

Contenidos Digitales Accesibles

Accesibilidad Web

ARIA en la Accesibilidad

Definición general

A grandes rasgos: *“Conjunto de atributos que se añaden a las etiquetas HTML para que los agentes de usuario (navegadores y productos de apoyo) comprendan que estas etiquetas tienen un comportamiento diferente al habitual y actúen en consecuencia.”*

<https://www.w3.org/TR/using-aria/>

ARIA Accessible Rich Internet Application

Accessible Rich Internet Applications (ARIA) ayuda a crear contenido Web (especialmente las desarrolladas con Ajax y JavaScript) más accesibles a personas con discapacidades.

ARIA es un conjunto de atributos especiales para accesibilidad que pueden añadirse a cualquier etiqueta, pero están pensadas especialmente para HTML.

Está implementado en la mayoría de los navegadores y lectores de pantalla más populares. Sus implementaciones varían y las tecnologías antiguas no lo soporta del todo bien (si es que lo soportan).

Se recomienda el uso de *"safe"* en ARIA, que reduce elegantemente la funcionalidad en caso necesario o sugiere a los usuarios actualizarse a una tecnología más actual.

ARIA Funciones

Esta especificación está basada en tres atributos diferentes: **roles**, **estados** y **propiedades**.

Sus funciones son las las siguientes:

- **Roles** (roles): describen algunos de los elementos y widgets comunes en la UI web pero que no están disponibles en HTML, como *sliders*, *pestañas*, etc., contruidos con HTML, CSS y JavaScript. También se utilizan para describir la estructura de la página incluyendo encabezados, tablas y otros elementos.
- **Propiedades** (properties): describen los estados de los widgets. Por ejemplo, cuando un elemento en la página es “arrastrable”, tiene un elemento popup asociado con él.
- **Estados** (states): describen si un elemento es interactivo o no, es decir, los ‘estados’ informan a los lectores digitales si el estado de un elemento en la página es ‘ocupado’, ‘deshabilitado’, ‘seleccionado’ u ‘oculto’.

ARIA Principales características

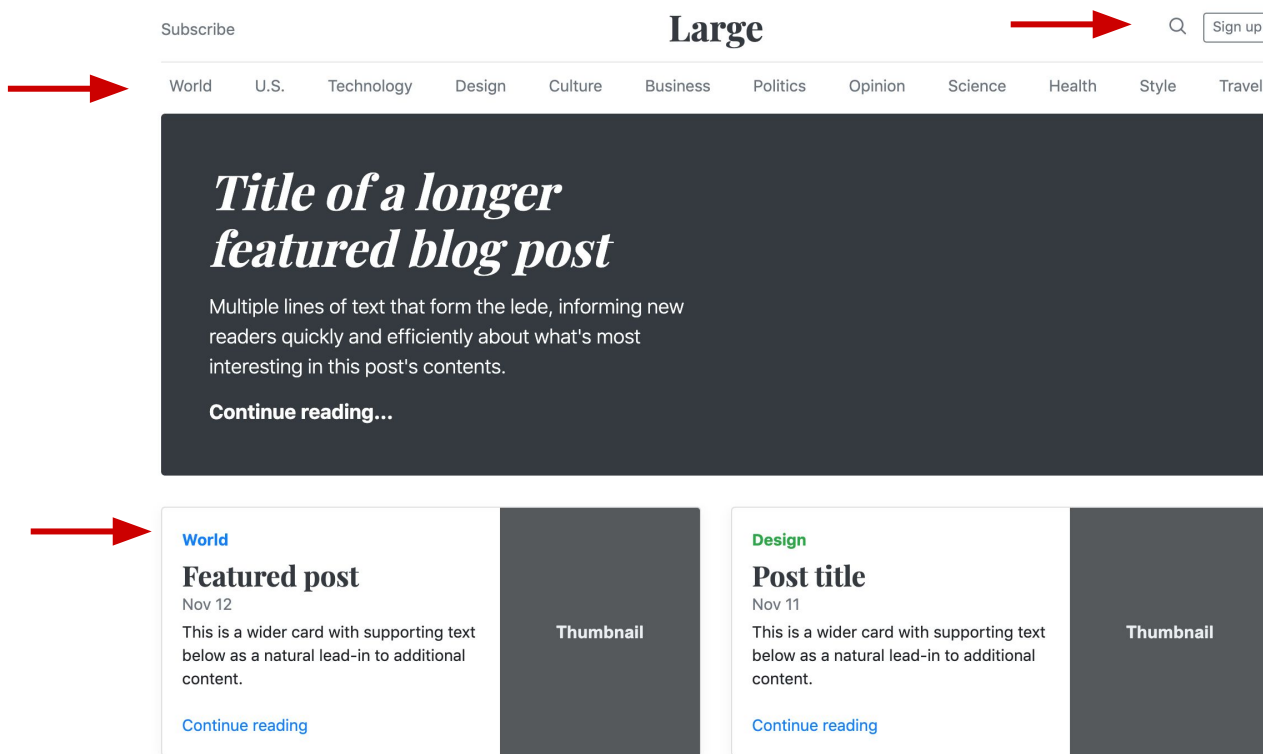
- La W3C ha desarrollado la tecnología ARIA o WAI-ARIA
- Aria es de 2008, su versión actual es **WAI-ARIA 1.1 de 2017**
- Permite añadir información semántica a cualquier elemento de la interfaz web.
- Los agentes de usuario transmiten esa información a la tecnología de apoyo para que se dé una mejor interacción con el sitio Web.
- ARIA comprende: una taxonomía de roles, estados y propiedades que ayudan a enriquecer el comportamiento de los elementos de la interfaz.
- Es necesario conocer la dificultad del usuario que accede con tecnologías de apoyo (lector de pantalla). Es decir, ***hay que testear con estas herramientas.***

ARIA Componentes interactivos

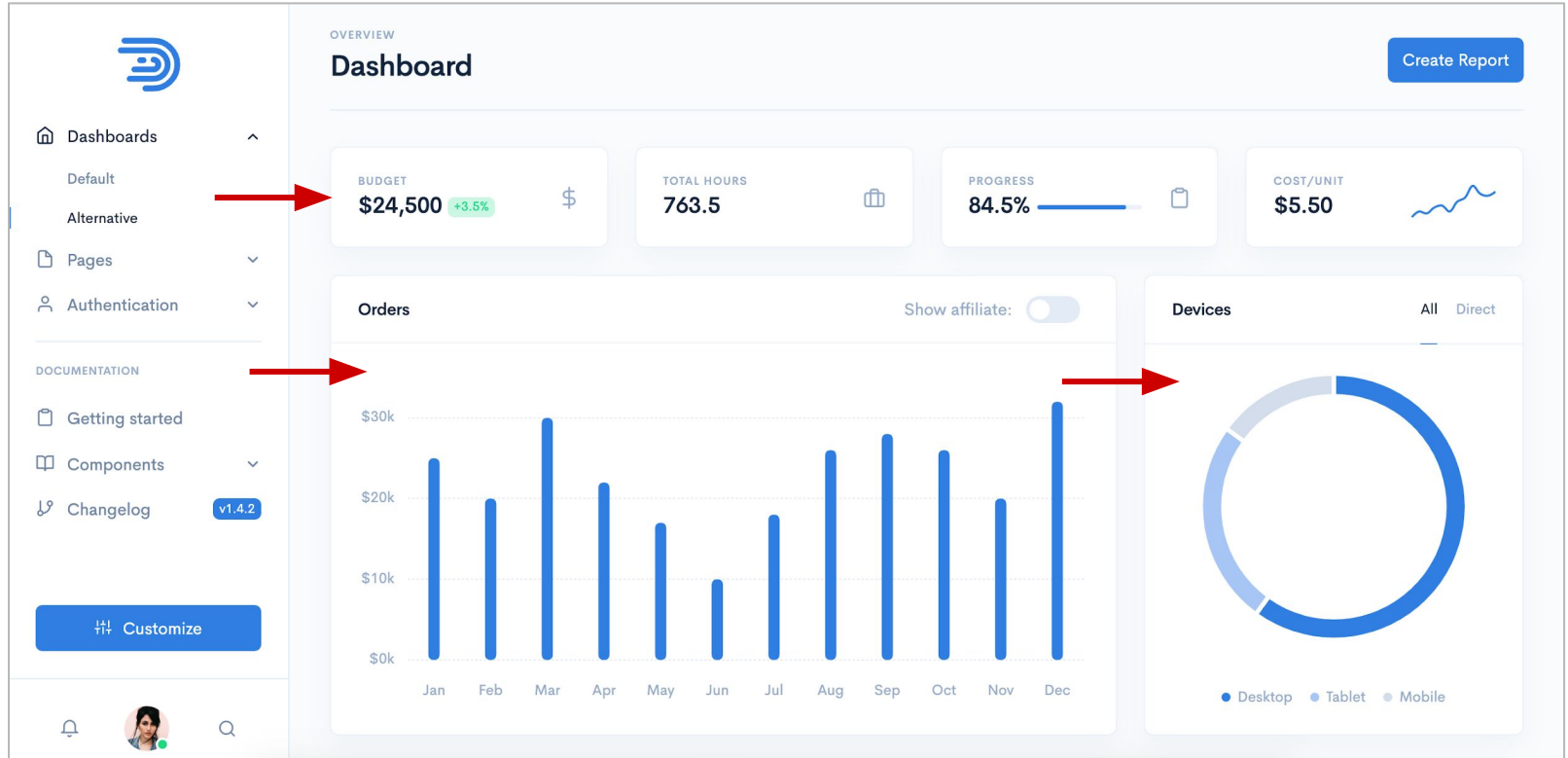
- Permite crear elementos web complejos:
 - Desplegables, destiladores, alertas, cuadros de diálogos
 - Tooltips flotantes, barras de progreso, pop-bover con opciones de diálogo
 - Árboles de contenido y listas de elementos
 - posicionables y para desplegar
 - Drag-and-drop
 - Campos obligatorios en formularios y errores de validación.
 - Zonas con contenido ajeno al usuario:
 - Tiempo para completar una tarea
 - Redes sociales (Twitter, Facebook)
 - Mensajes de error al rellenar un formulario



ARIA Ejemplos de uso



8 ARIA Ejemplos de uso (ii)



Cómo usar WAI-ARIA



Añadir la dtd de la W3C en el DOCTYPE correspondiente para evitar errores de validación:

```
<! DOCTYPE html PUBLIC "-//W3C//DTD XHTML+ARIA 1.0//EN"  
"http://www.w3.org/WAI/ARIA/schemata/xhtml-aria-1.dtd">
```

```
<! DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML+ARIA 1.0//EN"  
"http://www.w3.org/WAI/ARIA/schemata/html4-aria-1.dtd">
```



ARIA HTML y ARIA



En un Sitio Web lo ideal es usar los elementos HTML para que cumplan su función original: Botes, Enlaces, Secciones, etc.

Hay muchos elementos HTML que por exigencias del diseño, se utilizan definir elementos diferentes

- DIV como Botones o enlaces
- Pestañas y Ventanas modales (Bootstrap)
- Árbol desplegable y Mensajes de Alertas.

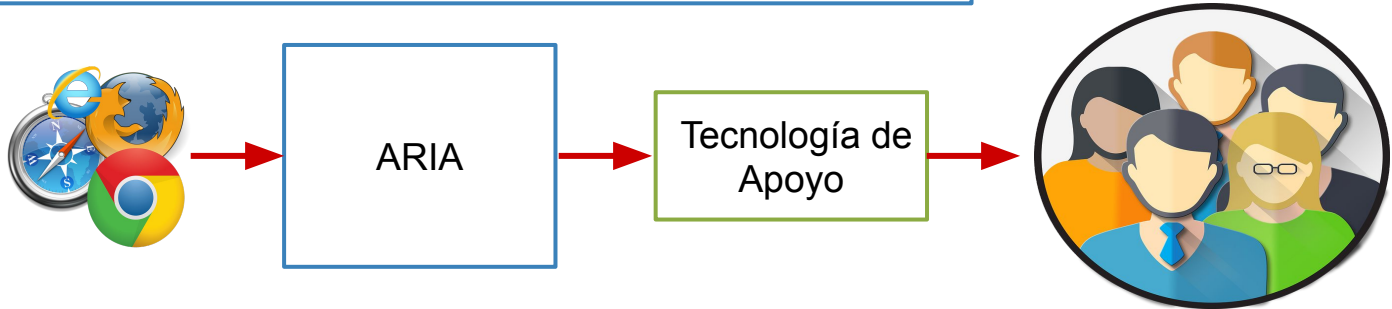
``
 ↓ ↓ ↓
- Elemento 1
- Elemento 2

``



ARIA HTML y ARIA (ii)

```
<ul role="tree" aria-label="Lista de Opciones">  
  <li role="treeitem" aria-expanded="false"  
    aria-posinset="1" aria-setsize="3"  
    aria-level="1" tabindex="0">  
    ....  
  </li>  
</ul>
```



Lector de pantalla: “Árbol Listado de Opciones Nivel 1 Opción X contraído, 1 de n”.

Comportamiento de los elementos HTML (i)

```
<div></div>
```

```
<button> => <div>
```

```
role="button"
```

```
<div role="button">
```

```
  
```

```
</div>
```



- Botón en el listado de campos del lector
- Se accede a él mediante el teclado, e.g. "b".

Special page

Log in

Username

Password [Forgot your password?](#)

☐ Keep me logged in (for up to 30 days)

[Help with logging in](#)

Don't have an account?



Botón gráfico Log in

Comportamiento de los elementos HTML (ii)

Con solo `role="button"` no se va a comportar como botón

Hay que añadir más comportamiento de botón incluyendo los eventos con JavaScript (*[onclick](#)*, *[onkeypress](#)*)

- Foco en el botón con el tabulador: no será accesible por teclado pero si por ratón.
- Añadir el atributo `tabindex=0`: indica al navegador que incluya al `<div>` en la lista de elementos.
- Es posible usar el Tabulador para llegar al elemento `<div>` *(en el orden que le corresponda en el DOM)*
- Con ESPACIO o ENTER para pulsarlo.

```
<div role="button" tabindex="0" id="buttonEnviar">
  
</div>
```



Botón gráfico Log in



ARIA o HTML

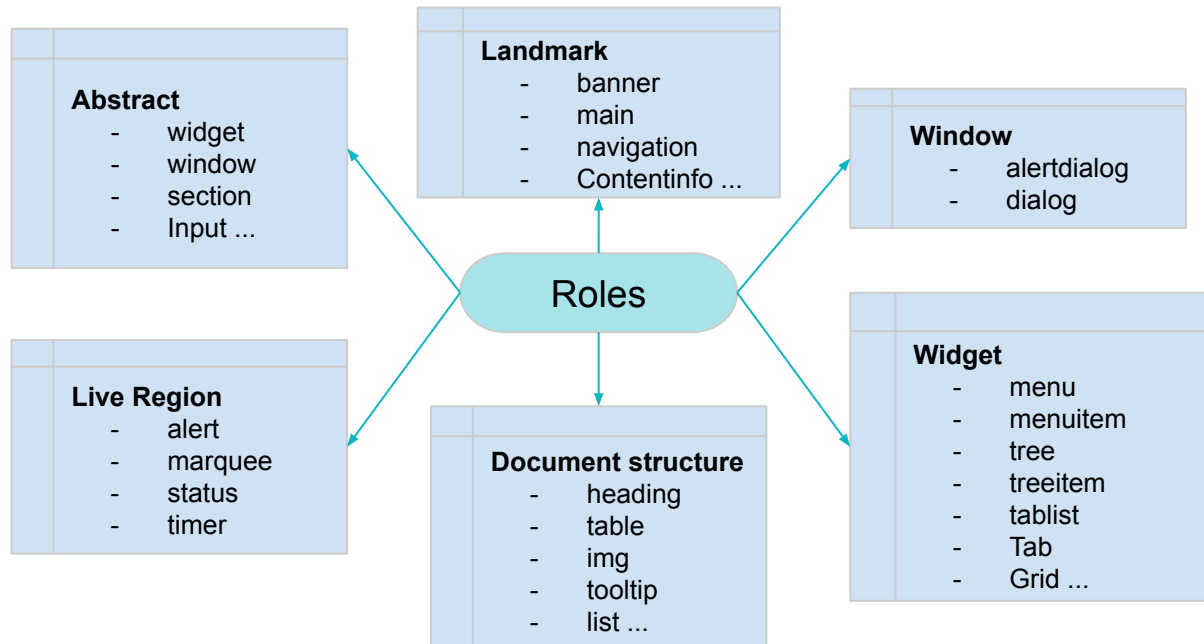
- Es mejor usar la etiqueta nativa.
- ARIA es solo un complemento que potencia la accesibilidad.
- No se debe usar como sustituto de los elementos nativos.

¿Cuándo se recomienda utilizar ARIA y no elementos HTML:

- No hay el HTML para el elemento que se quiere.
- Está el elemento HTML pero los agentes de usuario no la reconocen.
- Está el elemento HTML el agente de usuario no brinda opciones de accesibilidad de ese elemento.
- Los de diseño no “obligan” a usar un estilo concreto y con elementos HTML cuesta lograr dicho efecto.

Roles [role="..."] (i)

Permiten señalar qué función cumplirá un elemento de la Web.
En total son 81 roles organizados se muestra a continuación:



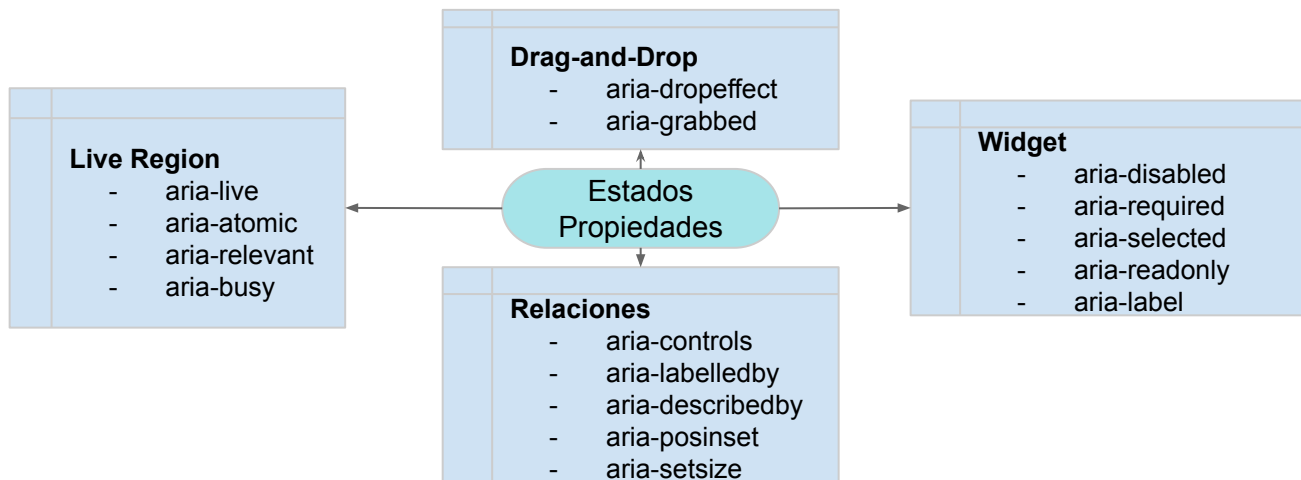
Roles [role="..."] (ii)

- **Abstract:** no son para contenido, sino para definir roles en general
widget, window, section, input, ...
- **Widget:** menu, menuitem, tree, treeitem, tablist, tab, tabpanel, grid, etc.
- **Document Structure:** por lo general no son para interactuar
heading, table, omg, tooltip, list, presentation, ...
- **Landmark:** se usan para establecer zonas de la página del mismo modo que lo hace HTML5 (*reader, main, nav, footer, otros*). Los LP por defecto tienen atajos para llegar a estas zonas: *ir de zona en zona, usar la estructura de la página generado (PDF)*.
- **Live Region:** especificar zonas “vivas” de la página (que cambian sin la acción del usuario: *alert, log, marquee, status y timer*).
- **Window:** mostrar capas con modo de ventana *alertdialog y dialog*.

Estados y Propiedades (i)

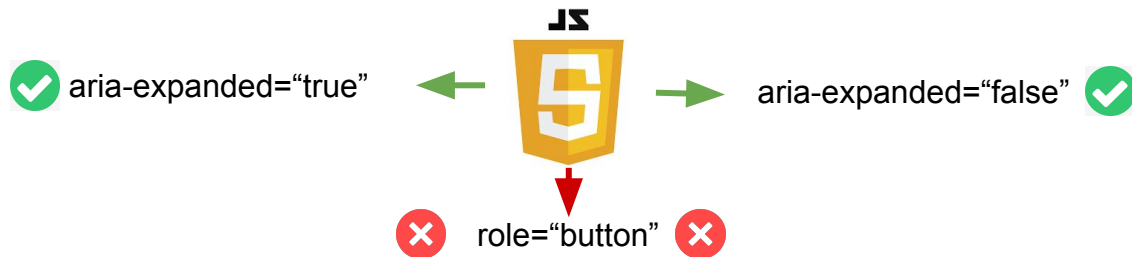
ARIA establece los Estados y las Propiedades de los controles, 48 en total.

- **Estados:** cambian mucho por la interacción con el usuario
- **Propiedades:** cambian con menos frecuencia
- Comienzan por **aria-**
- Se dividen en cuatro categorías: **Drag-and-Drop, Widget, Live Region, Relaciones**



Estados y Propiedades (ii)

- **Atributos de Widget**
aria-checked, aria-disabled, aria-required, aria-selected, aria-readonly, aria-expanded, aria-label
- **Atributos de Live Region:** aria-live, aria-atomic, aria-relevant y aria-busy
 - Cuando la tecnología anuncia los cambios producidos en las zonas que se actualizan solas, qué parte se anunciará,
 - Qué tipo de actualización se quiere anunciar
 - Si no se quieren anunciar por cierto tiempo.
- **Atributos de Drag-and-Drop:** aria-dropeffect y aria-grabbed.
- **Atributos de relaciones:** indican las relaciones entre elementos
aria-controls, aria-labelledby, aria-describedby, aria-posinset, aria-setsize, ...



Estados y Propiedades (iii)

WCAG 2.1 promueve aún más el uso de **aria-label**

`<button aria-label="Cerrar">X</button>`



Botón Cerrar

`Herramienta`



Enlace Ir a la
herramienta

Landmark: diferenciar zonas de la páginas roles del mismo tipo

`<div role="navigation" aria-label="Menú Lateral">`

Estados y Propiedades (iv)

WCAG 2.1 promueve aún más el uso de **aria-label** y **aria-labelledby**

- Indican el contenido de la etiqueta del elemento.
- Si el nombre de la etiqueta está visible en otro elemento de la página, se recomienda usar **aria-labelledby**, referenciando el id(s) del elemento que contiene dicha información.
- **aria-label** **anula** a **aria-labelledby**.
- No se usan juntos.
- Anula la etiqueta nativa.

Estados y Propiedades (v)

WCAG 2.1 promueve aún más el uso de **aria-label** y **aria-labelledby**

```
<h2 id="catalogo"> Catálogo de Productos </h2>
<p>
  <a id="fichero" aria-labelledby="catalogo fichero" href="">
    Descarga Fichero
  </a>
</p>
```



Catálogo de productos Descargar
fichero

```
<h2 id="catalogo"> Catálogo de Productos </h2>
<p>
  <a id="fichero" aria-labelledby="catalogo fichero" href="">
    
  </a>
</p>
```

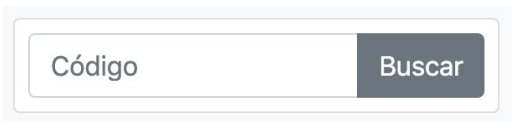


¿?

Estados y Propiedades (vi)

WCAG 2.1 promueve aún más el uso de **aria-label** y **aria-labelledby**

```
<input type="text" name="etiquetaBtn" aria-labelledby="btnBuscar">  
<input type="submit" name="btnBuscar" id="btnBuscar" value="Buscar">
```

A visual representation of the HTML code above. It shows a light gray rounded rectangle containing a white text input field with the placeholder text 'Código' and a dark gray button with the text 'Buscar'.

Buscar

```
<label for="timeSesion" id="endSesion">
```

Extender el tiempo de sesión

```
</label>
```

```
<input type="text" id="timeDuration" value="5" aria-labelledby="endSesion timeDuration timeUnit">
```

```
<span id="timeUnit"> minutos </span>
```



Extender el tiempo de sesión 5 minutos

Estados y Propiedades (vii)

WCAG 2.1 promueve aún más el uso de **aria-label** y **aria-labelledby**

```
<div role="img" aria-labelledby="puntos">
  
  
  
  
  
</div>

<div id="puntos" aria-hidden="true">
  <span class="oculto"> puntuación </span>
  <span>3 de 5</span>
</div>
```



¿?



¿?

Estados y Propiedades (viii)

WCAG 2.1 promueve aún más el uso de **aria-describedby**

Su función es referenciar al id del elemento que contiene la descripción de otro.

```
<button aria-label="Salir" aria-describedby="descSalir"> X </button>
```

```
<div id="descSalir">
```

Al salir se perderán los cambios que no haya guardado y volverá al menú principal.

```
</div>
```



Botón Salir. Al Salir se perderán los cambios que no haya guardado y volverá al menú ...

```
<img src="" alt="WCAG 2.1" aria-describedby="descPrincipios" />
```

```
<div id="descPrincipios">
```

```
<p>Los principios de WCAG son 4</p>
```

```
<ol>
```

```
<li>Perceptible</li>
```

```
<li>Operable</li>
```

```
<li>Comprensible</li>
```

```
<li>Robusto</li>
```

```
</ol>
```

```
</div>
```



¿?

Estados y Propiedades (ix)

WCAG 2.1 promueve aún más el uso de **aria-describedby**

```
<label for="user"> Usuario (obligatorio) </label>
<input type="text" name="user" id="user"
      aria-describedby="descriptionEmail" aria-required="true" />

<p id="descriptionEmail" class="ayuda">
  El usuario es el correo electrónico.
</p>

<label for="pass"> Contraseña (obligatorio) </label>
<input type="password" name="pass" id="pass"
      aria-describedby="descriptionPass" aria-required="true" />

<p id="descriptionPass" class="ayuda">
  La contraseña debe tener al menos 8 dígitos.
</p>
```

El usuario es el correo electrónico.

La contraseña debe tener al menos 8 dígitos.



Usuario obligatorio El usuario es el correo electrónico.



Contraseña obligatorio. Campo de edición contraseña. Requerido. La contraseña debe tener al menos 8 dígitos. En blanco.

Estados y Propiedades (x)

WCAG 2.1 promueve aún más el uso de **aria-describedby**



```
<ul>
  <li><a href="#cap0">Principal</a></li>
  <li><a href="#cap1">Pestaña 1</a></li>
  <li><a href="#cap2">Pestaña 2</a></li>
  <li><a href="#cap3">Pestaña 3</a></li>
</ul>
```

Pestañas

Contenido

```
<section id="cap0">
  El índice ...
</section>
<section id="cap1">
  Aquí está la pestaña 1...
</section>
<section id="cap2">
  Aquí está la pestaña 2...
</section>
<section id="cap3">
  Aquí está la pestaña 3...
</section>
```

Estados y Propiedades (x)

WCAG 2.1 promueve aún más el uso de **aria-describedby**

```
<ul>
```

Enlaces

```
  <li><a href="#cap0">Principal</a></li>
  <li><a href="#cap1">Pestaña 1</a></li>
  <li><a href="#cap2">Pestaña 2</a></li>
  <li><a href="#cap3">Pestaña 3</a></li>
```

```
</ul>
```

Lista

```
<section id="cap0">
```

El índice ...

```
</section>
```

```
<section id="cap1">
```

Aquí está la pestaña 1...

```
</section>
```

```
<section id="cap2">
```

Aquí está la pestaña 2...

```
</section>
```

```
<section id="cap3">
```

Aquí está la pestaña 3...

```
</section>
```

Secciones

Estados y Propiedades (xi)

Lista:

- Asignar el comportamiento de lista de pestañas: **role="tablist"**.
- Suministrar una etiqueta con **aria-label** o **aria-labelledby**.
- Indicarse que cada elemento de la lista cumple el rol de presentación de contenidos: **role="presentation"**

Enlaces:

- Asignar el rol de pestaña: **role="tab"**.
- La sección que controla: **aria-controls="cap1"**
- Si está seleccionado o no: **aria-selected=true** (con javascript habrá que cambiarlo).
- Establecer **aria-pasinet** y **aria-setsize** para indicar la posición que ocupa y cuántas pestañas hay.

Secciones:

- Asignar su: **roles=tabpanel**
- Habilitar el foco con **tabindex=0**
- Si solo se oculta la pestaña visualmente (text-indent:-1000px), añadir el atributo **aria-hidden=true** (indica que el LP tampoco le verá)
- Cambiar el estado dinámicamente con JavaScript a medida que el usuario pulse las pestañas.
- Añadir el atributo **aria-labelledby** para etiquetarla con el nombre de su pestaña.

Estados y Propiedades (x)

WCAG 2.1 promueve aún más el uso de **aria-describedby**

Lista

```
<ul role="tablist" aria-label="Capítulos del Quijote">
```

Enlaces

```
<li role="presentation">
```

```
<a href="#cap0" tabindex="0" role="tab" aria-controls="cap0"
aria-selected="true" aria-posinset="1" aria-setsize="3"> Principal</a>
</li>
```

```
<li role="presentation">
```

```
<a href="#cap1" tabindex="-1" role="tab" aria-controls="cap1"
aria-posinset="2" aria-setsize="3"> Capítulo 1</a>
</li>
```

```
<li role="presentation">
```

```
<a href="#cap2" tabindex="-1" role="tab" aria-controls="cap2"
aria-posinset="3" aria-setsize="3"> Capítulo 2</a>
</li> ...
```

```
</ul>
```

Estados y Propiedades (x)

WCAG 2.1 promueve aún más el uso de **aria-describedby**

Secciones

```
<section id="cap0" role="tabpanel" tabindex="0" aria-labelledby="cap0">
  El índice ...
</section>

<section id="cap1" role="tabpanel" aria-hidden="true" tabindex="0" aria-labelledby="cap1">
  ...
</section>

<section id="cap2" role="tabpanel" aria-hidden="true" tabindex="0" aria-labelledby="cap2">
  ...
</section>
...
```

Tener en cuenta que...

- **roles=presentation** equivale a **alt=""** y a **aria-hidden** en una imagen: el contenido es ignorado por el lector de pantalla.
- **roles=presentation** en cualquier otro elemento, hace que el LP indique el contenido textual pero elimina la información semántica de su rol original (una lista, una tabla o un encabezado).
- **tabindex=0** y **tabindex=-1** a enlaces: unificar el comportamiento de las pestañas con el acceso mediante teclado:
 - Al entrar el foco en la Pestaña 1 (**tabindex=0**), la manera de moverse entre pestañas es con las flecha derecha e izquierda.
 - Las teclas Inicio y Fin posicionan en la primera y última pestaña
 - Enter o Espacio en una pestaña la activarla.
 - LP anunciará el total de pestañas y le dirá al usuario cuál está seleccionada.
 - Con JavaScript habrá que hacer el rowing tabindex (**de tabindex=-1 a tabindex=0**)

Y Qué pasa con CSS...

- Es necesario que se sincronicen la visualización con la accesibilidad.

```
.treeitem[role="treeitem"] [aria-selected="true"] {  
    color: #000;  
    background-color: #f2f2f2;  
}  
.treeitem[role="treeitem"] [aria-selected="false"] {  
    color: #f2f2f2;  
    background-color: #000;  
}
```

- Usar Accesibilidad de forma nativa y usar javascript “accesible”.

Campos obligatorios

WCAG 2.1 promueve aún más el uso de **aria-describedby**

```
<div id="msgError" aria-live="assertive" role="alert">  
  <p>  
    <svg role="img" aria-label="error">[...]</svg>  
    El campo Email es obligatorio.  
  </p>  
</div>
```



```
<label for="email">Email (Obligatorio):</label>  
<input type="text" id="email" name="email" aria-invalid="true" aria-required="true" value="">
```

Cuidado con ARIA (i)

ARIA da semántica a la interfaz Web, permite describir casi cualquier elemento pero esto elimina la semántica original.

W3C recomienda:

- No utilices ARIA si no es necesario. Usa siempre que puedas etiquetas estándar de HTML de manera estándar. Si puedes usar `<input type="checkbox">` o `<button>` o úsalos en vez de `<div role="checkbox">` o `<div role="button">`.
- Recuerda que el rol de ARIA anula el rol nativo.
- Un rol es una promesa. Si indicas que un `<div>` es un botón (`role=button`) como hemos visto en el primer ejemplo de este capítulo, esto no hace que los navegadores proporcionen a ese elemento el estilo o el comportamiento de un botón, solo conseguimos que sea anunciado como un botón, pero será responsabilidad tuya que se comporte como tal.
- Usa los roles y las propiedades según la especificación. Recuerda que el rol debe cambiar dinámicamente, solo se cambian las propiedades y estados. Debes marcar la estructura del widget (menubar, tablist...) y las relaciones entre sus elementos (`aria-labelledby`, `aria-control`...).

Cuidado con ARIA (ii)

- Evita los conflictos. No añadas ARIA a etiquetas si pueden entrar en conflicto con su propia semántica. Por ejemplo, si redefinimos el comportamiento de un radio button como si fuera un checkbox (**`<input type="radio" role="checkbox" />`**), cada agente de usuario podría implementarlo de una forma diferente, el comportamiento sería caótico, y se perdería la robustez del código.
- Evita la redundancia. No añadas ARIA a los controles nativos con el mismo valor, ya que es redundantes. Por ejemplo, estos ejemplos serían absurdos: **`<input type="checkbox" role="checkbox" />`** **``**.
- Cambia los estados y las propiedades en respuesta a los eventos. Si indicamos que un árbol está abierto con **`aria-expanded=true`**, cuando el usuario lo cierre, debemos cambiar la propiedad por JavaScript y ponerla a false, para que el producto de apoyo pueda anunciar adecuadamente que ahora está cerrado.
- **Accesible por teclado.** En HTML, los elementos que pueden coger el foco por defecto son los elementos de formulario, los botones y enlaces. A cualquier otro elemento que queramos dotar de interacción (un div, un elemento de lista, etc.) deberemos incluir el atributo `tabindex=0` (o `tabindex="-1"` si solo queremos que coja el foco por programación).

Cómo revisar ARIA

➤ Usar un lector de Pantalla:

- NVDA
- VoiceOver

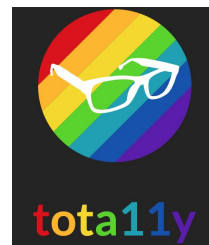


➤ A través de herramientas (Accessibility Viewer)

- Muestra cómo se “visualiza” la información a otras APIs de accesibilidad
- Inspecciona específicamente elementos como roles y sus atributos ARIA.

➤ Otros

- Web Developer Toolbar (Chromes, Firefox y Opera).
- Visual ARIA
- Total11y €€



Las técnicas específicas de ARIA (i)

La WCAG 2.1 incluyen 20 técnicas (34) específicas para la tecnología ARIA:

- Usa la propiedad **aria-describedby** para describir los controles de interfaz de usuario.
- Identifica los campos obligatorios con la propiedad **aria-required**.
- Utiliza los roles para informar del rol de cada componente de interfaz de usuario.
- Utilizas los estados y propiedades para informar del estado de cada componente de interfaz de usuario.
- Etiqueta objetos con **aria-label**.
- Define el objetivo del enlace con **aria-labelledby**.
- Define el objetivo del enlace con **aria-label**.
- Crea una etiqueta concatenando varios nodos de texto con **aria-labelledby**.
- Usa **aria-labelledby** para dar un texto alternativo a contenido no textual.
- Usa los roles landmarks para identificar zonas de la página.

Las técnicas específicas de ARIA (ii)

- Identifica encabezados con **role=heading**.
- Nombre regiones y landmarks con **aria-labelledby**.
- Provee etiquetas invisibles con **aria-label** cuando no puedas usar etiquetas visibles.
- Describe imágenes con **aria-describedby**.
- Nombre controles de interfaz de usuario con **aria-labelledby**.
- Usa roles de agrupación para identificar controles de formulario relacionados.
- Identifica errores con **roles=alertdialog**.
- Identifica errores con **roles=alert** o con live regions.
- Identifica regiones de la página con **role=region**.
- Identifica los campos con errores con **aria-invalid**.

¡Gracias!
¿Un café?

