COAX

Form UX audit Standard Operating Procedure

Checklist

Form UX audit standard operating procedure (SOP)

The COAX team put together an end-to-end SOP for developers to conduct a complete form UX audit after designers' handoff. The following framework serves as a repeated process that can serve as a simple handoff checklist from the designer to the developer to achieve both project consistency and traceability from design specifications to automated verification.

Goal: Ensure **100%** of forms meet WCAG 2.1 AA compliance and achieve a completion rate **>80%** before production deployment.

Standard operating procedure

STAGE 1:
AUDIT PREPARATION
(Day 1, 0-2 hours)

Who:

UX designer

Actions:

- [ ] Identify all forms in scope (registration, checkout, contact, profile, etc.)
- [ ] Document current completion rates and abandonment points from analytics
- [ ] Set up testing environment with screen readers (NVDA, JAWS, VoiceOver)
- [ ] Create an audit tracking spreadsheet with all forms listed
- [ ] Schedule a 30-minute kickoff with the designer, developer, and PM

SLA target:

Audit scope defined and documented within **2 hours**

STAGE 2:
HEURISTIC EVALUATION
(Day 1-2, 2-8 hours)

Who:

UX Designer

Actions:

Visual design & layout

- [ ] Verify all form fields have visible labels positioned above or to the left
- [ ] Check for adequate spacing between fields (minimum 8px vertical)
- [ ] Verify button hierarchy is clear (primary vs. secondary actions)
- [ ] Ensure focus indicators meet 3:1 contrast ratio minimum

Error handling & validation

- [ ] Confirm error messages appear inline, near the problematic field
- [ ] Verify errors are specific (not "Invalid input" but "Email must include @")
- [ ] Check that validation triggers on blur, not only on submit
- [ ] Ensure error messages persist until the issue is resolved
- [ ] Verify the error summary appears at the top of the form for multiple errors

Microcopy & content

- ☐ Verify label text is clear and concise (no jargon)
- ☐ Check that button text is action-oriented ("Create Account" not "Submit")
- ☐ Ensure optional fields are marked "(optional)" not just required fields with *
- ☐ Verify help text uses plain language and active voice

SLA target:

Heuristic evaluation completed within 8 hours per form

STAGE 3:
KEYBOARD & SCREEN READER TESTING
(Day 2-3, 4-6 hours)

Who:

Accessibility Specialist + Developer

Actions:

Keyboard navigation (manual testing)

- ☐ Navigate entire form using only Tab/Shift+Tab (no mouse)
- ☐ Verify tab order follows visual layout (top to bottom, left to right)
- ☐ Confirm all interactive elements are reachable
- ☐ Test that Enter submits the form only when the submit button is focused
- ☐ Verify no keyboard traps exist in custom components
- ☐ Check that Escape closes any modals and returns focus appropriately

Focus management

- ☐ Verify visible focus indicator on all interactive elements
- ☐ Check that focus doesn't disappear or jump unexpectedly
- ☐ Confirm focus moves to the first error after a failed submission
- ☐ Test that focus returns to trigger after closing modal forms

Screen reader testing

- ☐ Test with NVDA (Windows) or VoiceOver (Mac) on the entire form
- ☐ Verify all labels are announced correctly
- ☐ Confirm error messages are announced when they appear
- ☐ Check that hint text is associated via aria-describedby
- ☐ Verify required fields are announced as required
- ☐ Test that form submission success/failure is announced

SLA target:

Keyboard and screen reader testing completed within **6 hours** per form

STAGE 4:
AUTOMATED TESTING
(Day 3, 1-2 hours)

Who:

Developer

Actions:

- ☐ Run axe DevTools on all form pages
- ☐ Run WAVE browser extension on all form pages
- ☐ Execute automated accessibility tests in CI/CD pipeline
- ☐ Verify HTML validation passes (W3C Validator)
- ☐ Check color contrast with automated tools (minimum 4.5:1 for text)

SLA target:

Automated tests completed and documented within **2 hours** per form

STAGE 5:
CODE REVIEW
(Day 3-4, 2-4 hours)

Who:

Developer + Accessibility Specialist

Actions:

Semantic HTML review

- [ ] Verify <button> elements used (not <div> with onClick)
- [ ] Confirm all inputs have corresponding <label> elements
- [ ] Check that <form> element wraps all related fields
- [ ] Verify proper input types (email, tel, number, date, etc.)
- [ ] Ensure fieldsets with legends are used for grouped inputs

Focus management code

- [ ] Review focus trap implementation in modal forms
- [ ] Check for proper focus restoration after modal close
- [ ] Verify programmatic focus movement to the first error
- [ ] Ensure no tabindex values greater than 0

ARIA implementation

- ☐ Verify required attribute and aria-required="true" on required fields
- ☐ Check aria-invalid="true" on fields with errors
- ☐ Confirm aria-describedby links hint to fields
- ☐ Verify aria-live regions for dynamic error announcements
- ☐ Check role="alert" for critical error summaries
- ☐ Ensure aria-expanded is set on combo boxes/dropdowns

SLA goal:

Code review completed within **4 hours** per form

GUIDELINES FOR TESTING PRIORITY

Priority 0 (P0) Critical

User cannot complete the task due to issues being reported - therefore, the conversion cannot happen.

Priority 1 (P1) High

Issues create a negative experience that significantly decreases the chances for conversion.

Priority 2 (P2) Medium

Issues generate user frustration and create a minimal effect on numerous metrics.

Priority 3 (P3) Low

Minimal inconvenience, not affecting many users at this time

DOCUMENTATION AND TASK MANAGEMENT REQUIREMENTS

Documentation format: The 5 C's

Developers need to document every keyboard accessibility issue using the 5 C's format. This framework provides clarity and enables manual fixes and reasonable automated prevention.

Criteria

The specific standard that was violated should be stated explicitly. This can include reference to WCAG success criteria, patterned design patterns, or documented keyboard interaction models.

Example: WCAG 2.4.3 Focus Order - components receive focus in an order that maintains meaningfulness and operability.

Root cause

The modal component was implemented without a focus management approach. The component did render an overlay, but did not prevent tab navigation from reaching elements outside of the modal boundary.

Technical details

Focus trap logic not implemented; no keydown event listener for the Tab key; no inert attribute on the background content.

Consequence

Define the actual user impact. Link the technical violations to human experience.

Example: Keyboard-only users will lose context when navigating through the checkout modal, as focus moves to the underlying hidden background content area. The user would not be able to complete the purchase and checkout flow with a mouse.

Severity

of the detected accessibility issue.

Example: Critical - keyboard user cannot complete a task

Corrective action

Reiterate specific, actionable steps.

Example: Use the focus-trap-react (or an equivalent) library to enforce focus trap in the modal component.

Every Jira ticket must contain the following items:

- ☐ A specific WCAG compliance criterion and associated criteria
- ☐ A manual keyboard navigation test was completed
- ☐ An automated testing result showing a passing result in CI/CD
- ☐ Confirmation that screen readings announce the content being tested correctly
- ☐ Confirmation that existing functionality was unaffected during testing and verification