ISTITUTO NAZIONALE DI FISICA NUCLEARE
Direzione Sistemi Informativi

INFN-24-03-DSI
1 Aprile 2024

Accessibilità delle Web APP della DSI

Marzio D'Alessandro[1], Claudio Ciamei[1]

[1)] *INFN, Direzione Sistemi Informativi, I-00044 Frascati (Roma), Italy*

Abstract

Rendere un'applicazione web accessibile, oltre ad essere un dovere sociale, permette ad ogni utente di poter usufruire delle sue funzionalità in un modo più profittevole.
Gli aspetti caratterizzanti l'accessibilità si dividono in due categorie principali: visibili e invisibili.
I primi sono degli aspetti di cui l'utente ha un'esperienza diretta e sono ad esempio i colori utilizzati e il modo in cui sono realizzati gli elementi interattivi della pagina.
I secondi, invece, corrispondono al modo in cui le pagine web sono codificate e possono essere analizzate da un lettore automatico.
In questa trattazione saranno discusse entrambe le tipologie di aspetti e il focus sarà puntato sulla loro gestione nell'ambito dello sviluppo interno alla DSI, che ha tra i propri obiettivi la realizzazione di applicativi web a supporto delle attività amministrative dell'INFN.

DOI n. 10.15161/oar.it/143726

Published by
Laboratori Nazionali di Frascati

1 Le motivazioni

Il Codice dell'Amministrazione Digitale (Decreto Legislativo 7 marzo 2005 n. 82) ha imposto un vero percorso di cambiamento, prevedendo tra l'altro, che il canale preferenziale per l'interazione tra Pubblica Amministrazione ed utenti debba essere, in prevalenza, l'utilizzo delle tecnologie informatiche e la Rete Internet.

Si presenta quindi necessario rendere queste informazioni fruibili e accessibili a tutte le persone, anche a quelle che presentano una disabilità.

L'accessibilità è un tema sentito fin dagli albori del web, sia per garantirne l'utilizzo a una fetta più ampia possibile della popolazione, sia per dare delle linee guida di sviluppo di interfacce web che garantiscano in generale una migliore UX (*User Experience*).

Si stima infatti che il 15% delle persone nel mondo (circa un miliardo) abbiano delle imparità raggruppabili in 5 gruppi [1].

- **Uditivo**: occorre fornire le trascrizioni per gli audio e i sottotitoli per video.
- **Cognitivo**: va usata una struttura consistente, una etichettatura descrittiva e chiara, link predicibili e un linguaggio semplice.
- **Fisico:** gli elementi interattivi devono avere un target size abbastanza ampio e sufficiente margine l'uno dall'altro; occorre permettere l'uso della tastiera e supportare tecnologie di assistenza come gli switches controls, i tracciatori oculari e i comandi vocali.
  Inoltre, è importante lasciare la possibilità di attivare combinazioni di tasti in modo differenziato (sticky keys).
- **Parola:** alcune persone non sono in grado di parlare; quindi, si dovrebbe evitare di sviluppare un sito basato solo su comandi vocali.
- **Visuale:** i testi e le immagini devono poter essere allargati senza perdita di informazioni, deve esserci un buon contrasto tra testo e sfondo e non bisogna affidarsi unicamente al colore per trasmettere un'informazione.
  Tutte le informazioni devono essere fornite anche in forma testuale "leggibile" da tecnologie assistive come gli screen readers, deve essere possibile una chiara identificazione delle funzionalità presenti nella pagina e un loro scorrimento ordinato.

Considerate le precedenti motivazioni, già nel 1996 è stata costituita la WAI (Web Accessibility Initiative) [2], un gruppo di lavoro della W3C che ha definito una serie di principi di accessibilità in collaborazione con privati e ha definito delle regole tecniche che gli sviluppatori devono seguire nella realizzazione dei siti web.

Tali regole sono raccolte nelle WCAG, la cui versione 1.0 uscì nel 1999, la 2.0 nel 2008, la 2.1 nel 2018 e la 2.2 ad ottobre 2023 e sono distribuite su tre livelli di conformità A, AA e AAA dal meno al più stringente.

Il nostro riferimento sono le WCAG AA [3].

1.1 Il mio sito web è accessibile?

Per determinare se un sito web, quindi ciascuna pagina di cui è composto, è accessibile occorre farsi le seguenti domande.

1) È facile per ogni tipologia di utente leggere le informazioni presenti nelle pagine ed utilizzare le funzionalità messe a disposizione?

2) È possibile navigare il sito senza saltare elementi di interesse e senza aver difficoltà a operare con i componenti interattivi, usando esclusivamente la tastiera o uno screen reader?

3) Gli elementi delle pagine sono codificati in maniera da essere intercettati e correttamente annunciati da uno screen reader?

I tool automatici che analizzano il codice HTML e CSS di un sito per una valutazione sull'accessibilità, riescono a rispondere in maniera esauriente solamente alla terza domanda, per le altre due è necessario ricorrere a un audit manuale.

Ci sono vari riferimenti in rete che guidano l'auditor nella verifica del sito, anche procedure a step che permettono di produrre un report di accessibilità come nel tool Chrome extension - Accessibility Insights for Web [4], descritto nella sezione sui Tool, o direttamente il report tool [5] della WAI.

La verifica sull'accessibilità attraverso la visualizzazione e l'utilizzo del sito finale, deve coprire i fattori più disparati e comprende valutazioni soggettive che solo l'esperienza aiuta a prendere nel miglior modo.

1.1.1 Focus su applicativi web

Il focus principale di questa trattazione sono gli applicativi web, che possiedono una dinamicità senz'altro maggiore dei classici siti informativi statici, data da un maggior utilizzo di dialoghi e cambi del DOM a seguito delle interazioni utente.

In particolare, le nuove Web App della DSI (Direzione Sistemi Informativi) sono Single Page Application, basate su Angular 2+, che non possono essere analizzate automaticamente nella loro interezza da un tool automatico a differenza di quanto è possibile fare con i siti statici del nostro Ente.

Quest'ultimi, per lo più siti sviluppati in linguaggio PHP mediante l'uso del CMS Wordpress, sono esaminati periodicamente dalla compagnia Siteimprove [6], che fornisce un report contenente i problemi riscontrati e un punteggio complessivo in centesimi dell'accessibilità verificata.

2 Indicazioni per lo sviluppo e tool di supporto

In questo capitolo verrà descritto il metodo utilizzato internamente alla DSI per la produzione di applicativi web accessibili.

Affermare che un applicativo web sia accessibile al 100% non è in realtà possibile, al massimo si può ottenere da parte di una terza parte, che sia universalmente riconosciuta e apprezzata nel settore, la certificazione di un apprezzabile livello di accessibilità raggiunto.

Quindi, l'obiettivo interno è sviluppare applicativi che abbiano il massimo grado di usabilità e accessibilità raggiungibili in relazione alla disponibilità di risorse interne, affidandoci ai punteggi restituiti dai tool, le buone prassi indicate da esperti del settore e il buon senso che ciascun sviluppatore e auditor sviluppa nel tempo.

Vi sono tre aspetti principali da approfondire:

1. Accorgimenti e linee guida durante lo sviluppo
2. Tool di supporto allo sviluppo
3. Tool di verifica nel browser

Prima di descrivere nel dettaglio ciascuno di questi punti, si ricorda che il processo seguito per conseguire e verificare l'accessibilità all'interno della DSI è in continua evoluzione e stiamo procedendo internamente a evolvere i vari applicativi in questa direzione.

2.1 Linee Guida per lo sviluppo

Lo sviluppo dei frontend web si avvale spesso di librerie di appoggio, che espongono componenti html interattivi e stilizzati che possono facilmente essere integrati nelle nuove applicazioni.

Il nostro sviluppo interno Angular si basa sulla libreria PrimeNg [7], che durante il 2023 ha subìto vari aggiornamenti per aumentare il grado di accessibilità dei suoi componenti.

Quindi alla base del grado di accessibilità delle applicazioni sviluppate internamente c'è sicuramente quello della libreria PrimeNg che va tenuta aggiornata in accordo alla versione Angular utilizzata.

Di seguito un elenco (non esaustivo) delle best practices e linee guida seguite internamente per lo sviluppo di app web accessibili.

Importante sottolineare che le regole indicate, soprattutto gli approfondimenti, derivano dallo studio di documentazione presente sul web e in particolare [8].

1) Quando si usano i componenti messi a disposizione dalla libreria PrimeNg, occorre ricordare che attributi fondamentali per l'accessibilità come **id** e **aria-label** (descritti in seguito) non possono essere applicati direttamente ma va usato un loro derivato, rispettivamente **inputId** e **ariaLabel**.

```
<p-button icon="pi pi-refresh" (click)="search()" ariaLabel="Refresh lista">
</p-button>

<p-dropdown inputId="sede" [options]="sedi" placeholder="Sede INFN">
</p-dropdown>
```

2) Quando non si usano direttamente componenti messi a disposizione dalla libreria, occorre usare i tag base html secondo la loro semantica originale, evitando di forzare l'utilizzo di un tag per simularne un altro.
   Esempi:
   - Evitare di usare o elementi custom che simulano bottoni o link.
   - Evitare di usare link con comportamento di bottoni e viceversa.
     Un link conduce a un'altra pagina o fragment (#) della pagina stessa; invece, il bottone causa un' azione confinata nella stessa pagina in cui si trova.

   Inoltre, occorre valorizzare propriamente gli attributi dei tag, per esempio il tag <a> senza l'attributo "href " non prende il focus.
   Nel caso in cui si debba o si scelga di usare un elemento custom, tenere a mente i seguenti punti.

   a) Ricordare di descriverli correttamente sia con **aria-role** che **aria-label.**
   b) Ricordare di dargli un **tabindex** pari a zero.
   c) Ricordare di definire il comportamento del keydown in modo da triggerare la stessa funzione richiamata dal click.

Nota tecnica: in caso di pressione del tasto "Enter" occorre replicare il comportamento del click, altrimenti l'evento va propagato per evitare l'intrappolamento del focus.

d) Verificare che ci sia un focus visibile e che ci si possa interagire con solo la tastiera

3) Gli elementi interattivi devono avere un nome e possibilmente una descrizione.

a) Ai bottoni senza testo va specificata un **aria-label**
b) Ogni input deve avere una label associata. Di seguito tre alternative.

```
<label for="input-id"> <input id="input-id">

<p id="p-id"> <input aria-labelledby="p-id">

<input id="input-id" aria-label="input description">
```

Occorre tenere a mente che le label sono lette da screen reader solo quando associate ai tag **input**, **select** e **textarea**. Inoltre, sono ancora più importanti quando associate a input di tipo **radio** o **checkbox** perché se cliccate invertono il valore del relativo input, contribuendo così ad aumentare il suo target size, altro aspetto importantissimo per un design accessibile.

c) Nel caso sia supportata più di una lingua ricordare di modificare il valore dell'attributo lang del tag html e in accordo tradurre le aria-label.

4) Da notare che gli accordion, quando chiusi, devono impostare a visibility hidden il contenuto, in quanto non è sufficiente usare gli attributi aria-hidden='true' ed aria-expanded='false'.
Infatti, se il contenuto contiene elementi che possono prendere focus (come <input>) lo screen reader andrà a leggerli anche quando l'accordion è chiuso, creando confusione all'utente [9].

5) Gli elementi **th** delle tabelle devono avere sempre un nome discernibile, non va lasciato vuoto.

6) Utilizzo di alternative text (attributo **alt**) per le immagini, ad esclusione dele immagini con solo scopo decorativo.

7) Usare misure relative per le dimensioni dei font e gli altri componenti della pagina (rem o em) in modo da permettere all'utente di beneficiare a pieno dello zoom del browser.

2.1.1 Approfondimento sul supporto allo zoom

Oltre l'utilizzo degli screen magnifier ci sono due vie per effettuare lo zoom di una pagina web e per ciascuna di esse occorre tenere a mente alcune regole in fase di sviluppo per evitare che l'utente che le utilizza non abbia problemi di accessibilità.

1) Aumentare la dimensione del testo nelle preferenze del browser (la dimensione di default dei font è 16px corrispondente a 1rem).

```
.model-name {   /* versione non accessibile */
      font-size: 18px;
      line-height: 22px;
      height: 66px;
      overflow: hidden;
      text-overflow: ellipsis;
      display: -webkit-box;
      -webkit-line-clamp: 3;
      -webkit-box-orient: vertical;
}

.model-name { /* versione accessibile */
      font-size: 1.125rem;
      line-height: 1.2;
      overflow: hidden;
      text-overflow: ellipsis;
      display: -webkit-box;
      -webkit-box-orient: vertical;
}
```

In questo scenario affinché il testo non si sovrapponga ad altri elementi, rendendone difficile la lettura ed utilizzo, occorre usare misure relative per il font-size e line-height ed evitare di usare contenitori ad altezza fissa.

2) Effettuare uno zoom dell'intera pagina. La misura massima dello zoom che occorre supportare, senza che qualche elemento del sito sia tagliato o oscurato è del 200%.

A tale scopo vi sono una serie di punti su cui prestare attenzione.

a) Rendere il sito responsive, in modo tale che all'aumentare dello zoom vengano triggerati i media queries.

b) Evitare lo scrolling orizzontale.
c) Prestare attenzione all'utilizzo di "position: fixed;" in quanto potrebbe rendere inaccessibile il relativo contenuto all'aumentare dello zoom.
d) Evitare di usare testo nelle immagini e preferire l'utilizzo di grafica vettoriale alla raster, ove possibile quindi utilizzare SVG invece di PNG o JPG.
e) In fase di design va utilizzato sapientemente lo spazio soprattutto per il contenuto chiave della pagina che non deve essere troppo vicino ad altri elementi.

2.1.2 Alcune note sui link

I differenti stati dei link permettono agli utenti di interagire più facilmente con essi. Per chiarire meglio, lo stato visitato può aiutare una persona, che soffre di perdita di memoria a breve termine, a ricordare quale link ha navigato; lo stato "hover" aiuta le persone con ridotto controllo muscolare a sapere quando cliccare e lo stato focus indica quale link stanno per attivare agli utenti che usano le tastiere per navigare il sito.

```
a:link {
  color: red;
}
a:visited {
  color: green;
}
a:hover {
  color: black;
}
a:active {
  color: blue;
}
```

Per non creare problemi a utenti con imparità visive è opportuno non rimuovere mai la sottolineatura di un link mediante la regola CSS "text-decoration: none".
Rimane possibile, a scopi di resa grafica, personalizzare la sottolineatura con "text-underline-offset" e "text-decoration-color".

2.1.3 Approfondimento sui form

Di seguito una serie di punti da tenere a mente.
1) Nei form occorre segnalare all'utente i campi di input obbligatori, di solito con un asterisco al termine della relativa label e mediante l'utilizzo dell'attributo "required".

```
<label for="email">Your email address * </label>

<input id="email" name="email" required aria-required="true" placeholder="Email">
```

2) In caso di errore va fornito all'utente un messaggio di errore accessibile senza affidarsi solamente a icone o colori. Tale messaggio testuale va posto vicino all'elemento errato e deve contenere una chiara spiegazione sul come correggere l'errore.

   Quando l'utente sottomette il form, il focus deve essere posizionato sul primo campo invalido.

```
<input name="firstName" id="firstNameInput" type="text" pattern="[^.]*?" aria-describedby="firstName-length-error" aria-invalid="true">



<p id="firstName-length-error" role="alert">

Your first name must have at least two letters and no unusual characters</p>
```

3) Nel caso di più input collegati tra loro come nel caso di date di nascita va usato l'elemento <fieldset> in congiunzione con <legend>.

```
<fieldset>

  <legend>Your date of birth</legend>

  <label for="dobDay">Day</label>

  <select id="dobDay">…</select>

  <label for="dobMonth">Month</label>

  <select id="dobMonth">…</select>

  <label for="dobYear">Year</label>

  <input id="dobYear" type="text" placeholder="YYYY">
```

4) Utilizzare dove possibile l'attributo "autocomplete" per facilitare la compilazione dei form [10].

```
<input id="email" autocomplete="email" name="email" aria-required="true"
placeholder="Email" required>
<select id="dobDay" autocomplete="bday-day" aria-required="true" required>
<select id="dobMonth" autocomplete="bday-month" aria-required="true" required>
<input id="dobYear" autocomplete="bday-year" placeholder="YYYY" aria-required="true"
required>
```

2.2 Tools di supporto e di verifica

Deque [11], azienda leader nel mercato dell'accessibilità, afferma che con AXE, la sua testing engine di accessibilità, possono essere rilevate automaticamente fino a un 57% delle WCAG, azzerando i falsi positivi.

AXE [12] è stata scelta da Google come base dei Chrome Dev Tools, in particolare dello strumento **Lighthouse,** attraverso cui è possibile effettuare un report di accessibilità sulla singola pagina del proprio sito web [13].

Prima di passare all'analisi di Lighthouse e dei tool di verifica accessibilità disponibili nel browser di cui facciamo uso, occorre esaminare i *linter* e gli altri strumenti usati durante lo sviluppo.

Lo sviluppo dei frontend Angular viene eseguito su due IDE principali: Visual Studio Code e Web Storm; entrambi supportano l'installazione di plugin per l'estensione delle funzionalità offerte.

Per avere indicazioni durante la codifica del template html dei componenti Angular in relazione al tema dell'accessibilità è possibile avvalersi di segnalatori quali Axe Linter che evidenziano in giallo le righe di codice in cui sono individuate anomalie.

Altro strumento molto utile di cui stiamo valutando un impiego più regolare è SonarQube [14].

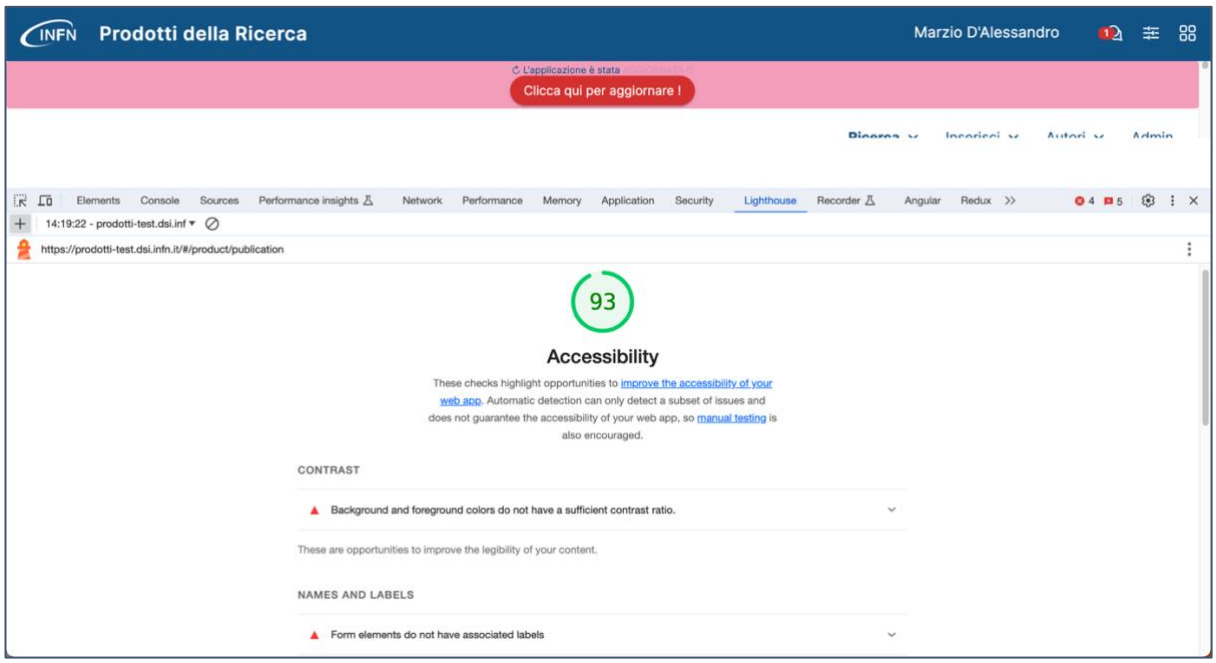
2.2.1 Tools di verifica nel browser

Il browser di riferimento per gli sviluppi, i test di integrazione ed UAT della DSI è Chrome.

Pertanto, a partire da Chrome sono stati individuati e provati vari tool per la verifica dell'accessibilità.

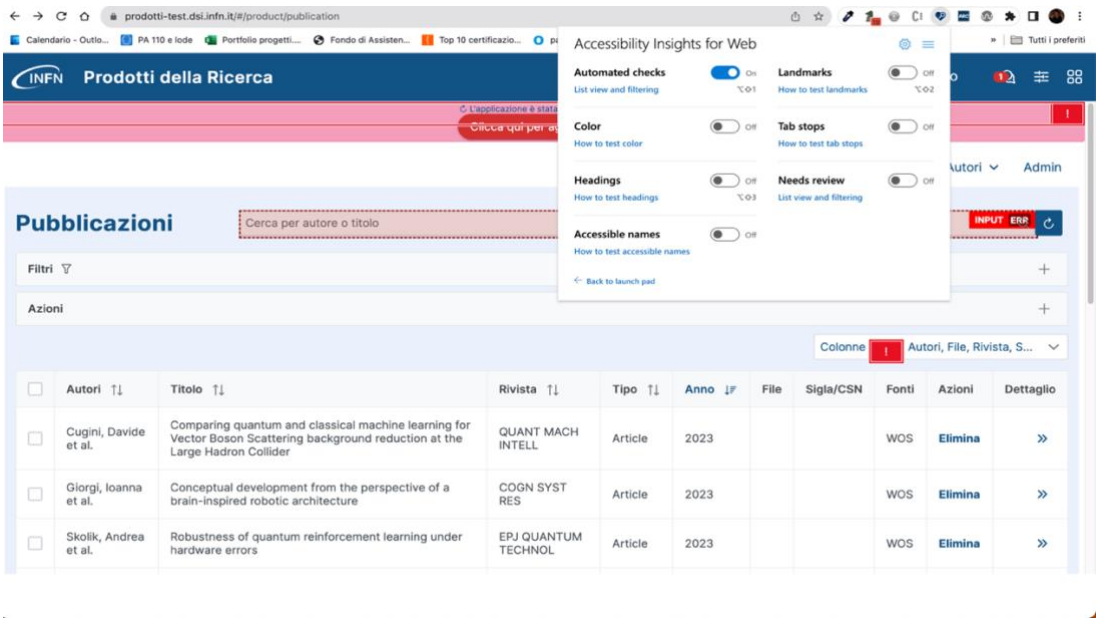
Di seguito un approfondimento sui due tool risultati migliori per la nostra esperienza.

1) Chrome Dev Tools – Lighthouse [15] che definisce accessibile una pagina web avente uno scoring di almeno 90/100 sulla base di ben definiti criteri [16].



Lighthouse restituisce un punteggio che usiamo come base per dire se l'app web è nella giusta direzione per l'accessibilità e indica i vari elementi errati a livello di html, nonché errori sul contrasto di colore.
A volte non è immediato individuare nel codice gli elementi indicati, né vengono descritte esaurientemente le metodologie da adottare per la correzione degli errori; pertanto, è opportuno usare in congiunzione altri tool.

2) Chrome extension - Accessibility Insights for Web



Accessibility Insights for Web segnala direttamente nella pagina in modo preciso gli elementi errati, fornisce un modo per individuarli agevolmente nel codice e ne suggerisce la risoluzione. Inoltre, mette a disposizione altri strumenti per l'analisi della struttura della pagina e guida l'auditor del sito nel costruire report di accessibilità.

3 Verifica manuale dell'accessibilità

Come detto nell'introduzione, per una verifica dell'accessibilità di un sito web è imprescindibile una verifica umana.

Lo stesso sito di Chrome Dev Tools rimanda al canale YouTube **a11ycasts** gestito da Rob Dodson e in particolare all'articolo [17] per avere delle linee guida su come effettuare la review.

Pertanto, anche internamente alla DSI, in misura subordinata alle risorse disponibili, la verifica è fatta in accordo all'elenco presentato di seguito.

1) Verificare che si può navigare il sito con la tastiera premendo il tasto TAB (e SHIFT + TAB) e che gli elementi scorsi abbiano degli stati focus ben visibili.

   a) Alcuni siti presentano lo skip link nell'angolo in alto a sinistra dello schermo, ovvero un link invisibile al cui click l'utente viene portato alla sezione principale della pagina (vedi di seguito *).

   b) Si deve essere in grado di raggiungere tutti gli elementi interattivi usando la tastiera e interagire con essi.

Verificare che non ci sia contenuto web al di fuori della viewport che può prendere focus accidentalmente, come menu di navigazione laterali; o elementi in cui il focus rimane intrappolato.

a) Il focus dagli elementi non va rimosso né nascosto attraverso le direttive CSS

    1. outline: 0
    2. overflow: hidden quando l'elemento che prende focus è contenuto in un elemento troppo piccolo per mostrarlo

    In caso si volesse personalizzare il focus si possono usare le regole css outline-color e outline-offset ma è fondamentale non rimuoverlo.

b) Tenere a mente che gli elementi raggiungibili con il TAB sono quelli di seguito elencati a meno che non abbiano un tabindex valorizzato con -1 o che siano figli di un elemento avente diplay: none [18].
Ricordare che l'ordine di scorrimento secondo tabindex è 1,2,3, …., 0 ed è buona prassi usare solamente tabindex 0 o -1.

    - **input**, **select**, **textarea**, **button**, e **object** non disabilitati
    - **a** e **area** con un **href** valorizzato
    - ogni elemento con un attributo **tabindex** maggiore o uguale a 0.

2) Usare uno screen reader come **NVDA** per Microsoft, **TalkBack** for Android o **VoiceOver** per Mac OS (quest'ultimo si avvia con command + F5 e il suo rotore si accede con control+option+u), in modo da verificare che i vari elementi siano presentati correttamente per il loro ruolo, nome ed altre informazioni (ad esempio: *visited link, Shopping cart: 0 items)*

a) Verificare che le immagini abbiano degli alt text significativi. Ciò vale nel caso di immagini di tipo "descrittivo" e richiedono anche role='img' e un apposito aria-label se implementate mediante CSS background-image, tag <i> o <svg>. Nel caso invece siano immagini di tipo "decorativo" devono avere un alt text vuoto e possibilmente anche un aria-hidden="true".

b) Se ci sono controlli personalizzati, implementati attraverso e il supporto di JS, verificare che siano interattivi con lo screen reader.

c) Se appaiono dialog o elementi dinamici sullo schermo il focus dello screen reader dovrebbe essere ridiretto lì.

3) Verificare la struttura delle pagine.

a) Corretto utilizzo degli headings: deve essere rispettata la gerarchia per H1, H2 e H3 (etc..), di solito c'è un solo H1 per ogni pagina e Hx è contenuto in Hy se x > y, non vanno scelti in base alla loro dimensione ma vanno usati secondo la loro semantica e nel giusto ordine, in quanto definiscono lo scheletro della pagina. A volte per mantenere la gerarchia sono anche usati dei titoli nascosti, leggibili solo da screen reader.

b) Utilizzo appropriato di elementi di landmark o ruoli ARIA (Accessible Rich Internet Applications) perché agevolano la navigazione con gli screen reader. Si parla in generale di HTML semantico. (nello schema sottostante "-> role =" significa equivalente a tag html con attributo **role** valorizzato).

<header> -> role = "banner"

<nav> -> role = "navigation" e "aria-label" nel caso di più nav

<main> -> role = "main"

<aside>

4) Ci deve essere un contrasto di colore sufficiente tra i testi o altri elementi (specialmente quelli interattivi) e lo sfondo.

- Secondo WCAG AA il rapporto minimo deve essere 4.5:1, secondo AAA 7:1 (considerare che il contrasto massimo si ha tra nero/bianco ed è di 21:1)
- In caso di testo su immagine va confrontato il colore del testo con il punto più chiaro o scuro dell'immagine a seconda se il testo sia chiaro o scuro.
- Gli elementi interattivi possono avere vari stati: hover, focus, active, unvisited, visited and deactivated, occorre assicurare che anche questi mantengano un buon contrasto, ad esempio il cambio di contrasto tra uno stato unfocused ad uno focused deve essere di almeno 3.

```
/* - * skip links - */
.skip {
  position: absolute;
  left: -10000px;
  top: auto;
  width: 1px;
  height: 1px;
  overflow: hidden;
}

.skip:focus {
  position: static;
  width: auto;
  height: auto;
}
```

3.1.1 Check aggiuntivi raccomandati da WAI

La WAI mette a disposizione una guida per l'esecuzione dell'audit di accessibilità [19].

Le raccomandazioni fornite coprono dei punti ulteriori rispetto a quelli analizzati finora; di seguito un estratto.

- Verificare che ogni pagina del sito abbia un titolo significativo
- Verificare che il testo delle pagine sia ridimensionabile senza che venga tagliato del contenuto
- Assicurarsi che tutti i form sia correttamente etichettati e siano gestiti i casi di errore
- Nel caso siano presenti animazioni o contenuti con effetti di blinking o flash, occorre dar modo all'utente di fermarli o configurarne la durata e/o velocità.
- Nel caso di contenuto solo video assicurarsi che vi siano dei sottotitoli e che vi siano le trascrizioni dei contenuti audio.

Riferimenti

[1] https://www.w3schools.com/accessibility/index.php

[2] https://www.w3.org/WAI

[3] https://accessibleweb.com/rating/aa/

[4] https://chromewebstore.google.com/detail/accessibility-insights-fo/pbjjkligggfmakdaogkfomddhfmpjeni

[5] https://www.w3.org/WAI/eval/report-tool/

[6] https://www.siteimprove.com/

[7] https://primeng.org/

[8] https://www.w3schools.com/accessibility/

[9] http://web-accessibility.carnegiemuseums.org/code/accordions

[10] https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes/autocomplete

[11] https://www.deque.com/

[12] https://github.com/dequelabs/axe-core

[13] https://www.deque.com/blog/google-selects-deques-axe-chrome-devtools/

[14] https://medium.com/@ishant-sahu/sonarqube-integration-for-react-projects-80deeed78e92

[15] https://developer.chrome.com/docs/devtools/accessibility/reference

[16] https://developer.chrome.com/docs/lighthouse/accessibility/scoring/

[17] https://web.dev/articles/how-to-review?hl=it

[18] https://itecnote.com/tecnote/javascript-focus-next-element-in-tab-index/#google_vignette

[19] https://www.w3.org/WAI/test-evaluate/preliminary/