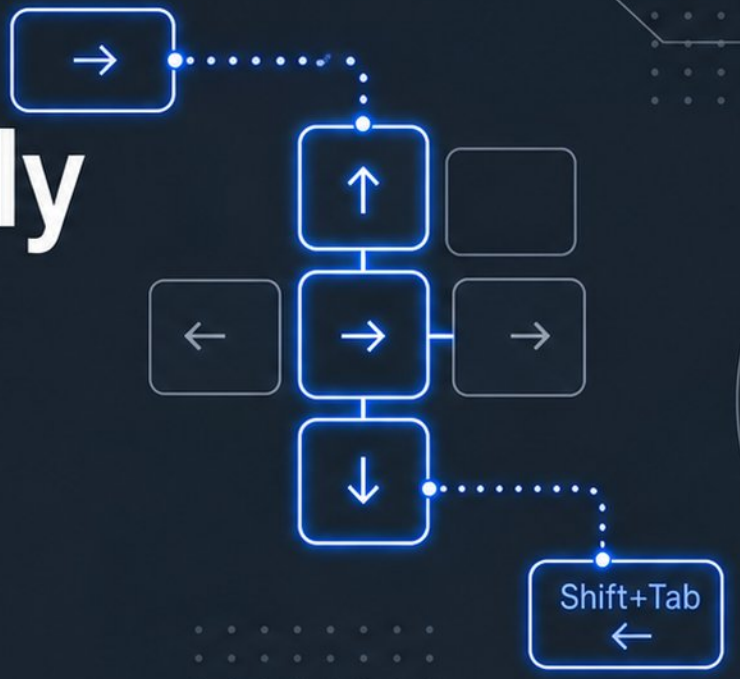
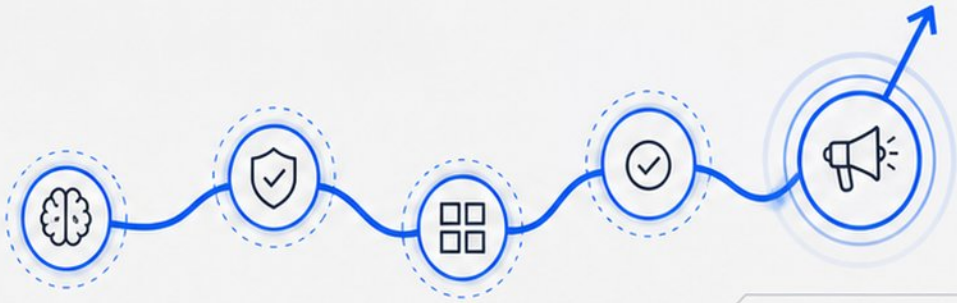
Designing Keyboard-Friendly Interfaces



Keyboard accessibility is a core design responsibility, not an engineering afterthought

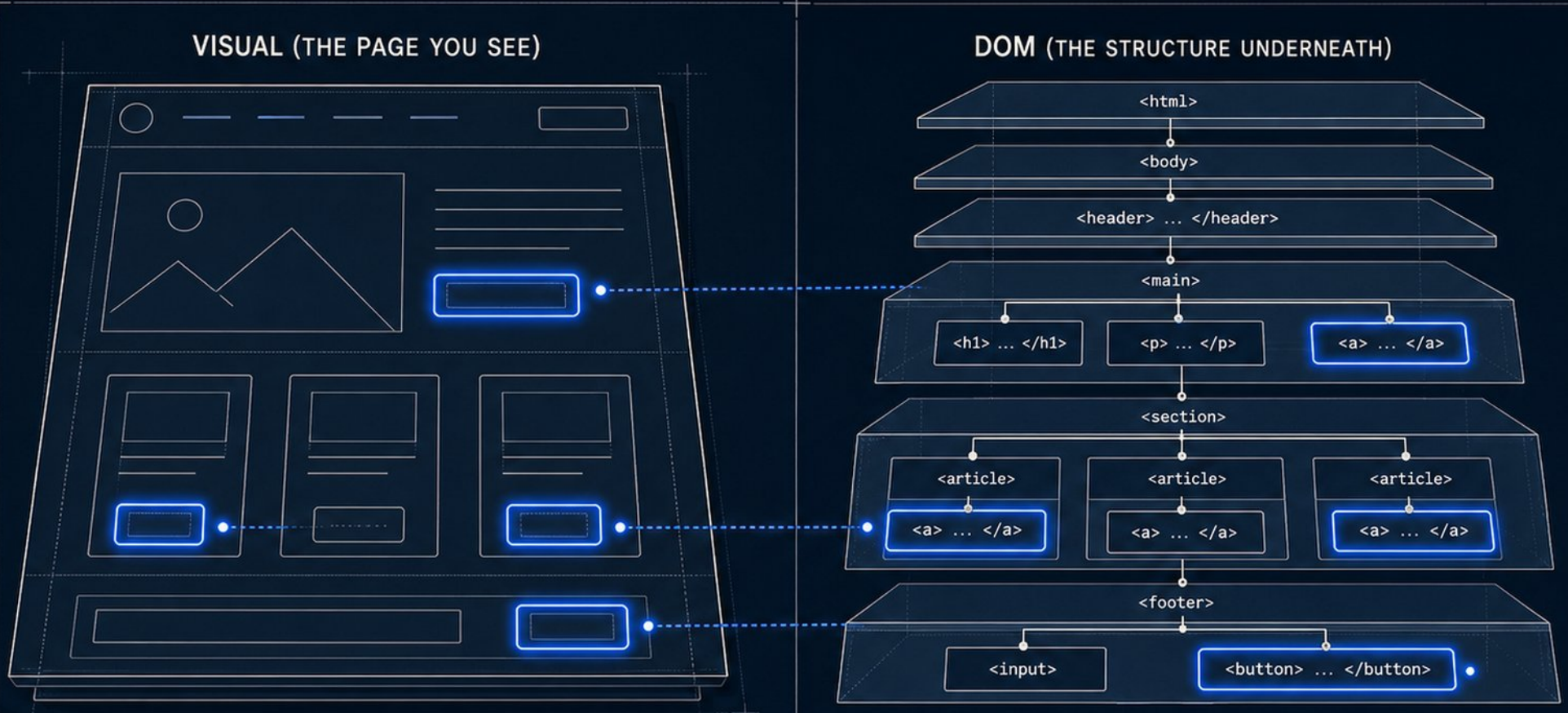
- Course path: mental models → standards → design patterns → testing → advocacy
- Learn to diagnose focus issues and design efficient keyboard flows
- Build skills to advocate for keyboard UX across your team



User Lenses and Mental Models

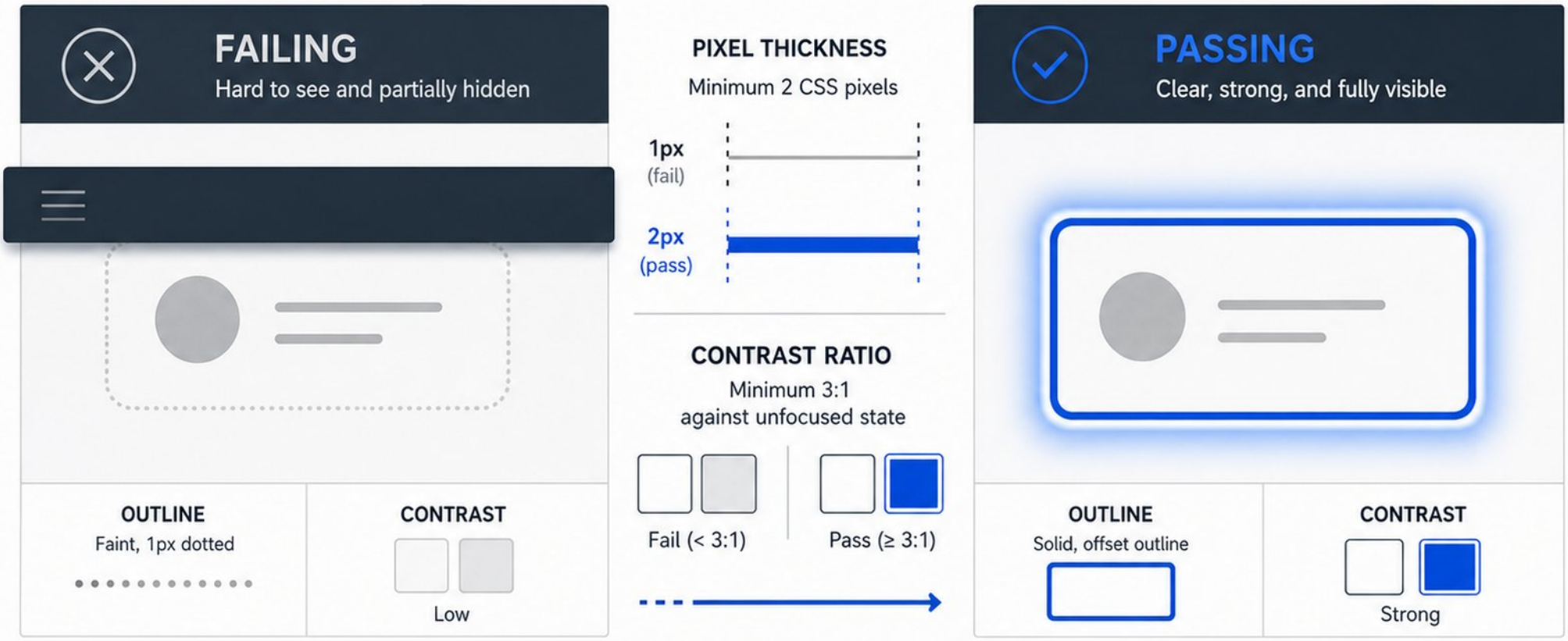
- Keyboard reliance spans permanent disabilities, temporary injuries, situational needs, and power users
- Interaction models go beyond Tab: screen readers, switch devices, voice fallback, custom shortcuts
- Broken keyboard UX drives task abandonment, legal risk, brand erosion, and lost revenue

Focus, Tab Order, and the DOM



- Focus is the keyboard cursor; every interactive element needs a visible indicator
- DOM order defines default tab sequence; adjust with `tabindex="0"` or `-1` only
- Never use positive `tabindex` values — they break the visual-to-keyboard flow
- Trap focus inside modals and dialogs using `inert` and cyclic containment

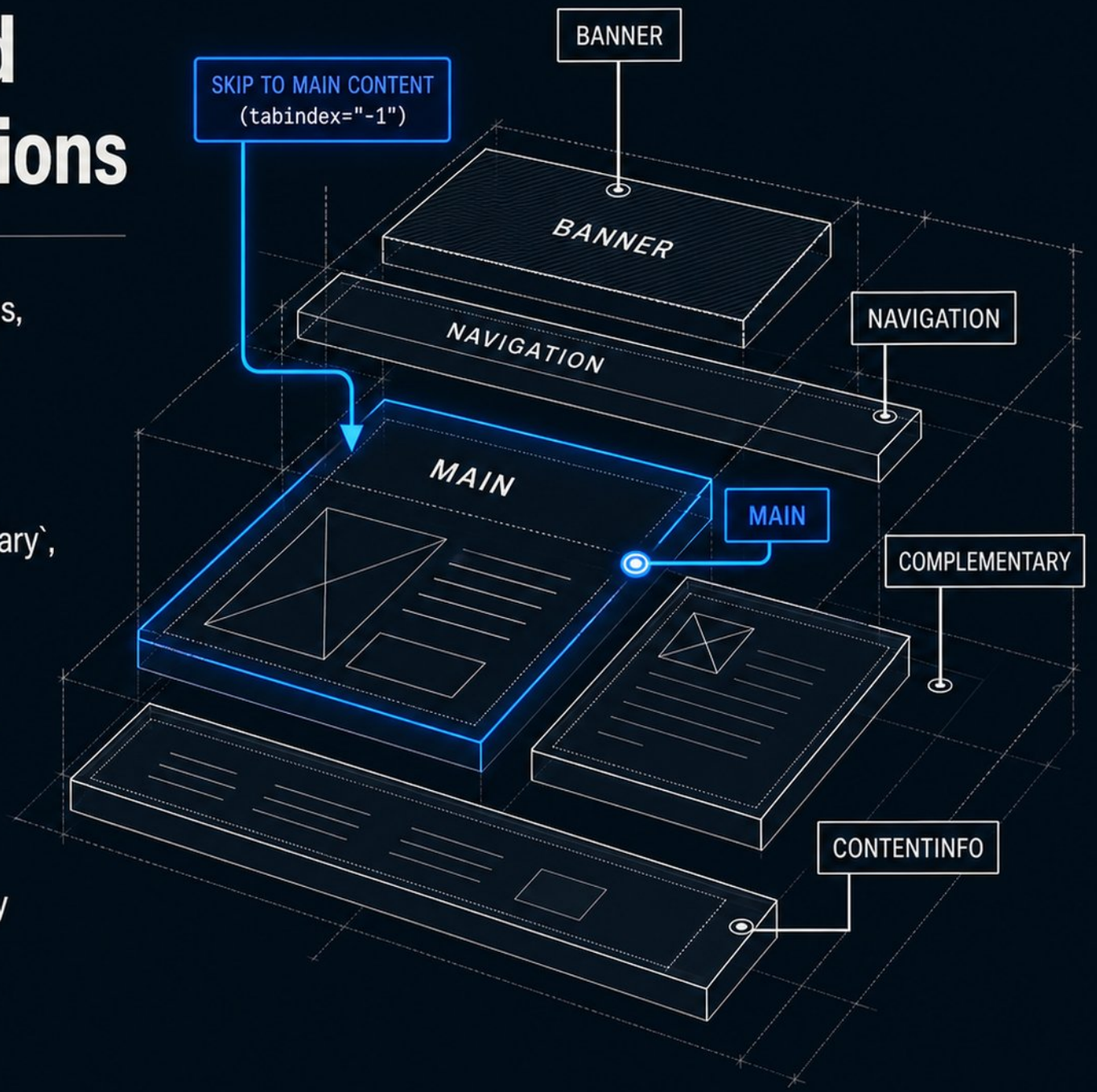
Visible Focus Indicators and WCAG 2.2 Requirements



- Focus indicator must be at least 2 CSS pixels thick
- 3:1 minimum contrast ratio against unfocused state
- Focus must not be fully obscured by sticky elements
- Use solid outlines and `outline-offset` patterns
- Avoid subtle changes like 1px dots or color-only shifts

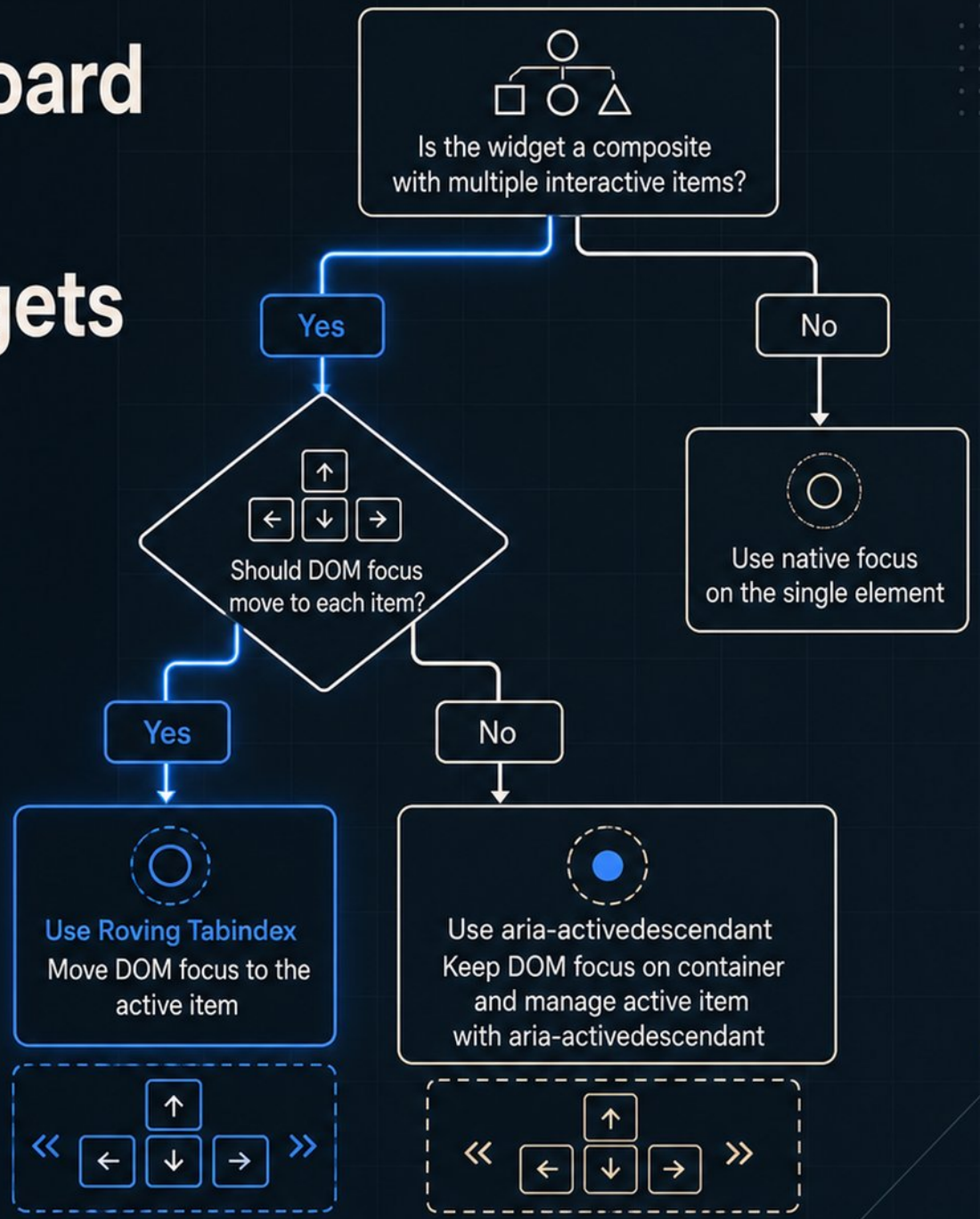
Skip Links and Landmark Regions

- Design skip links: visible on focus, positioned early, targeting main content with `tabindex="-1"`
- Use HTML landmarks: `banner`, `navigation`, `main`, `complementary`, `contentinfo` for screen reader quick-nav
- Landmarks replace hard-to-find bypass blocks — a single `main` is expected on every page
- Reinforce with heading hierarchy and ARIA regions, especially in single-page applications



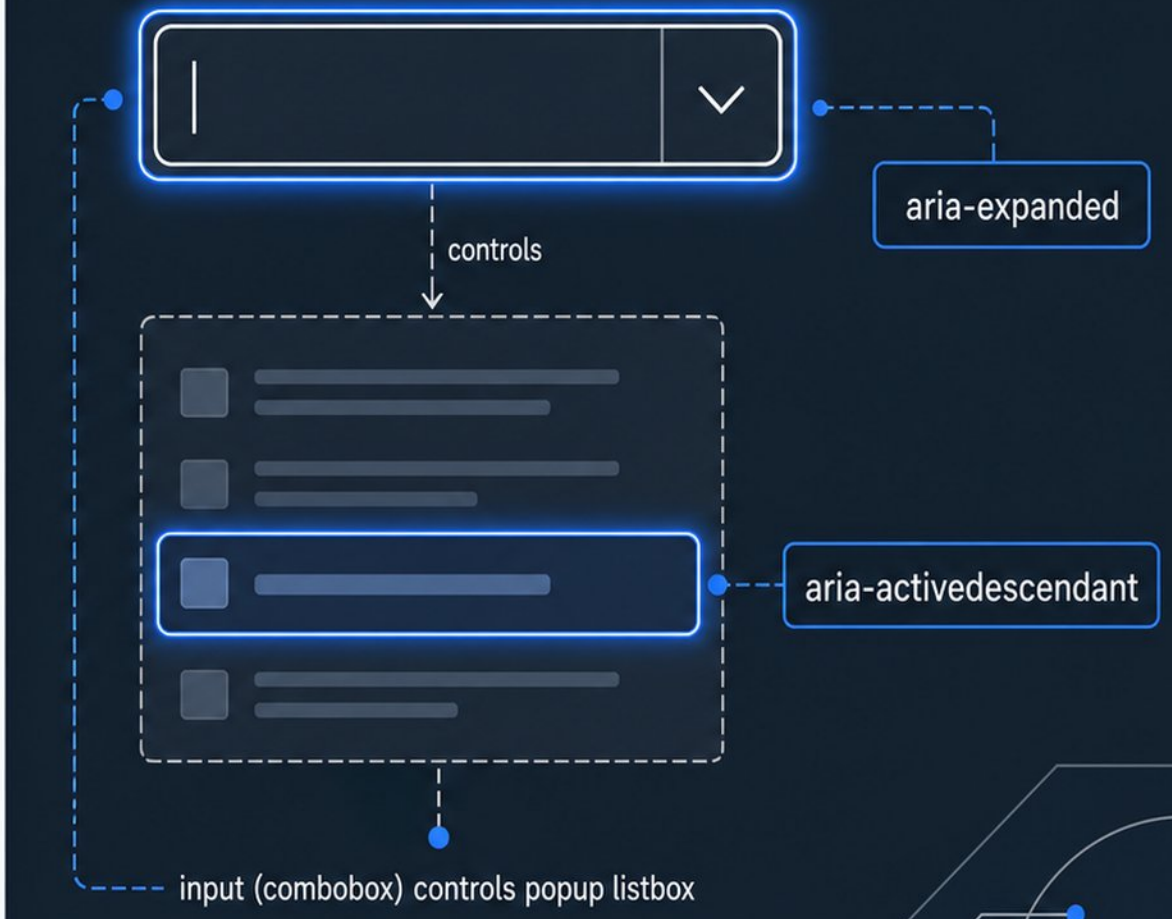
Designing Keyboard Interaction for Composite Widgets

- Follow WAI-ARIA patterns for menus, tabs, accordions, and toolbars
- Manage focus with roving tabindex or aria-activedescendant strategies
- Design shortcut keys that avoid AT, browser, and OS conflicts
- Ensure custom scrollable regions are fully keyboard-operable

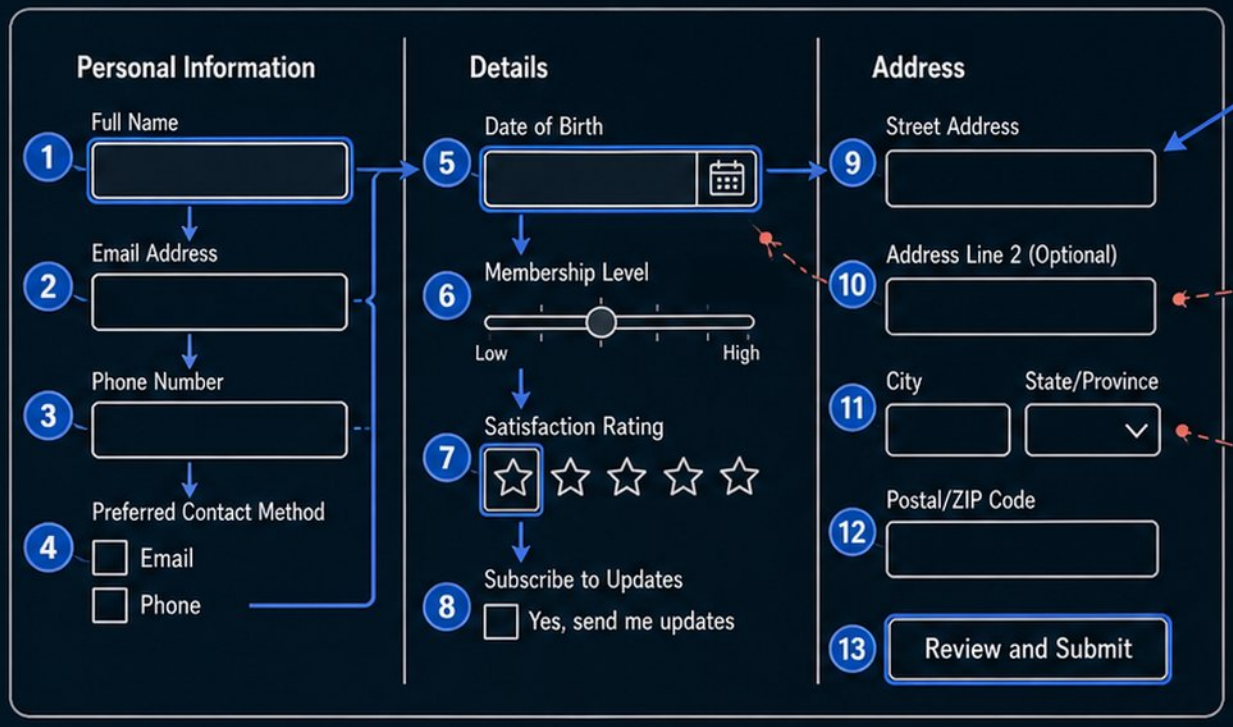


Accessible Custom Select, Combobox, and Autocomplete Patterns

- Distinguish listbox, combobox, and menu by ARIA pattern and keyboard model
- Combobox: `role="combobox", aria-activedescendant`, aria-expanded`, aria-autocomplete`
- Keep DOM focus on the input; track highlighted option with aria-activedescendant`
- Keyboard checklist: Arrow keys, Enter, Escape, Home, End, typeahead



Forms, Validation, and Error Recovery



There are 2 errors on this form

- Email Address: Enter a valid email address.
- Date of Birth: Please enter your date of birth.

Press F6 to move focus to the first error.

Enter a valid email address. (id="email-error")

aria-live="assertive" Error: Enter a valid email address.

Please enter your date of birth. (id="dob-error")

aria-live="assertive" Error: Please enter your date of birth.

Your session will expire soon

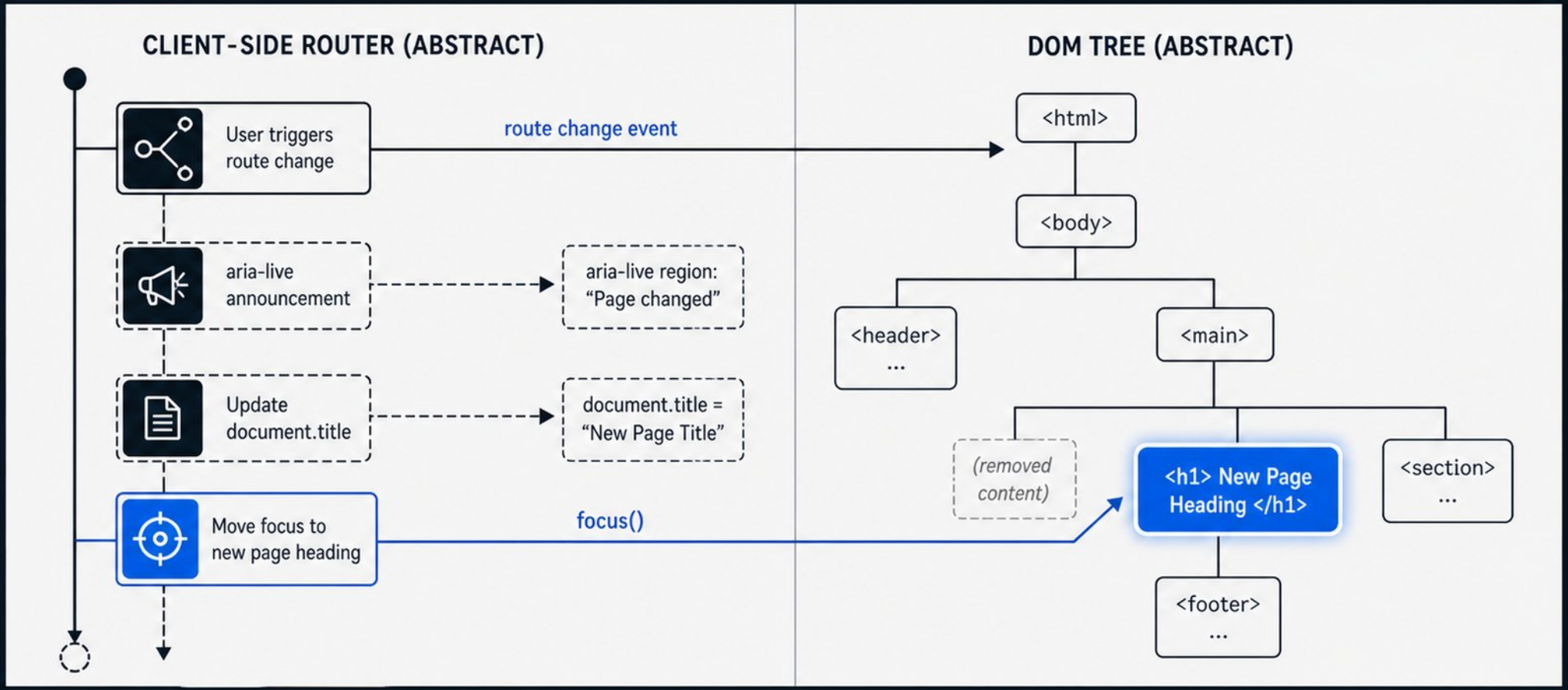
You will be signed out in 2 minutes due to inactivity.

Press Tab to stay signed in. Tab

This is an alert. (role="alert")

1. Design logical tab order across multi-column and complex layouts
2. Ensure custom controls are keyboard-operable: date pickers, sliders, ratings
3. Announce errors in real time using `aria-live` regions and focus movement
4. Link error messages to fields with `aria-describedby` for context
5. Handle session timeouts with keyboard-accessible `role="alert"` warnings

SPA Routing, Client-Side Focus, and Dynamic Content



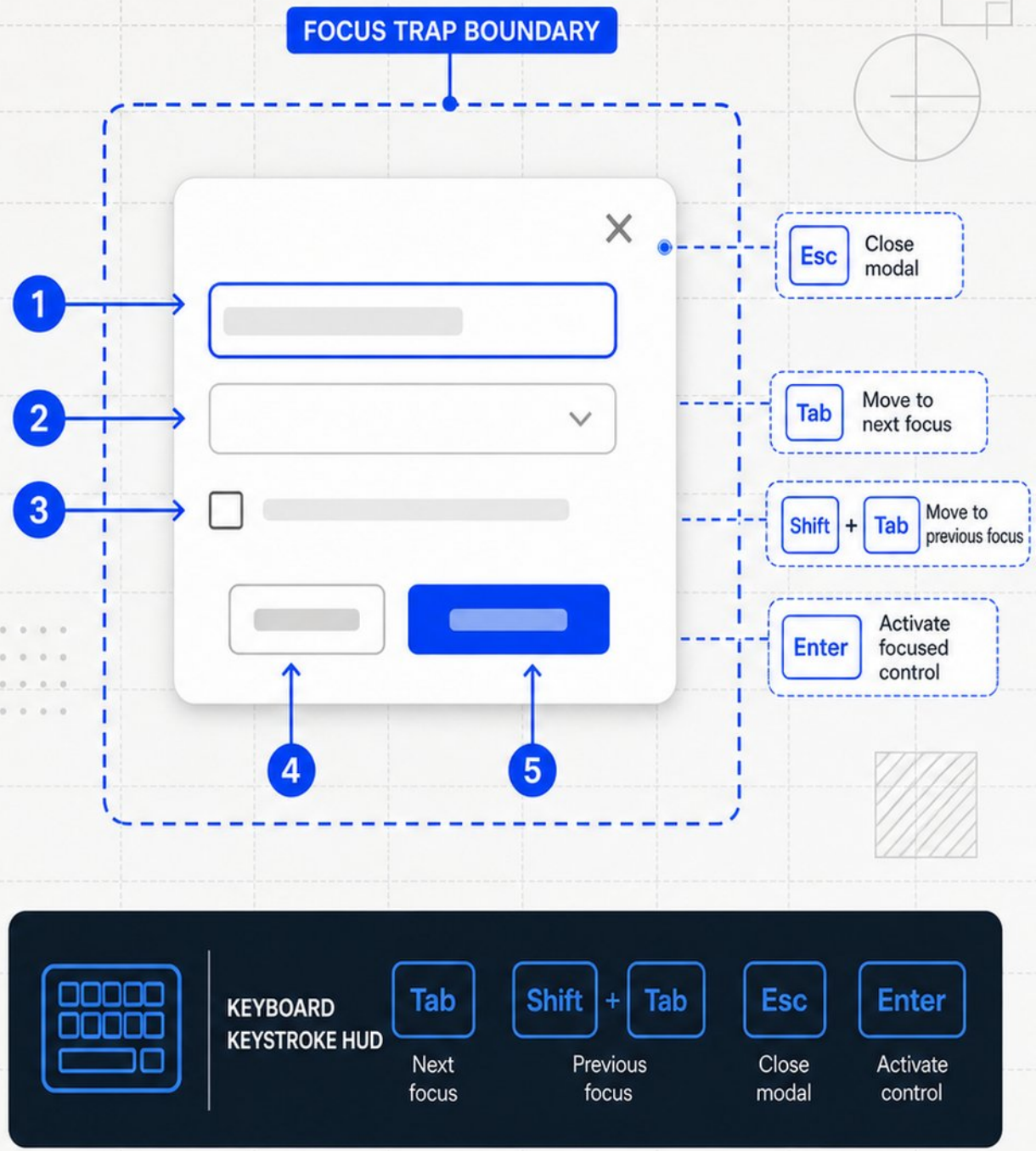
- - Client-side routing strands keyboard focus on removed content
- - Move focus to the new page heading after each route change
- - Announce route changes via `aria-live` and update `document.title`
- - Maintain meaningful tab order in infinite scroll and virtual lists
- - Apply focus patterns consistently across React, Vue, Angular, and Svelte

Motion, Animation, and Keyboard-Driven Experiences

- Respect `prefers-reduced-motion` while keeping motion clear for keyboard users
- Meet WCAG 2.2: Motion Actuation, Animation from Interactions, and Pause Stop Hide
- Provide manual pause, stop, and keyboard-alternative controls for media and parallax
- Avoid keyboard-triggered animation that disrupts focus management or screen readers

Design Annotations and Developer Handoff

- Annotate focus order, keyboard specs, and trap boundaries directly in design files
- Use tab order diagrams, shortcut maps, and dynamic focus move tables
- Add keyboard UX reviews to design critiques and QA checklists
- Build collaborative handoff workflows between designers, developers, and accessibility specialists



Testing, Advocacy, and Building a Keyboard-First Culture

- Quick manual audits: test Tab, Enter, Escape, arrows, and focus visibility
- Verify with VoiceOver, NVDA, and JAWS using keyboard-driven checklists
- Link keyboard UX wins to conversion, retention, compliance, and ROI
- Champion one keyboard-priority improvement in your next sprint

PersonWise.

PersonWise • AI digital-human course platform

Scan the code or click anywhere to watch this course live, with voice Q&A

personwise.ai/t/7067d00a/public