Introduction to ARIA: A Developer’s Guide

Anisat Ahmed

Conf42.com JavaScript 2025

Where We Are Going Today

- **The Invisible Problem**
- **ARIA Fundamentals**—What it is and how it works
- **The Three Core Concepts**—Roles, Properties & States
- **Best Practices**—When to use (and not use) ARIA
- **Common Mistakes**—And how to avoid them
- **Testing & Tools**—Making accessibility a habit
- **Action Plan**—What to do on Monday

The Invisible Problem


<!-- What we see: A beautiful custom button -->

Click Me

What a screen reader announces:

"Clickable. Click Me."

This could translate to "There's something you can click here called 'Click Me'... but I have no idea what it actually does or if it's even a button."

HTML speaks two languages:

Visual Language:

What users see

- Colors, layouts, animations, hover effects

Semantic Language:

What assistive technologies interpret

- "This is a button"
- "This is a navigation menu"
- "This is heading level one"

ARIA aligns the two.

It gives structure, meaning, and state to custom UI elements.

What is ARIA?

Accessible Rich Internet Applications (ARIA)

A specification of attributes that:

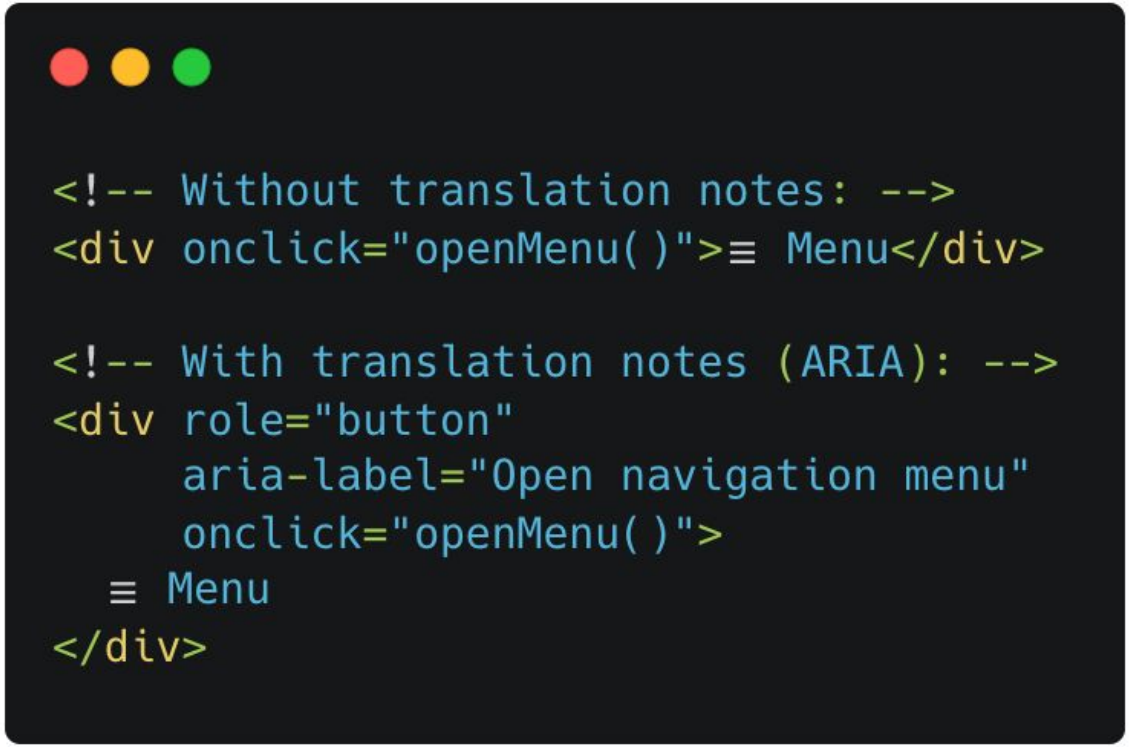
- Describe roles (role="dialog")
- Announce names and descriptions (aria-label, aria-labelledby)
- Communicate states (aria-expanded, aria-pressed)

ARIA doesn’t create functionality; it makes existing functionality understandable.

Think of ARIA as **translation notes for screen readers.**

That's it. ARIA attributes are just notes that say:

- "Hey, this is actually a button"
- "This is what it does"
- "This is its current state"


<!-- Without translation notes: -->
≡ Menu
<!-- With translation notes (ARIA): -->
<div role="button"
aria-label="Open navigation menu"
onclick="openMenu()">
≡ Menu

The Three Core Concepts

ARIA Core Concepts

ROLES

Define what an element is.

For example, it defines the code below as a button, even though it's a div.

PROPERTIES

Define its characteristics.

For example, it defines the characteristics of the button as 'Close dialog' and not just 'X'

STATES

Define what's happening right now.

For example, it tells if the button is currently pressed/active.

The Golden Rule:

"No ARIA is better than bad ARIA."

- Use native HTML first.
- Reach for ARIA only when there's no semantic element available.
- Don't re-implement built-in behavior (<button>, <input>, <a>).

When You Actually Need ARIA

Use ARIA for components that HTML can’t describe alone:

- Custom widgets (tabs, carousels, tree views)
- Dynamic updates (notifications, live status)
- Complex single-page interfaces

Avoid ARIA for standard controls or uncertain cases; test first.

A Few Code Examples

Example 1 — Live Notifications

```
<!-- Without ARIA -->

<div id="notice">Message sent successfully!</div>
```

Screen reader: No announcement.

```
<!-- With ARIA -->

<div id="notification"
     role="status"
     aria-live="polite">
  Message sent successfully!
</div>
```

Screen reader: “Message sent successfully.”

Example 2 — Modal Dialogs

The Problem Most Modal:

When a modal opens, screen reader users often don't realize:

- That a modal opened
- What the modal is for
- How to close it

```
<div role="dialog"
     aria-labelledby="dialog-title"
     aria-describedby="dialog-desc"
     aria-modal="true">
  <h2 id="dialog-title">Confirm Delete</h2>
  <p id="dialog-desc">Are you sure you want to delete this item?</p>
  <button>Cancel</button>
  <button>Delete</button>
</div>
```

Screen reader: “Confirm Delete, dialog. Are you sure you want to delete this item?” Now the purpose and context are clear.

Common Mistakes

- Adding ARIA to everything
- Hiding interactive elements (using `aria-hidden="true"`)
- Forgetting keyboard support
  - ❏ ARIA doesn't add functionality! You still need click handlers AND keyboard handlers.
- Conflicting roles or invalid states

The Fix → Use HTML. Test with keyboard. Test with screen reader.

Tools & Testing

Keyboard Test

- Navigate without a mouse.
- Ensure focus is visible and logical.

Screen Reader Test

- NVDA (Windows) · VoiceOver (Mac).
- Check labels and announcements.

Automated Test

- axe DevTools · WAVE · Lighthouse.
- Fix issues, then retest manually.

Quick Reference - ARIA Attributes You'll Use Most

Roles:

- role="button" - It's a button
- role="navigation" - It's a nav menu
- role="dialog" - It's a modal
- role="alert" - Urgent message
- role="status" - Status update

Labels:

- aria-label="text" - Name this element
- aria-labelledby="id" - This element names it
- aria-describedby="id" - Extra description

States:

- aria-expanded="true/false" - Is it open?
- aria-selected="true/false" - Is it selected?
- aria-hidden="true/false" - Should screen readers ignore it?

Quick Reference (contd.)

Live Regions:

- `aria-live="polite"` - Announce when idle
- `aria-live="assertive"` - Announce immediately

Your Action Plan

- **Learn**—Read WAI-ARIA Authoring Practices
- **Audit**—Test keyboard flow · Run axe DevTools.
- **Fix—**Start with high-impact issues.
- **Sustain—**Add accessibility to every PR and review.

Small steps → huge impact.

Key Takeaway:

Here's what I want you to remember:

- ❏ It's about respecting that people use the web in different ways.
- ❏ It's about ensuring your work is usable by everyone.
- ❏ It's about building for the real, diverse web, not just the web as we personally experience it.

Every ARIA attribute you write is an act of inclusion.

Resources

- WAI-ARIA Authoring Practices
- MDN ARIA Documentation
- WebAIM Article on ARIA
- Axe Dev Tools
- NVDA
- WAVE

Q & A