Engineer

ARIA

ekstra markup der navngiver en kontrol for hjælpeteknologi

Hvad det er

ARIA er et sæt attributter fra W3C, som fortæller hjælpeteknologi, hvad et element på en webside er, hvad det hedder, og hvilken tilstand det er i, der hvor HTML alene siger for lidt. Forkortelsen står for Accessible Rich Internet Applications. Specifikationens fulde navn er WAI-ARIA, efter W3C's Web Accessibility Initiative, og den gældende anbefaling er version 1.2.

Attributten `role` angiver, hvilken slags ting elementet er: en widget som `button` eller `tab`, en strukturdel som `list`, en af de otte landmærkeroller som `navigation` eller `main`, eller en live-region som `status`, hvis ændringer skærmlæseren læser op af sig selv. De øvrige `aria-*` -attributter er enten egenskaber, for eksempel `aria-labelledby`, eller tilstande, for eksempel `aria-expanded`, som skifter undervejs, mens brugeren arbejder. Browseren oversætter det hele til styresystemets tilgængeligheds-API, og det er dér, en skærmlæser eller et program til stemmestyring henter sine oplysninger.

ARIA ændrer kun, hvad der bliver sagt. En `div` med `role="button"` bliver annonceret som knap, men Tab springer den over, og Enter og mellemrum gør intet, før en `tabindex` og et script har sørget for begge dele. Derfor lyder den første regel for brug af ARIA i W3C's vejledning Using ARIA sådan: Findes der et indbygget HTML-element eller en attribut med den semantik og den adfærd, du har brug for, så brug det frem for at give et andet element ny betydning og lægge ARIA på.

```
<!-- annonceres som knap, opfører sig som en div -->
<div role="button">Send</div>

<!-- annonceres, får fokus og betjenes som knap -->
<button type="button">Send</button>
```

Den anden halvdel er det tilgængelige navn, altså de ord, skærmlæseren siger om en kontrol. Accessible Name and Description Computation 1.2 fastlægger rækkefølgen: først `aria-labelledby`, så `aria-label`, så værtssprogets egen etiket (et `label`-element, en `alt`-tekst, en `title` i SVG), dernæst elementets tekstindhold for de roller, der henter navnet derfra, og til sidst attributten `title`. Et `aria-label` på en knap erstatter altså de synlige ord i stedet for at supplere dem.

Brug begrebet i en beslutning

Brug »Hvad er ARIA?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Brugerflader, der indlæses hurtigt, virker på en telefon og opfører sig ens i alle de browsere, jeres kunder nu bruger.

Se vores ydelser Kontakt os →

Undersøg antagelserne bag ordet

Brug forklaringen og det tilknyttede arbejde til at formulere spørgsmål til jeres projekt. Vi kan undersøge antagelserne sammen, før et værktøj vælges. Har styret full-stack-udviklingen af GIS-kort — full stack-kortløsning, fra database til browser

Har styret full-stack-udviklingen af GIS-kort Kontakt os →

Hvorfor det betyder noget

Det valg, ARIA stiller et team over for, er at bygge selv eller bruge det, der findes. Succeskriterium 4.1.2 i WCAG 2.2, Name, Role, Value, kræver, at hver komponent i brugerfladen giver software adgang til sit navn og sin rolle, lader software sætte de tilstande og værdier, brugeren selv kan ændre, og giver besked, når de skifter. En indbygget `button`, `select` eller `details` opfylder det fra starten. En hjemmelavet fanebjælke, kombinationsboks eller træstruktur opfylder det kun gennem ARIA og det script, der holder attributterne i takt med virkeligheden.

En hjemmelavet widget er derfor en løbende udgift: tastaturhåndtering, fokusstyring, tilstande, der følger hver handling, og test i mere end én skærmlæser ved hvert redesign. Vurder et komponentbibliotek på, hvilket mønster fra W3C's ARIA Authoring Practices Guide hver widget følger, og på, om mønstrets tastaturadfærd holder, når musen er lagt væk.

Det juridiske pres rammer nu også private virksomheder. Webtilgængelighedsdirektivet, (EU) 2016/2102, dækker offentlige organers websteder og apps. Tilgængelighedsdirektivet, direktiv (EU) 2019/882, som de fleste kender som European Accessibility Act, gælder fra 28. juni 2025 for en række forbrugerprodukter og -tjenester, blandt andet e-handel og bankydelser til forbrugere, mens mikrovirksomheder, der leverer tjenester, er undtaget. Begge læner sig op ad den harmoniserede standard EN 301 549, hvis krav til web

er WCAG på niveau AA. Name, Role, Value ligger på niveau A, så en kontrol, hvis navn eller rolle aldrig når frem til skærmlæseren, dumper på det laveste niveau, en revision tjekker.

Sådan ser det ud i praksis

Skabelonerne bag dette site bruger ARIA sparsomt. Topbjælken på alle undersider er en `div` med `role="banner"` frem for det sædvanlige `header`-element. Kommentaren i skabelonen giver grunden: Stilarket gør et bart `header` til forsidens hero i fuld højde, så elementet ville slæbe den geometri med ind på hver underside, mens rollen giver landmærket og intet andet. Det er rollen, der placerer bomærket, linket til forsiden, inden for et landmærke. Netop det snævre tilfælde giver den første regel plads til. Menuen går den modsatte vej: Den er et par af `details` og `summary` og overlader åben eller lukket, knapsemantikken og `aria-expanded` til browseren. Menuens `nav` og sidefodens `nav` har hver sin `aria-label`, fordi to navigationslandmærker med samme navn bryder regelmotorens krav om entydighed, og sitets tilgængelighedstjek kræver nul fund. Den bevarede implementering af porteføljefilteret bruger en region med `role="status"`, som er beregnet til at forholde sig tavs, indtil læseren skriver. Filteret er for tiden deaktiveret med `searchEnabled = false`; det viser den bevarede implementering, ikke en interaktion, der er tilgængelig på de offentliggjorte sider.

Gennemgangen af, hvordan sitet blev løftet til WCAG 2.2 niveau AA, leverer den anden halvdel af historien om topbjælken: Da regelmotorens best practice-regler blev slået til, gav en af dem fejl med det samme, fordi bomærket på hver eneste underside stod helt uden for landmærkestrukturen. Samme side regner manglende test med rigtig hjælpeteknologi blandt de begrænsninger, den selv offentliggør.

Beretningen om elleve fejl, som browseren kaldte sunde, viser, hvor lidt opmærkningen kan stå inde for. Alle elleve opstod i den måde, siden blev tegnet på skærmen, og ikke i det, opmærkningen erklærede, og opmærkningen er det eneste, ARIA rører ved. Otte links i sidefoden kunne stadig nås med Tab, efter at en stilregel havde slået interaktion med pegeredskab fra og ellers intet ændret: samme mønster som `aria-hidden` på et element, der kan få fokus, hvor skærmlæseren mister elementet, mens tastaturet beholder det. Desuden var kun 70 af motorens 105 regler slået til, og den regel, som skulle fange det manglende landmærke, var blandt de fravalgte.

Den sidste side viser samme problem på en platform, hvor ARIA ikke indgår. Alle sytten ikoner i brugerfladen fik navne til skærmlæsere handler om et SwiftUI-program uden HTML og uden en eneste `aria-` -attribut. Tilgængelighedsetiketten dér er SwiftUI's egen, den indbyggede pendant til `aria-label` og ikke en udgave af den. Knapper tegnet med systemsymboler blev læst op med symbolets interne navn, så send-knappen blev annonceret som »arrow up circle, button«, og sytten rene ikonkontroller beskrev deres tegning i stedet for deres funktion. Hver fik en etiket, der nævner handlingen og følger tilstanden, hvor det betyder noget, og en linter afviser nu en ikonkontrol, der mangler en etiket inden for otte linjer fra det sted, hvor den oprettes, så en attende kontrol kun kommer ind med etiket. Siden siger selv, hvor grænsen går, og den gælder også for `aria-label` : Tjekket bekræfter, at en etiket står tæt på, mens kun et menneske, der lytter, kan afgøre, om det er den rigtige.

Hvor det går galt

En rolle lover en widget, som tastaturet aldrig når. En `div` får `role="button"` og en klikhandler. Skærmlæseren siger »knap«, men Tab hopper forbi, og Enter gør intet. Den tredje regel for brug af ARIA kræver, at alle interaktive ARIA-kontroller kan betjenes med tastaturet. Brug det indbyggede element, eller byg hele tastaturmodellen fra APG-mønstret, og test med musen trukket ud.

Fokus lander på noget, skærmlæseren regner for fraværende. `aria-hidden="true"` sidder på et element, der kan få fokus, eller på et af dets overordnede elementer, for eksempel en skuffemenu uden for skærmen. Tab flytter stadig derhen, og brugeren hører stilhed. Den fjerde regel for brug af ARIA udelukker netop det. Tag skjult indhold ud af tab-rækkefølgen med `hidden` , `display: none` eller `inert` , og gem `aria-hidden` til pynt, der aldrig får fokus.

Et navn havner, hvor elementet ikke tager imod et. Et `aria-label` på en `div` , `span` eller `p` er ment som en forklaring, men WAI-ARIA 1.2 forbyder at navngive rollerne `generic` og `paragraph` , og skærmlæsere ignorerer navnet eller læser det uensartet op. På en kontrol erstatter etiketten de synlige ord, og når de to er forskellige, rammer en bruger af stemmestyring ved siden af ved at sige de ord, der står på skærmen. Giv kun navne til elementer, der kan bære et, og lad ethvert `aria-label` begynde med de synlige ord, som WCAG 2.5.3, Label in Name, beder om.

To roller modsiger hinanden. En eksplicit rolle erstatter elementets egen. Et link med `role="button"` annonceres som knap, men ruller siden ved tryk på mellemrum; `role="tab"` på en `h2` fjerner den fra listen over overskrifter; `role="navigation"` på en `ul` tager listen og antallet af punkter med sig. Den anden regel for brug af ARIA er kun at ændre indbygget semantik, når det er nødvendigt, og ARIA in HTML opregner, hvilke roller hvert element må få.

En live-region snakker i ét væk. `aria-live="assertive"` eller `role="alert"` sidder på et antal resultater eller en nedtælling, der skifter for hvert tastetryk eller hvert sekund, og skærmlæseren afbryder brugeren for hver ny værdi. Brug `polite`, som `role="status"` giver af sig selv, meld først, når indtastningen er faldet til ro, og hold regionen stille, når siden indlæses.

Læs mere

- WAI-ARIA 1.2 — alle roller, tilstande og egenskaber, hvilke attributter hver rolle understøtter, og hvilke roller der ikke må have et navn.
- ARIA in HTML — de roller og attributter, hvert HTML-element må have, og hvilke af dem der er overflødige.
- Using ARIA — W3C's praktiske vejledning med reglerne for brug af ARIA og et eksempel på hver.
- Accessible Name and Description Computation 1.2 — algoritmen, der afgør, hvad en skærmlæser siger, at en kontrol hedder.
- Understanding Name, Role, Value — hvad succeskriterium 4.1.2 i WCAG 2.2 kræver af hver komponent, med teknikker og typiske fejl.
- ARIA Authoring Practices Guide — opmærkning og tastaturadfærd for hvert almindeligt widgetmønster, med eksempler, der virker.

Ordliste

Vælg ud fra jeres faktiske vilkår

Beskriv arbejdsgangen, de eksisterende grænseflader og de brugere, der har brug for ændringen. Vi kan definere en testbar leverance og en overdragelse, der gør teamet i stand til at vedligeholde løsningen.

Se, hvad det koster Kontakt os →

Gør begreberne praktiske

Spørg, hvordan »Hvad er ARIA?« gælder i jeres organisation. Et konkret eksempel fra jeres nuværende system er nok til at begynde samtalen.

Se vores ydelser

welcome@engineer.company +45 50 10 61 84 København, Danmark

Tag denne side med dig

Visitkort

Lavet med kærlighed af D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – nu · Engineer ApS · Alle rettigheder forbeholdes.

Udskrevet fra https://engineer.company/da/glossary/aria/