns them into selectors for one page. Seven widgets, 29 asserting steps.

widgets.keyboard.json (tabs, excerpt)

```
{
  "widget": "tabs",
  "apg": "https://www.w3.org/WAI/ARIA/apg/patterns/tabs/",
  "sc": ["2.1.1", "2.4.3", "4.1.2"],
  "steps": [
    { "id": "tabs.01", "do": "focus", "target": "tabs.first" },
    { "id": "tabs.02", "do": "press", "key": "ArrowRight",
      "expect": [{ "focus": "tabs.second" },
                 { "attr": ["tabs.second", "aria-selected", "true"] },
                 { "visible": "tabs.second.panel" }],
      "why": "Right Arrow moves to the next tab (and activates it, with automatic activation)."
},
    { "id": "tabs.04", "do": "press", "key": "ArrowLeft",
      "expect": [{ "focus": "tabs.last" }], "why": "Left Arrow on the first tab wraps to the
last." },
    { "id": "tabs.06", "do": "press", "key": "Tab",
      "expect": [{ "notFocus": "tabs.all" }], "why": "Tab leaves the tab list in one step: the
tab list is a single tab stop." }
  ],
  "optional": ["Home and End move to the first and last tab."]
}
```

Expectation	Passes when
`focus`, `notFocus`	The active element is (or is not) the target
`focusWithin`, `notWithin`	The active element is (or is not) inside the target
`visible`, `hidden`, `inViewport`	The target is rendered, not rendered, or fully on screen
`attr`	An attribute has the value, such as `aria-expanded="false"`
`activeDescendant`	`aria-activedescendant` on the owner points at the option
`text`	The target's text equals another target's text
`tabStops`	The container holds exactly n tab stops

`fixtures/bindings.json` maps each target name, such as `tabs.second.panel`, to a selector such as `#panel-invitations`.

APPENDIX B

One page, clean and broken.

Both tools on both fixtures in Chromium at 1280 by 720. The clean page passes everything; the broken page carries 12 planted bugs, each marked BUG in its source.

terminal (clean fixture)

```
$ node scripts/tab-walk.mjs fixtures/team-settings.html
Focus order (Tab):
   1. a "Skip to main content"            7. button#invite-open "Invite teammate"
   2. a "Acme"                            8. listbox#member-filter "All members ..."
   3. a "Projects"                        9. button#actions-ana "Actions for Ana Silva"
   4. a "Team"                           10. button#actions-ben "Actions for Ben Okafor"
   5. a "Billing"                        11. gridcell "24 Sep 2026"
   6. tab#tab-members "Members"          12. a "Help with team settings"
12 stops forward, 12 backward, 0 failures, 0 warnings
```

Printed in two columns, with one label shortened, for space. The widget scripts report 29 checks, 29 pass.

Planted bug	Found by	Result
Team notes editor swallows Tab	Tab walk	`trap` forwards and backwards (SC 2.1.2)
"Export members" is a clickable div	Tab walk	`unreachable` (SC 2.1.1)
Billing link has `tabindex="1"`	Tab walk	`tabindex`; it becomes stop 1 (SC 2.4.3)
Header links `outline: none`	Tab walk	`invisible` on 4 links (SC 2.4.7)
No `scroll-padding` under a sticky header	Tab walk	`obscured` on 3 tabs, Shift+Tab pass only (SC 2.4.11)
Navigation in `row-reverse`	Tab walk	2 `order` warnings for a person to judge (SC 2.4.3)
Skip link `display: none`	Widget scripts	skip.01 fails; 1 step blocked
Dialog without focus move or Escape	Widget scripts	dialog.02 fails; 3 steps blocked
Menu ignores Escape	Widget scripts	menu.04 fails; 2 steps blocked
Tabs without arrow keys	Widget scripts	tabs.02 fails; 4 steps blocked
Combobox ignores Enter and Escape	Widget scripts	combo.04 fails; 1 step blocked
Every grid cell a tab stop, no arrows	Widget scripts	grid.01 fails; 5 steps blocked

Walk: 22 stops each way, 11 failures, 3 warnings (one partly obscured button). Scripts: 7 pass, 6 fail, 16 blocked. The listbox is correct on both pages.

KEEP WITH THE RELEASE

Leave a keyboard QA record.

One record per key route per release. It is the evidence that someone put the mouse away.

Route / release	**Tester / date / browser**
Focus order file (link) and diff approved by	**Tab walk run (link)**
Widget script run (link)	**Manual checks K10, K21 to K24 (notes)**
Screen reader used on dialog and combobox	**Bugs filed (links)**
Script steps added for fixed bugs	**Owner / next review**

> **A blocked step is not a pass.**
>
> The runner stops a widget at its first failure. Fix that step and run again before reading anything into the steps after it.

SOURCES / MAINTENANCE

Keep the guide current.

Sources checked 24 September 2026. Every number in the guide was produced by running the kit on that date with the versions on the cover.

S01 / W3C, WCAG 2.2
W3C Recommendation, 12 December 2024. SC 2.1.1, 2.1.4, 2.4.1, 3.2.1 and 4.1.2.
https://www.w3.org/TR/WCAG22/

S02 / W3C, Understanding SC 2.1.2 No Keyboard Trap
Confinement is allowed when the user knows how to leave it.
https://www.w3.org/WAI/WCAG22/Understanding/no-keyboard-trap.html

S03 / W3C, Understanding SC 2.4.3 Focus Order
Order preserves meaning; failures F44 (tabindex) and F85 (dialogs).
https://www.w3.org/WAI/WCAG22/Understanding/focus-order.html

S04 / W3C, Understanding SC 2.4.11 Focus Not Obscured (Minimum)
Sticky headers; scroll-padding as a sufficient technique.
https://www.w3.org/WAI/WCAG22/Understanding/focus-not-obscured-minimum.html

S05 / W3C, Understanding SC 2.5.7 Dragging Movements
AA; a keyboard alternative alone does not satisfy it.
https://www.w3.org/WAI/WCAG22/Understanding/dragging-movements.html

S06 / W3C APG, Dialog (Modal) pattern
Initial focus, Tab containment, Escape, focus return.
https://www.w3.org/WAI/ARIA/apg/patterns/dialog-modal/

S07 / W3C APG, Menu Button pattern
Enter and Space open the menu on the first item.
https://www.w3.org/WAI/ARIA/apg/patterns/menu-button/

S08 / W3C APG, Menu and Menubar pattern
Arrow keys, Escape returns focus, Tab closes.
https://www.w3.org/WAI/ARIA/apg/patterns/menubar/

S09 / W3C APG, Tabs pattern
Arrow keys wrap; one tab stop; automatic or manual activation.
https://www.w3.org/WAI/ARIA/apg/patterns/tabs/

S10 / W3C APG, Combobox pattern
DOM focus stays on the combobox; aria-activedescendant; Enter and Escape.
https://www.w3.org/WAI/ARIA/apg/patterns/combobox/

S11 / W3C APG, Listbox pattern
Focus on entry, arrow keys, optional Home, End and type-ahead.
https://www.w3.org/WAI/ARIA/apg/patterns/listbox/

S12 / W3C APG, Grid pattern
One tab stop, arrows, Home, End, Control+Home and Control+End.
https://www.w3.org/WAI/ARIA/apg/patterns/grid/

S13 / Playwright, Keyboard
keyboard.press key names and modifier combinations.
https://playwright.dev/docs/api/class-keyboard

S14 / W3C, Understanding SC 2.4.7 Focus Visible
AA; a visible indicator in keyboard use.
https://www.w3.org/WAI/WCAG22/Understanding/focus-visible.html

Maintenance: recheck the APG patterns once a year (the scripts assert only required keys) and re-run `npm run check` after each Playwright upgrade. Update the PDF, HTML, Markdown and JSON together.