Développer et coder des sites accessibles

Développer et coder des sites accessibles

Développer des sites conformes aux recommandations des référentiels d'accessibilité numérique (WCAG ou RGAA)

Rappel WCAG & RGAA

Web Content Accessibility Guidelines

Recommendations du W3C de février 2020 : [w3c.org - WCAG 2.2]

Chaque individu qui souhaite utiliser le Web doit avoir du contenu qui est:

- **Perceptible** - L'information et les composants de l'interface utilisateur doivent être présentés à l'utilisateur de façon à ce qu'il puisse les percevoir.
- **Utilisable** - L'interface ne peut pas exiger une interaction qu'un utilisateur ne peut effectuer.
- **Compréhensible** - Compréhension du contenu et de l'interface.
- **Robuste** - Le contenu doit être suffisamment robuste pour être interprété de manière fiable par une large variété d'agents utilisateurs, y compris les technologies d'assistance. Si l'un de ces principes n'est pas vrai, les utilisateurs en situation de handicap ne seront pas en mesure d'utiliser le Web.

Web Content Accessibility Guidelines

Comment ça marche ?

Une liste de règles classées en fonction des quatres grands principes du WCAG est proposée par le W3C. Pour chacune de ces règles s'ensuivent des critères de succès visant à tester la validité de notre code selon les normes du Guideline.

Trois niveaux: A → AA → AAA.

RGAA: Le niveau AA est à viser prioritairement.

En France

Le RGAA

Référentiel Général d'Amélioration de l'Accessibilité

Le RGAA a pour objectif d'encadrer l’accessibilité des contenus diffusés sous forme numérique.

Il ne constitue pas une nouvelle norme ou un nouveau standard mais offre une méthodologie et un cadre opérationnel pour permettre la vérification de la mise en œuvre des standards internationaux d’accessibilité.

Règles de base du WCAG

- Percevable :
  - Fournir un texte alternatif pour tout élément non textuel
  - Fournir des alternatives pour les éléments multimédias
  - Présenter chaque contenu de différentes manières accessibles à tous, sans perdre le sens
  - Faciliter l’écoute et la lecture du contenu pour tous
- Utilisable
  - Permettre l’utilisation du clavier pour toute fonctionnalité
  - Laisser le temps aux utilisateurs de lire et utiliser le contenu
  - Ne pas créer de contenu pouvant créer un malaise ou l'épilepsie
  - Aider les utilisateurs à naviguer et trouver le contenu et se situer sur le site
- Compréhensible
  - Rendre le texte lisible et compréhensible
  - Rendre l’apparition du contenu et le déroulement de la visite prévisible
  - Aider les utilisateurs à éviter et corriger les erreurs
- Robuste
  - Optimiser le contenu pour le rendre compatible avec les outils existants, et futurs

Concevoir une maquette accessible

Concevoir une maquette accessible

1. Structure générale
2. Titre de la page
3. Langues
4. Grammaire HTML et sémantique
5. Liens et boutons
6. Images et icônes
7. Formulaires
8. Listes
9. Tableaux
10. Usage des CSS
11. Zoom et taille des textes
12. Navigation au clavier

1. Structure générale

- Structurer la zone d'entête principale avec `<header>` ou `<div role="banner">`
- Identifier le moteur de recherche avec `<form role="search">`
- Structurer la zone de contenu principal avec `<main>` ou `<div role="main">`
- Structurer les informations relatives au site avec `<footer>` ou `<div role="contentinfo">`
- Structurer les menus de navigation principaux et secondaires avec `<nav>` ou `<div role="navigation">`
- Structurer les menus de navigation avec des listes
- Mettre en place une hiérarchie de titres logique et exhaustive avec les balises `<h1>` à `<h6>`

2. Titre de la page

Renseigner un `<title>` précis sur chaque page

3. Langues

- Renseigner la langue principale de la page avec l'attribut `lang` sur la balise `<html>`
- Utiliser l'attribut `lang` pour signaler les changements de langue

4. Grammaire HTML et sémantique

- Écrire un code HTML valide selon les règles de grammaire du DOCTYPE utilisé
- Employer les balises HTML pour leur valeur sémantique

5. Liens et boutons

- Différencier les boutons des liens
- Compléter les liens et les boutons non explicites avec `aria-label` ou `title`

(!) Ne pas utiliser les attributs `aria-label` et `title` sur des liens ou boutons explicites

6. Images et icônes

- Gérer l'attribut `alt` des balises `<img />` & `<input type="image" />`
- Gérer l'alternative des SVG (images vectorielles)
- Gérer l'alternative des polices d'icônes (Icon Fonts)
- Baliser les images légendées avec `<figure role="figure">` & `<figcaption>`

7. Formulaires

- Utiliser la balise `<label>` ainsi que les attributs `for` & `id` pour étiqueter les champs avec intitulé visible
- Utiliser `title` pour étiqueter les champs sans intitulé visible
- Utiliser l'attribut `autocomplete` pour faciliter le remplissage automatique des champs
- Intégrer les aides à la saisie directement dans les balises `<label>`
- Intégrer `required` ou `aria-required="true"` dans les champs obligatoires
- Intégrer les messages d'erreurs et les suggestions de correction directement dans les balises `<label>`
- Regrouper et titrer les champs de même nature avec `<fieldset>` & `<legend>`

8. Listes

- Baliser les listes non ordonnées avec `<ul>` & `<li>`
- Baliser les listes ordonnées avec `<ol>` & `<li>`
- Baliser les listes de descriptions avec `<dl>`, `<dt>` & `<dd>`

9. Tableaux

- Baliser le titre des tableaux de données avec la balise `<caption>`
- Baliser chaque cellule d'entête de ligne et de colonne avec `<th>`
- Utiliser l'attribut `scope` pour associer les cellules aux entêtes des tableaux de données simples
- Intégrer `role="presentation"` dans chaque balise `<table>` de mise en forme

10. Usage des CSS

- Utiliser CSS pour mettre en forme les textes
- Garantir la lisibilité des contenus lorsque les images ne sont pas affichées
- Garantir la compréhension de l'information même lorsque CSS est désactivé

11. Zoom et taille des textes

- Utiliser uniquement des tailles relatives (rem, em, %, etc.) pour les polices de caractères
- Garantir la lisibilité des contenus même lorsque la taille du texte est doublée

12. Navigation au clavier

- Garantir le fonctionnement de l'interface à la souris et au clavier
- Veiller à ce que l'ordre de tabulation suive la logique de l'ordre de lecture
- Garantir la visibilité de la prise de focus au clavier
- Mettre en place un lien d'évitement

WAI - ARIA

ARIA

ARIA signifie "Accessible Rich Internet Applications".

Tout comme les règles d'accessibilité pour les contenus web, ARIA est une spécification du W3C. Elle définit une série d'attributs qui peuvent améliorer la sémantique d'une application web.

Ces attributs n'ont pas d'influence sur l'aspect visuel et ils n'influencent pas la manière dont les navigateurs traitent les éléments. Leur seul rôle est de fournir de l'information supplémentaire pour les logiciels d'assistance avec lesquels les personnes en situation de handicap utilisent leur ordinateur (surtout les lecteurs d'écran).

- WAI-ARIA Authoring Practices 1.1
- ARIA in HTML

Exemples de ce que les attributs ARIA peuvent faire

- améliorer l'accessibilité de widgets (comme les onglets) pour lesquels il n'existe pas de balises HTML spécifiques.
- communiquer un statut qui est visuellement très clair mais qui n'est pas annoncé par un lecteur d'écran, par exemple si un sous-menu est déplié ou pas.
- faire annoncer à un lecteur d'écran qu'une information est apparue ou a été mise à jour ailleurs sur la même page.
- nommer plus clairement des éléments d'une page (ou application) web pour les personnes aveugles qui n'ont pas la vue d'ensemble de la page.
- ajouter de l'information sémantique qui n'existe pas (encore) en HTML. Par exemple ajouter un `role="main"` au `div` qui englobe le contenu principal de votre page.

L'accessibility tree

L'accessibility tree est une structure, construite sur base du code source et des scripts, comme le DOM, mais qui

- ne contient pas les information inutiles pour l'accessibilité (le lecteur d'écran n'a pas besoin de savoir qu'il y a 15 div imbriqués)
- associe à chaque élément un rôle, un nom et un état.

Pour les éléments HTML standards (img, button, h1,...) ces informations sont disponibles par défaut. Par exemple pour le <button>Search</button>, les informations remplies dans l'accessibility tree sont:

- Rôle: button,
- Nom: Search,
- Statut: focusable, unselected.

Sur base de ces informations, l'internaute sait de quelle manière il peut interagir avec le contenu.

(!) Les attributs ARIA peuvent servir à compléter l'information de l'accessibility tree, pour rendre les interfaces riches compréhensibles et utilisables avec un lecteur d'écran.

Utilisez le plus possible des éléments HTML standards

S'il existe un élément HTML qui correspond à ce dont vous avez besoin, utilisez-le !!!

Quand on utilise des éléments HTML standards, les navigateurs implémentent le comportement standard correspondant et transmettent toute l'information aux lecteurs d'écrans et autres technologies d'assistance.

Deux manières de programmer un bouton

`<button>Pause</button>`

- Les navigateurs, technologies d'assistance et moteurs de recherche l'identifient comme un bouton
- Le lecteur d'écran dit "Bouton pause"
- Même sans CSS on reconnait un bouton; il est clair visuellement qu'il est cliquable
- On peut l'atteindre avec le TAB
- On peut l'activer avec ENTER et ESPACE
- Outline visible (focus)

`<div onclick="..." class="boutonpause">Pause</div>`

- Au niveau sémantique ce n'est pas un bouton (donc pas non plus dans l'accessibility tree)
- Le lecteur d'écran dit "Pause"
- Sans CSS on ne reconnait pas un bouton; même pour onMouseOver il n'y a pas d'indication visuelle que c'est cliquable
- On ne peut pas l'atteindre avec le TAB
- Pas activable au clavier (car on ne peut l'atteindre)
- Pas d'outline

L'attribut rôle

L'attribut rôle peut être ajouté à tout élément HTML5. Il faut tenir compte des règles suivantes:

- Si le composant n'a pas encore de rôle dans l'accessibility tree, l'attribut `role` va en ajouter un.
- Si le composant a déjà un rôle dans l'accessibility tree, alors l'attribut `role` va écraser ce rôle existant.
- C'est la seule chose que fait l'attribut `role`. Il ne modifie ou n'ajoute pas de comportement, le statut ou les propriétés.

Si vous ajoutez un rôle ARIA, vous écrasez la sémantique native de l'élément. Il ne faut donc en principe utiliser cet attribut que pour des éléments vides de sémantique, comme un `div`.

L'attribut rôle

Mauvais exemple:

```
<nav>
    <ul role="navigation">
        ...
    </ul>
</nav>
```

Bon exemple:

```
<nav role="navigation">
    <ul>
        ...
    </ul>
</nav>
```

La sémantique de l'élément ul (liste) est remplacée par `role="navigation"`. Dans l'accessibility tree il n'y a donc plus de liste. Le lecteur d'écran va bien annoncer qu'un bloc de navigation commence ici, mais n'indiquera pas qu'il s'agit d'une liste de x éléments.

(!) Il faut être extrêmement prudent en ajoutant des rôles ARIA car un élément ne peut avoir qu'un rôle. Si vous utilisez l'attribut role, il a priorité, quel que soit l'élément sur lequel vous l'ajoutez. ARIA l'emporte!

Aria

Certains rôles correspondent sémantiquement à des éléments natifs en HTML, comme par exemple `role="button"` qui est équivalent à `<button>`, ou `role="checkbox"` qui est équivalent à `<input type="checkbox" />`. D'autres rôles n'ont pas d'équivalent sémantique en HTML, comme par exemple `role="tablist"`.

Dans l'accessibility tree, les roles ARIA n'influencent que le rôle des éléments. Il existe d'autres attributs ARIA qui permettent de modifier le nom, le statut et les propriétés d'un élément. Le nom de ces attributs commencent par **"aria-"**.

Les éléments qui se trouvent dans l'accessibility tree ont besoin d'un nom. Dans le cas d'éléments HTML le nom est automatiquement rempli. Par exemple, pour une image c'est l'attribut alt qui sert de nom. Le nom d'un lien est le texte du lien. Pour un bouton d'envoi, c'est le contenu de l'attribut value qui sert de nom au bouton.

Aria

Il y a deux attributs aria qui permettent de donner un nom à un élément ou de modifier le nom existant: `aria-label` et `aria-labelledby`. La valeur de `aria-label` doit être un texte, qui deviendra le nom de l'élément; `aria-labelledby` prend la valeur de l'id d'un élément qui contient le texte qui doit servir à identifier l'élément.

Exemple: `<a href="/en" aria-label="Anglais">EN</a>`

`aria-describedby` va rajouter au nom accessible de l'élément, une description accessible, plus longue ou plus complète, logiquement, que le nom accessible.

```
<h3 id="titre">Code de la page de formulaire de connexion<h3>
<button id="code" aria-describedby="code titre">Accéder au code HTML</buttton>
```

Sortie pour un lecteur d'écran : "Accéder au code HTML code de la page de formulaire de connexion"

Quand utiliser ARIA?

- N'utilisez ARIA que lorsqu'il manque des informations dans l'accessibility tree ou lorsque l'information qui s'y trouve est incorrecte.

- Si vous construisez vous-même un widget qu'il n'est pas possible de rendre totalement accessible en utilisant uniquement HTML et CSS, vous pouvez utiliser ARIA. Suivez toujours les motifs de conception (Design Patterns) décrits dans les WAI-ARIA Best Practices. Ils décrivent le comportement de chaque type de widget. Vous y trouverez le comportement au clavier que vous devrez implémenter ainsi que les attributs ARIA à utiliser.

/!\ Si vous utilisez des attributs ARIA, vous devez le faire de manière informée. Un mauvais usage peut avoir des effets néfastes.

HTML5 & ARIA

Parmi tous les avantages que l'on peut lister pour les balises sémentiques HTML5, nous pouvons citer l'apport d'une meilleur accessibilité.

Les navigateurs vont rajouter automatiquement des rôles ARIA sur certaines balises.

Le site html5accessibility présente le support actuel des balises html5 par navigateur.

Mise en forme css et widgets Javascript

Mise en forme css et widgets Javascript

- lien d'évitement
- changer la taille de la police
- changer la couleur de l'interface
- lien "haut de la page"
- fil d'arianne
- navigation
- définir les éléments avec le focus
- Masquer du contenu
- DOM order
- Tabindex
- Modale

Lien d'évitement

Ce lien doit être le premier élément interactif dans le code HTML.

Il s'agit d'un lien interne qui doit permettre un accès direct au contenu principal de la page.

HTML

```
<body>
  <a class="evitement" href="#contenu">Aller au contenu</a>
  […]
  <main role="main">
     <a id="contenu" tabindex="-1"></a>
     […]
  </main>
  […]
</body>
```

Lien d'évitement

CSS

```
a.evitement {
   display: inline-block;
   color: #555;
   background: #fff;
   padding: .5em;
   position: absolute;
   left: -99999rem;
   z-index: 100;
}
a.evitement:focus {
   left: 0;
}
```

Masquer du contenu

Invisible pour tous

Certaines pages web contiennent des éléments cachés qui deviennent visibles après une action de l'utilisateur.

Exemples:

- Les diapositives inactives d'un diaporama et les onglets cachés d'un ensemble d'onglets.
- Les réponses d'un module de FAQ qui ne sont pas visibles.
- Le sous-menu d'un menu déroulant.
- Le contenu destiné à apparaître dans une lightbox.
- Le lien qui fera apparaître le menu sur les sites 'responsive'. Ce lien n'apparaît que sur les petits écrans. (mobiles, tablettes...)

Pour masquer ces éléments on utilisera la propriété css `display: none`

Invisible pour tous

En utilisant `display: none` le contenu:

- n'apparaîtra pas dans le navigateur,
- ne sera pas lu par un lecteur d'écran,
- ne sera pas repris dans l'ordre de tabulation.

Par contre les éléments cachés de cette manière pourront:

- être lus dans le code source,
- apparaître lorsque le CSS est désactivé ou mal supporté,
- être lus par d'anciennes versions de lecteurs d'écran qui ne prennent pas en compte le CSS,
- être indexés par les moteurs de recherche.

N'utilisez donc cette technique que pour du contenu caché temporairement destiné à être affiché sous certaines conditions. Sinon effacez-le du code source ou placez-le en commentaire.

Texte invisible mais lu par un lecteur d'écran

Dans certains cas vous pourriez souhaiter ajouter du texte qui ne doit pas être visible à l'écran mais qui aide une personne aveugle à naviguer dans la page ou à comprendre la structure/le contenu plus facilement.

Exemple:

- texte de remplacement pour les images d'arrière-plan.
- alternative textuelle pour les polices d'icônes (icon fonts) significatifs,
- alternative à l'information visuelle tel que le poids d'une étiquette dans un nuage de tags.

En assignant 0 à la propriété `clip` de CSS, on rend l'élément invisible, sauf pour les lecteurs d'écran.

Texte invisible mais lu par un lecteur d'écran

Parmi d'autres, HTML5 Boilerplate utilise ceci:

```
.sr-only {
  position: absolute;
  width: 1px;
  height: 1px;
  padding: 0;
  margin: -1px;
  overflow: hidden;
  clip-path: rect(0, 0, 0, 0);
  white-space: nowrap;
  border: 0;
}
```

Une autre technique consiste à positionner le texte hors de la portion visible de l'écran. Celui-ci devient invisible pour tous sauf pour les lecteurs d'écran.

```
.hidden {
    position: absolute;
    left: 0;
    top: -10000px;
    overflow: hidden;
}
```

Caché seulement pour les lecteurs d'écran

Rendre invisibles uniquement des éléments pour les lecteurs d'écran qui sont affichés par le navigateur. Par exemple pour un élément qui sert davantage pour la mise en forme et qu'il vaut mieux ne pas lire. Ou afin d'éviter que le lecteur d'écran ne lise des informations redondantes ou inutiles.

Dans ce cas, on peut utiliser l'attribut `aria-hidden="true"` sur un élément qui sera ignoré par les lecteurs d'écrans mais reste visible pour les autres visiteurs.

/!\ l'ajout de `aria-hidden="true"` à un élément, s'applique également aux éléments imbriqués. Il conviendra dès lors de l'utiliser avec la plus grande prudence et tester avec un lecteur d'écran.

Focus

Le focus détermine où les événements de clavier vont dans la page à un moment donné et met en évidence l'élément sélectionné.

- Liens
- Champs de formulaire
- Boutons

DOM order

W3C

L'objectif de cette technique est de garantir que l'ordre du contenu dans le code source est le même que la présentation visuelle du contenu. L'ordre du contenu dans le code source peut être modifié par l'auteur par différent moyen de présentations visuelles avec CSS. Chaque commande peut avoir un sens en soi, mais peut créer de la confusion pour les utilisateurs de technologies d'assistance.

Exemple: Google développeur

Tabindex

Developper Mozilla

L'attribut universel tabindex est un entier indiquant si l'élément peut capturer le focus et si c'est le cas, dans quel ordre il le capture lors de la navigation au clavier (généralement à l'aide de la touche Tab). Si plusieurs éléments partagent la même valeur d'attribut tabindex, leur ordre sera calculé en fonction de leur position dans le document.

```
<label>First in tab order:<input type="text"></label>
<div tabindex="1">Tabbable due to tabindex.</div>
<div>Not tabbable: no tabindex.</div>
<label>Third in tab order:<input type="text"></label>
```

Tabindex: Améliorer l'accessibilité au clavier

ARIA permet de contourner ce problème, à l'aide de la propriété tabindex. ARIA permet d'ajouter cette propriété à n'importe quel élément HTML. La valeur tabindex="0" place l'élément dans l'ordre de tabulation normal du document, c'est-a-dire que le navigateur s'arrêtera sur cet élément lorsque l'utilisateur l'atteindra en tabulant. Ce mécanisme permet d'ajouter des fonctionnalités et de l'interactivité à cet élément, par exemple en déclenchant un évènement au moment où l'élément reçoit le focus, ou en fonction d'une touche enfoncée alors que l'élément possède le focus.

La valeur tabindex="-1" permet à un élément de recevoir le focus mais uniquement par programmation. Il faut donc que l'utilisateur active un lien pointant vers cet élément (<a href="/traduction/accessibilite-des-applications-internet-riches#contenu_principal">...) ou que le focus soit déclenché par Javascript (par exemple avec document.getElementById('messageErreur').focus();).

Tabindex: Améliorer l'accessibilité au clavier

En augmentant le nombre d'éléments pouvant recevoir le focus dans le navigateur, ARIA augmente les possibilités de rendre les scripts accessibles au clavier.

Les autres composants (widget)

Access & use

- Modales
- Boites de dialogues (non modale)
- Menus dépliants
- Onglets
- Caroussels
- Les contenus masqués
- Contenus au survol ou focus
- Formulaires et gestion des erreurs

Comment faire et vérifier le bon choix des couleurs

Plusieurs critère à prendre en compte:

- taille (font-size)
- type (font-family)
- graisse (font-weight)

Outils de test de contraste

- Web AIM - Contrast checker
- contrastchecker.com
- Color Safe : Donne une liste de couleur de texte ayant un contrast fort avec le background choisi.
- Who Can Use

Quelle police utiliser et comment définir la taille

Quelle police utiliser

L’efficacité des polices pour les dyslexiques est encore incertaine ...

L’usage d’une police de caractères appropriée peut atténuer les effets de la dyslexie, un trouble spécifique de la lecture qui touche près de 10 % de la population mondiale.

Il est recommandé d'utiliser des polices sans empatement (sans-serif), afin de faciliter la compréhension des textes.

Comment définir la taille

Un principe d'accessibilité est de garantir la lisibilité des contenus même lorsque la taille du texte est doublée.

En revanche, l'accessibilité n'impose pas de taille de caractères minimum. Autrement dit, il est tout à fait possible d'utiliser une taille de police de caractères minuscule tout en étant conforme aux WCAG dès lors que l'unité utilisée est relative (rem, em, %, small, etc.).

(!) La majorité des navigateurs utilisent une taille de police par défaut de 16px

Les critères essentielles seront:

- l'espace entre les lignes
- l'espace entre les lettres
- l'alignement des textes (gauche, ~~centre~~, droite, ~~justifié~~)
- taille relative

Taille des polices

Source

Points	Pixels	Ems	Percent
6pt	8px	0.5em	50%
7pt	9px	0.55em	55%
7.5pt	10px	0.625em	62.5%
8pt	11px	0.7em	70%
9pt	12px	0.75em	75%
10pt	13px	0.8em	80%
10.5pt	14px	0.875em	87.5%
11pt	15px	0.95em	95%
12pt	16px	1em	100%
13pt	17px	1.05em	105%
13.5pt	18px	1.125em	112.5%
14pt	19px	1.2em	120%
14.5pt	20px	1.25em	125%
15pt	21px	1.3em	130%

Points	Pixels	Ems	Percent
16pt	22px	1.4em	140%
17pt	23px	1.45em	145%
18pt	24px	1.5em	150%
20pt	26px	1.6em	160%
22pt	29px	1.8em	180%
24pt	32px	2em	200%
26pt	35px	2.2em	220%
27pt	36px	2.25em	225%
28pt	37px	2.3em	230%
29pt	38px	2.35em	235%
30pt	40px	2.45em	245%
32pt	42px	2.55em	255%
34pt	45px	2.75em	275%
36pt	48px	3em	300%

Les CAPTCHAs

Pour que le CAPTCHA reste efficace vis-à-vis des bots, il est donc nécessaire d'accentuer la déformation de l'image ou du son proposé par le CAPTCHA : cela accentue d'autant les difficultés pour l'utilisateur.

Google tente de trouver des alternatives et propose le "No Captcha" (reCaPTCHA), une simple case à cocher du point de vue de l'utilisateur.

Bien que cette solution soit plus efficace, elle n'est pas aujourd'hui satisfaisante en termes d'accessibilité car en cas de doute, un CAPTCHA standard est affiché à l'écran. C'est souvent le cas pour un utilisateur qui ne peut pas utiliser une souris et qui navigue au clavier ou pour un utilisateur qui navigue à l'aide d'un lecteur d'écran (cas des personnes malvoyantes).

Les CAPTCHAs

Alternatives

- Double authentification
- Champ de formulaire caché à laisser vide (technique du honeypot), non-visibles pour l'utilisateur
- Mise à disposition d'un support téléphonique afin de s'assurer que le client est une vraie personne
- Un contrôle permettant de s'assurer qu'une même combinaison IP/User agent (navigateur) ne tente pas de soumettre le formulaire plus de x fois par seconde
- Une alternative sonore dans le cas d'un CAPTCHA visuel
- Une alternative visuelle dans le cas d'un CAPTCHA sonore
- Des tests logiques (question dont la réponse est évidente, test mathématique simple…) + captcha visuel classique
- Un test sur le temps de remplissage du formulaire
- Un système de filtrage antispam côté serveur

Valider les formulaires

Lors de la validation, si des champs obligatoires ne sont pas renseignés, ou si le format de la donnée saisie n'est pas valide, il faut prévenir l'utilisateur.

- `aria-invalid` pour indiquer une erreur de saisie
- Annoncer que le formulaire est incomplet ou qu'il existe des erreurs: on utilise le rôle ARIA `alert` pour notifier les technologies d'assistances du message lors de la soumission du formulaire.
- Avertir l'utilisateur au niveau du champ en cas d'erreur de saisie
- Afficher des messages d'erreur explicites et, si besoin, suggérer des corrections

Frameworks

https://darekkay.com/blog/accessible-ui-frameworks/

Ajax

AJAX ne fonctionne pas dans tous les navigateurs. Les conseils d'accessibilité pour le contenu Web (WCAG) stipulent qu'une application doit fonctionner même si Javascript est désactivé ou non supporté. De plus, AJAX nécessite XMLHttpRequest, qui n'est pas disponible dans tous les navigateurs.

Actuellement, la solution à ces problèmes consiste à fournir une alternative entièrement non-AJAX, ou à faire en sorte que l'application fonctionne également sans Javascript et XMLHttpRequest.

Le fonctionnement typique d'AJAX consiste à mettre à jour et manipuler des éléments de l'interface à la volée.

La modification d'une zone peut ne pas être remarquée, ce problème est d'autant plus ennuyeux pour les utilisateurs de lecteur d'écran que ces dispositifs parcourent la page de manière linéaire. Si un changement intervient dans l'interface, rien ne garantit que l'utilisateur en sera informé et que le contenu modifié sera lu.

Ajax: Mise à jour de contenu dynamique

Avec la technologie WAI-ARIA, les développeurs peuvent identifier des zones de la page comme étant "actives" ("live region" dans la terminologie ARIA), dans lesquelles la mise à jour s'effectuera d'une manière compréhensible pour un lecteur d'écran.

Pour créer une région active, le développeur doit ajouter l'attribut aria-live en lui donnant une valeur parmi "off" (désactivé), "polite" (poli), "assertive" (autoritaire), ~~et "rude" (malpoli).~~

Pour chaque valeur le lecteur d'écran aura un comportement différent:

- off: ignorera la mise à jour
- polite: l'utilisateur est informé du changement dès que sa tâche actuelle est terminée
- assertive: l'utilisateur est alerté immédiatement ou le plus tôt possible
- ~~rude: réserver aux modifications critiques, et informe l'utilisateur instantanément, pouvant même placer d'autorité le curseur au début du contenu modifié~~

Tester l'accessibilité de son application

Pa11y met à disposition plusieurs outils gratuit pour permettre de tester l'accessibilité d'application web.

- Pa11y: Une interface de ligne de commande qui charge les pages Web et met en évidence les problèmes d'accessibilité détectés.
- Pa11y CI: Un outil de ligne de commande qui itère sur une liste de pages Web et met en évidence les problèmes d'accessibilité. Il s'agit d'une CLI plus orientée vers l'utilisation en CI.
- Koa11y: est une application de bureau pour Windows, OSX et Ubuntu qui vous permet de commencer à utiliser Pa11y en quelques clics. Parfait si vous n'êtes pas développeur et que vous avez juste besoin d'un moyen simple d'utiliser Pa11y à partir de votre ordinateur de bureau sans vous soucier du code source et de la configuration compliquée!

/!\ Ces outils sont capables de tester plusieurs critères techniques de l'accessibilité (ratio de couleurs, attributs html, etc), mais ne sont à même de juger de la pertinence du contenu que vous y placez !

Exemple: `<img src="path/to/logo.png" alt="hello world" title="logo de mon site" />`

Bibliographie

- http://colorsafe.co/
- https://contrastchecker.com/
- https://webaim.org/resources/contrastchecker/
- https://blog.atalan.fr/taille-de-police-advisorytechnique/
- https://www.accede-web.com/notices/html-et-css/
- http://www.pompage.net/traduction/ameliorer-l-accessibilite-par-la-typographie
- https://a11y-guidelines.orange.com/web/exemples/formulaire/index.html
- https://www.w3.org/TR/WCAG20-TECHS/C27.html
- https://a11yproject.com/
- https://squizlabs.github.io/HTML_CodeSniffer/
- https://pa11y.org/

Bibliographie

- https://www.anysurfer.be/fr/en-pratique/sites-web/cacher-du-texte
- https://www.alsacreations.com/article/lire/1203-etat-des-lieux-accessibilite-html5.html
- https://developer.mozilla.org/fr/docs/Accessibilit%C3%A9/ARIA/Techniques_ARIA
- https://a11y-guidelines.orange.com/web/label-ledby-describedby.html
- http://www.pompage.net/traduction/accessibilite-des-applications-ajax-implications
- http://www.pompage.net/traduction/accessibilite-des-applications-internet-riches
- https://a11y-guidelines.orange.com/web/index.html
- https://a11y-guidelines.orange.com/web/exemples.html
- https://a11y-guidelines.orange.com/web/aria-live-alert.html