Rapport de stage

« Création d'un lecteur multimédia accessible en HTML5 »

IUT Orléans
Département informatique
Année universitaire 2014/2015

Stagiaire : Adrien Fallot
Maître de stage : M. Casson
Tuteur : Mme Brossier

Remerciements

Je tiens tout d'abord à remercier M. Casson pour m'avoir proposé un stage au sein de la passerelle handicap de l'université d'Orléans, ainsi que l'ensemble de son personnel pour m'avoir accueilli tout au long de ces dix semaines.

J'aimerais également remercier, ma tutrice de stage au sein de l'IUT, Mme Brossier pour avoir été disponible très tôt et nous avoir aidés lorsque nous avions des questions administratives à poser à l'IUT.

Et enfin, je tiens aussi à remercier tout le personnel de l'IUT, mais également de l’université pour avoir été présent et aidant tout le long ces deux années de formation.

Table des matières

Introduction

Dans le cadre de la deuxième année de formation du DUT informatique à l’université d'Orléans, nous avons la possibilité d'effectuer un stage de dix semaines. J'ai donc choisi la passerelle handicap de l’université pour l'y effectuer.

Ce stage m'a beaucoup appris sur l’accessibilité dans le domaine informatique et plus précisément sur le web. J'y ai, entre autres, appris certaines règles de présentation pour les personnes atteintes de troubles « dys ». Pour expliquer ce que l'on entend par ce terme, je citerais la Fédération française des Dys avec la phrase suivante « On regroupe sous « troubles Dys » les troubles cognitifs spécifiques et les troubles des apprentissages qu’ils induisent. », certaines personnes ne peuvent pas enregistrer les règles d’orthographes comme la plupart des gens, ils sont dysorthographiques. Mais il existe aussi la dyspraxie (pour l'automatisation des gestes) , la dyscalculie (pour le calcul), la dysgraphie (pour l'écriture), la dyslexie (pour la lecture), etc... Pour certaines personnes atteintes de ces troubles il faut respecter certaines règles notamment : ne pas mettre le texte en justifié, par contre il faut placer un interligne de 1,5 et de préférence utiliser une police spécifique et enfin placer deux espaces entre chaque mot au lieu d'un seul. Comme j'ai fait un stage pour l’accessibilité, je suis donc resté dans cette optique lors mon rapport, c'est pour cela que ces quelques règles y sont respectées.

1. Lieu de stage

Comme indiqué précédemment dans ce rapport, j'ai réalisé mon stage à la « passerelle handicap » de l'université d'Orléans. Elle est située juste à côté du centre culturel « Le Bouillon » et dans le même bâtiment que le restaurant universitaire « Le Forum ». Ceci la place au cœur d'un des lieux clés du campus, ce qui est très important au vu de ses différentes missions.

Figure 1 : La passerelle handicap vue de l'extérieur

En effet elle se charge d'accompagner les étudiants en situation de handicap – que ce dernier soit permanent ou temporaire – tout au long de leur parcours universitaire. Elle s'occupe d'eux aussi bien durant leurs études, à proprement dit, en mettant en place différentes aides aux études telles que des preneurs de notes, des interprètes, mais également des aménagements lors des examens comme par exemple une majoration de temps. Elle s'occupe également de mettre à disposition du matériel spécialisé ou des ordinateurs, elle peut également adapter les différents documents pédagogiques en faisant par exemple une transcription braille, un agrandissement, etc... Ce service s'occupe également d'accompagner les étudiants durant leur vie universitaire. Elle les aide par exemple lors des démarches administratives, mais également à trouver un logement adapté à proximité ou encore

dans leur vie de tous les jours – lors des déplacements ou des repas par exemple. Elle promeut également des rencontres entre étudiants ainsi qu'un accès aux activités physiques et culturelles présentes sur le campus.

La « passerelle » bénéficie aussi d'un lieu de travail dans lequel les étudiants peuvent venir quand ils le souhaitent. Il y a dans ce lieu un ordinateur qui fonctionne sous Windows avec un lecteur d'écran libre : NVDA. C'est un logiciel destiné aux personnes ne pouvant pas lire, il retranscrit par synthèse vocale et/ou sur un afficheur braille ce qui est affiché sur l'écran d'un ordinateur. Il permet également d'interagir avec lui à l'aide de raccourcis clavier. Tous ces outils permettent aux étudiants handicapés de travailler dans les meilleures conditions possibles. Il possède également un logiciel d'agrandissement ZoomText qui permet de zoomer la totalité de l'écran, inverser les couleurs… Un autre ordinateur sous GNU/Linux offre également ces services.

Mais le service possède toute une batterie de matériels spécialisés qui m'ont permis d'en apprendre plus sur l'accessibilité en général. Voici un petit échantillon du matériel disponible :

- un téléagrandisseur **:** c'est une machine qui permet d'afficher une feuille de papier de manière agrandie et même changer le contraste

Figure 2 : Un téléagrandisseur

- un clavier ZoomText et le logiciel qui lui est associé : c'est un logiciel qui permet de pratiquer des zooms sur un écran d'ordinateur, ce qui est très pratique pour les personnes mal-voyantes qui ont besoin d'un l'affichage plus grand. Le

clavier quant à lui est beaucoup plus grand et est muni d'une série de boutons permettant par exemple un zoom avant, arrière, ou un retour à 100 %.

- des plages brailles **:** ce sont des périphériques qui lorsqu'ils sont reliés à un lecteur d'écran permettent d'afficher en langage braille un texte « lu » par celui-ci.

Figure 3 : Une plage braille

J'ai effectué mon stage dans cette pièce avec mon ordinateur portable personnel ainsi que d'un deuxième écran mis à disposition par la « passerelle ». Par la suite, nous y avons ajouté un MacBook pro 15" avec le lecteur d'écran VoiceOver ainsi qu'un ordinateur fonctionnant sous Ubuntu avec le lecteur d'écran Orca. Ceci dans le but d'effectuer des tests dépendants du système d'exploitation et du navigateur web, mais également du lecteur d'écran utilisé.

2. Présentation du sujet

2.1. Origines

Aujourd'hui internet est énormément répandu, mais malheureusement celui-ci n'est pas accessible pour tout le monde alors qu'il suffirait de quelques petites modifications sur la page d'un site web pour la rendre possible d'accès pour tous. C'est pour cela que M. Casson, référent en accessibilité numérique à la passerelle handicap, a eu pour idée la création d'un lecteur multimédia HTML5 utilisable très facilement au clavier pour les personnes se servant peu ou pas de la souris, tels que par exemple les malvoyants.

2.1. Spécificités

Ce lecteur devra avoir un retour vocal lorsque l'utilisateur – possédant, bien entendu, un lecteur d'écran – y effectue une action. Il devra également respecter les critères d'accessibilité définis par le Référentiel Général d'Accessibilité pour les Administrations (RGAA 3.0) tel que le contraste, des descriptions pertinentes des liens et respecter l'ordre des titres. Il devra également être le plus simple possible à intégrer dans un site web pour que chaque intégrateur puisse s'en servir comme il le souhaite.

Il sera donc développé avec les technologies web suivantes :

- HTML5 : C'est un langage de programmation permettant de créer des pages web. Il permet d'en faire la structure interne. Actuellement en version 5 celle-ci inclut, entre autres, la possibilité d'ajouter des fichiers vidéos et des fichiers audio.
- CSS3 (Cascading Style Sheets) : C'est un langage de programmation permettant de modifier l'apparence des pages web. Actuellement en version 3 celle-ci inclut, entre autre, la possibilité d'ajouter des animations.

- JavaScript : C'est un langage de programmation qui est principalement utilisé pour dynamiser les pages web. Il permet par exemple de faire disparaître ou apparaître des éléments ou encore d'en modifier des attributs. Nous allons également beaucoup utiliser la bibliothèque JQuery qui facilite grandement la plupart des actions JavaScript.

Il sera principalement développé à l'aide de cette dernière autour des balises *audio* et *vidéo* disponibles avec HTML5. Pour mettre à disposition les sources et faciliter la collaboration, le projet sera mis sur GitLab à l'adresse suivante : https://gitlab.com/david.casson/MediaPlayerA11y. GitLab est une plate-forme en ligne qui permet de partager son travail avec d'autres membres de la communauté GitLab – soit tout le monde soit uniquement certaines personnes en fonction des paramètres du projet.

3. Présentation des objectifs

3.1 Sensibilisation et apprentissage de l’accessibilité web

Lors de mon arrivée à la « passerelle », mon maître de stage m'a d'abord présenté le planning ainsi que ses attentes pour le projet. Après nous sommes passés à une sensibilisation sur l'accessibilité web – voulant par la suite, administrer mon propre site web en mettant à disposition des logiciels pour les personnes en situation de handicap elle me sera très utile pour plus tard et pourra me faire un plus par rapport à beaucoup de développeurs qui n'en sont pas forcément dotés. L’accessibilité web est définie selon les quatre grands principes suivant :

- perceptible : Tout le contenu jugé utile doit être disponible sous forme textuelle sans aucune perte d'information, les contrastes doivent être suffisamment visibles pour tout le monde.

- utilisable : Toutes les fonctions doivent être accessibles au clavier, laisser à l'utilisateur assez de temps pour lire le contenu, etc…

- compréhensible : L’organisation des pages doit être facilement compréhensible.

- robuste : Le site doit être compatible avec les **technologies d’assistance** et les navigateurs.

Les différentes caractéristiques de celle-ci sont actuellement émises par le *Web Accessibility Initiative* (WAI), qui est une initiative lancée par le *World Wide Web Consortium* (W3C) et ayant pour but de promouvoir l'accessibilité sur le web. Elle s'occupe par exemple de fixer les limites de contrastes et de référencer toutes les règles qu'il va falloir respecter pour qu'un site web soit accessible. Le W3C quant à lui est un organisme de normalisation chargé de promouvoir la compatibilité des technologies du web dans le monde entier. C'est par exemple lui qui s'occupe de définir HTML5, de dire qu'elle est la syntaxe correcte et les bonnes utilisations à

avoir. Il s'occupe aussi de créer ou de mettre à jour les attributs et les balises HTML5, mais il s'occupe également des formats d'image PNG et SVG par exemple.

Le WAI écrit une spécification, le *Web Content Accessibility Guidelines* (WCAG) – actuellement en version 2.0 – celle-ci contient toutes les règles à respecter pour qu'un site soit certifié accessible – il y a actuellement trois niveaux de certification le A, AA et AAA en fonction du nombre de critères respectés. Avec cette spécification chaque pays fait sa propre version en fonction de ses besoins. En France il y a le RGAA 3.0 et AccessiWeb. Le premier est destiné aux applications web utilisées dans l'administration publique et le second est réservé uniquement au domaine privé.

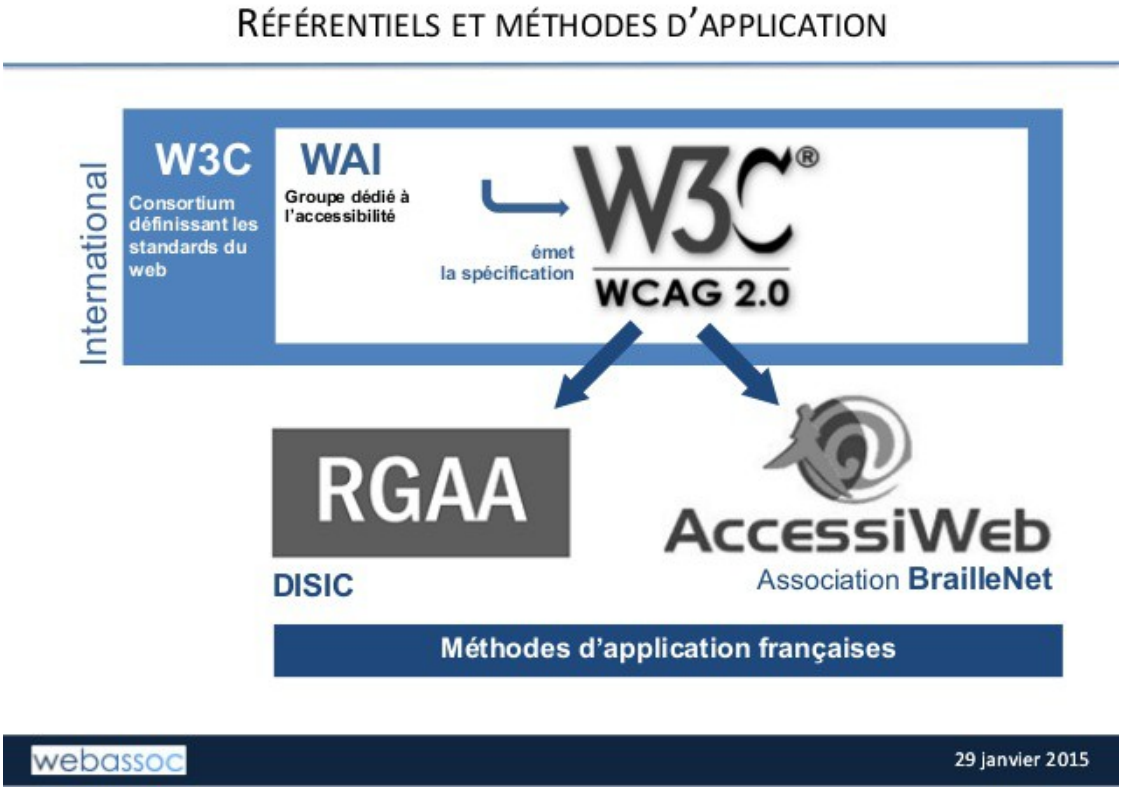


Figure 4 : Schéma récapitulatif des spécifications de l'accessibilité web en France

Comme ce stage est effectué dans le cadre d'un projet universitaire, nous nous baserons sur le RGAA 3.0**,** celui-ci doit respecter tous les critères de niveau A et AA pour être considéré comme valide (niveau de conformité), ce qui représente

102 de 133 critères qui sont répartis sur 13 catégories dont voici quelques exemples avec certains des critères A et AA à respecter :

- Couleur (Perceptible) : « [AA] Dans chaque page Web, le contraste entre la couleur du texte et la couleur de son arrière-plan est-il suffisamment élevé (hors cas particuliers) ? »
- Image (Perceptible) : « [A] Chaque image a-t-elle une alternative textuelle ? »
- Multimédia (perceptible, utilisable, robuste) : « [A] Chaque média temporel pré-enregistré a-t-il, si nécessaire, une transcription textuelle ou une audio-description (hors cas particuliers) ? »
- Structuration de l'information (perceptible, utilisable, compréhensible) : « [A] Dans chaque page Web, l'information est-elle structurée par l'utilisation appropriée de titres ? »

Durant ces premiers jours, je me suis donc renseigné sur le fonctionnement de ce système pour savoir où aller chercher les informations dont j'aurais éventuellement besoin. Mais j'ai également lu la spécification RGAA 3.0 pour connaître tous les critères que j'allais devoir respecter pour mener à bien ce projet.

3.2. Installation, configuration et prise en main de l'environnement de travail.

La deuxième partie de mon stage est constituée de l’installation des différents logiciels qui me permettront de me mettre dans la même situation que les futurs utilisateurs du projet. Autant les visiteurs du site qui pourront utiliser le lecteur en installant une machine virtuelle Windows avec le lecteur d'écran NVDA. Mais également « WAMP serveur » – C'est une plate-forme de développement qui permet de faire fonctionner un serveur sur sa machine sans avoir besoin de passer par Internet – pour me mettre à la place des intégrateurs utiliseront peut-être ce projet sur leurs propres sites web.

Après avoir installé NVDA, j'ai pris un moment pour lire sa notice d'utilisation et pour me mettre en situation, j'ai alors fermé les yeux et j'ai essayé de naviguer sur

internet uniquement à l'aide du retour vocal. J'ai fait cela sur des sites que je connais assez bien mais malgré cela j'ai très vite été perdu car le site n'était pas forcément accessible. Durant ces tests, j'ai compris que beaucoup de personnes ne peuvent pas utiliser correctement internet et que l'accessibilité est quelque chose qui n'est pas spécialement difficile à mettre en place mais qui est très importante.

3.3. Analyse

La partie suivante de mon stage a consisté à faire une analyse détaillée du projet ainsi que des projets similaires qui pouvaient déjà exister. L'étude de l'existant s'est déroulée en deux étapes : la première sans prendre en compte – ou de façon mineure – l'accessibilité de celui-ci, puis la seconde avec NVDA en tenant beaucoup plus compte de celle-ci. Ces analyses m'ont permis de voir au mieux les fonctionnalités à implémenter ainsi que de faciliter l'organisation de mon travail par la suite. J'ai donc fait une liste (voir annexe p 33) des points à améliorer et des points à garder de chacun des projets suivants :

- Access42 : http://access42.net/MFP-lecteur-video-accessible.html
- Vibéo : http://vibeo.libeo.com/
- Acorn : https://ghinda.net/acornmediaplayer/

Nous en avons par exemple conclu qu'il ne faut surtout pas que notre lecteur annonce constamment le temps actuel de la vidéo. Mais au contraire, le fait de pouvoir se déplacer image par image ou encore, le fait d'avoir des raccourcis clavier sont des idées à garder. De cette analyse, il en a résulté plusieurs documents, le premier a été la carte heuristique suivante :

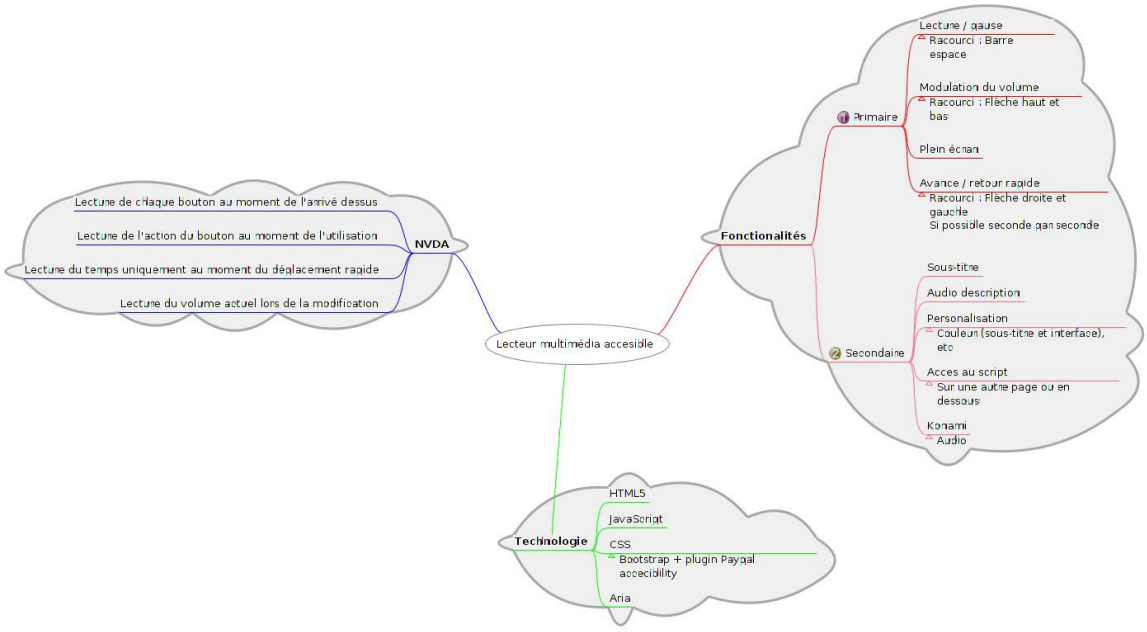


Figure 5 : Carte heuristique du projet.

Elle a été très utile pour lister correctement toutes les fonctionnalités que nous avions retenues et que le lecteur devra par la suite posséder. Les fonctions ont ensuite été classées en deux catégories :

- les principales, dont l'implémentation devra se faire en priorité – car sans elles le lecteur ne pourrait pas fonctionner

- les secondaires, qui sont obligatoires pour répondre correctement à la problématique du sujet – mais qui ne sont pas indispensable pour le bon fonctionnement du lecteur.

Ensuite la première architecture possible du lecteur ainsi que les premières maquettes ont été faites pour visualiser un peu mieux l'organisation de celui-ci :

Figure 6 : Maquette générale du lecteur.

Bien que les contrastes ne soient pas correctement respectés et que toutes les fonctionnalités ne soient pas là, cette première maquette donne une vision globale lecteur qui va pouvoir être suivie lors de la création de celui-ci.

3.4 Programmation du lecteur

JavaScript ayant déjà des fonctions qui permettent de mettre « lecture », « pause », « mute » ou encore de régler le volume sur une vidéo HTML5, l'implémentation des boutons associés à ces fonctions a été très rapide. Il a juste fallu trouver une apparence que le lecteur allait ensuite suivre tout le long du projet. Ayant vu la bibliothèque Bootstrap en cours, j'ai donc décidé de m'en servir pour faire l'apparence du lecteur. Bootstrap est une bibliothèque CSS / HTML et JavaScript qui permet de simplifier grandement la création du style d'une page web. Nous avons donc utilisé les boutons et les icônes disponibles avec celle-ci.

Je suis ensuite passé à la programmation du bouton plein écran, comme on

peut le constater ci-dessous, elle a été un peu plus complexe puisqu'il a fallu gérer les différents types navigateurs. À savoir ceux qui utilisent « web-kit » (Google Chrome, Safari, Opera), « moz » (Mozillia Firefox) et « ms » (Internet Explorer). Même si ce dernier n'est plus maintenu dans le futur, il y aura, quand même, certaines personnes qui l'utiliseront. De plus peut-être que Microsoft utilisera la même technologie pour son nouveau navigateur, il sera donc également compatible avec le lecteur.

```
//  fullscreen

var oldHeight;
var oldWidth;
var fullScreenID=-1;

function enableFullScreen(id) {

    var video = document.getElementById("video_"+id);
    var divVideo = document.getElementById("divVideo_"+id);

    if (!document.fullscreenElement && !document.mozFullScreenElement && !document.webkitFullscreenElement && !document.msFullscreenElement ) {  // current working metho
        oldHeight = video.height;
        oldWidth = video.width;
        if (divVideo.requestFullscreen) { //default
            divVideo.requestFullscreen();
            minToFull(id);
        }
        else if (divVideo.msRequestFullscreen) { //IE
            divVideo.msRequestFullscreen();
            minToFull(id);
        }
        else if (divVideo.mozRequestFullScreen) { //firefox
            divVideo.mozRequestFullScreen();
            minToFull(id);
        }
        else if (divVideo.webkitRequestFullscreen) { //chrome ; safari
            divVideo.webkitRequestFullscreen(Element.ALLOW_KEYBOARD_INPUT);
            minToFull(id);
        }
        $("#fullscreen_"+id).attr("class","elem playerBtn btn-primary glyphicon glyphicon-resize-small");
        $("#fullscreen_"+id).attr("aria-pressed","true");
        fullScreenID = id;
    }
    else {
        if(document.exitFullscreen) {
            fullToMin(fullScreenID);
            document.exitFullscreen();
        }
```

Figure 7 : Fonction plein écran

Après avoir effectué cela, il fallait pouvoir gérer très vite plusieurs lecteurs sur la même page web, avant d'avoir tout à refaire. Cette fonctionnalité a été assez complexe du fait de ne pas avoir de variable associée à tel ou tel lecteur – ne sachant donc pas exactement combien de lecteurs il pourrait y avoir sur la page, il n'y pas par exemple possibilité de stocker le bouton lecture de tel ou tel lecteur. J'ai donc implémenté un système identifiant qui permettrait de retrouver très facilement n'importe quel élément de n'importe quel lecteur.

Ce système d'identifiant fonctionne selon la manière suivante : « nomDeElement_id ». Il est automatiquement généré lors de la création du lecteur : lorsque la page web est chargée une fonction JavaScript cherche toutes les vidéos

de la page (voir numéro 1 sur la capture d'écran si après) et les analyse une à une pour leur associer les différents boutons que les lecteurs devront contenir. Mais elle y ajoute également les identifiants correspondants à ces derniers grâce à une variable (variable *cpt* dans la capture d'écran) qui s'incrémente à chaque nouveau lecteur. Avec cette méthode tous les boutons du premier lecteur s'appelleront donc : « nomDeElement_0 » – par exemple « play_0 » ou encore « mute_0 » (voir numéro 2). De plus, lors de la création d'un bouton l'identifiant du lecteur est placé en paramètre de la fonction JavaScript qui lui est associée (voir numéro 3), ce qui permet par exemple dans la fonction *fullscreen* de retrouver tout le lecteur et de redéfinir la taille de tous ces éléments très facilement.

Après les tests, nous avons pu constater qu'avec cette méthode de génération le programme pouvait créer 1400 lecteurs sur une seule page, au-delà le JavaScript surcharge lors de la recherche des vidéos.

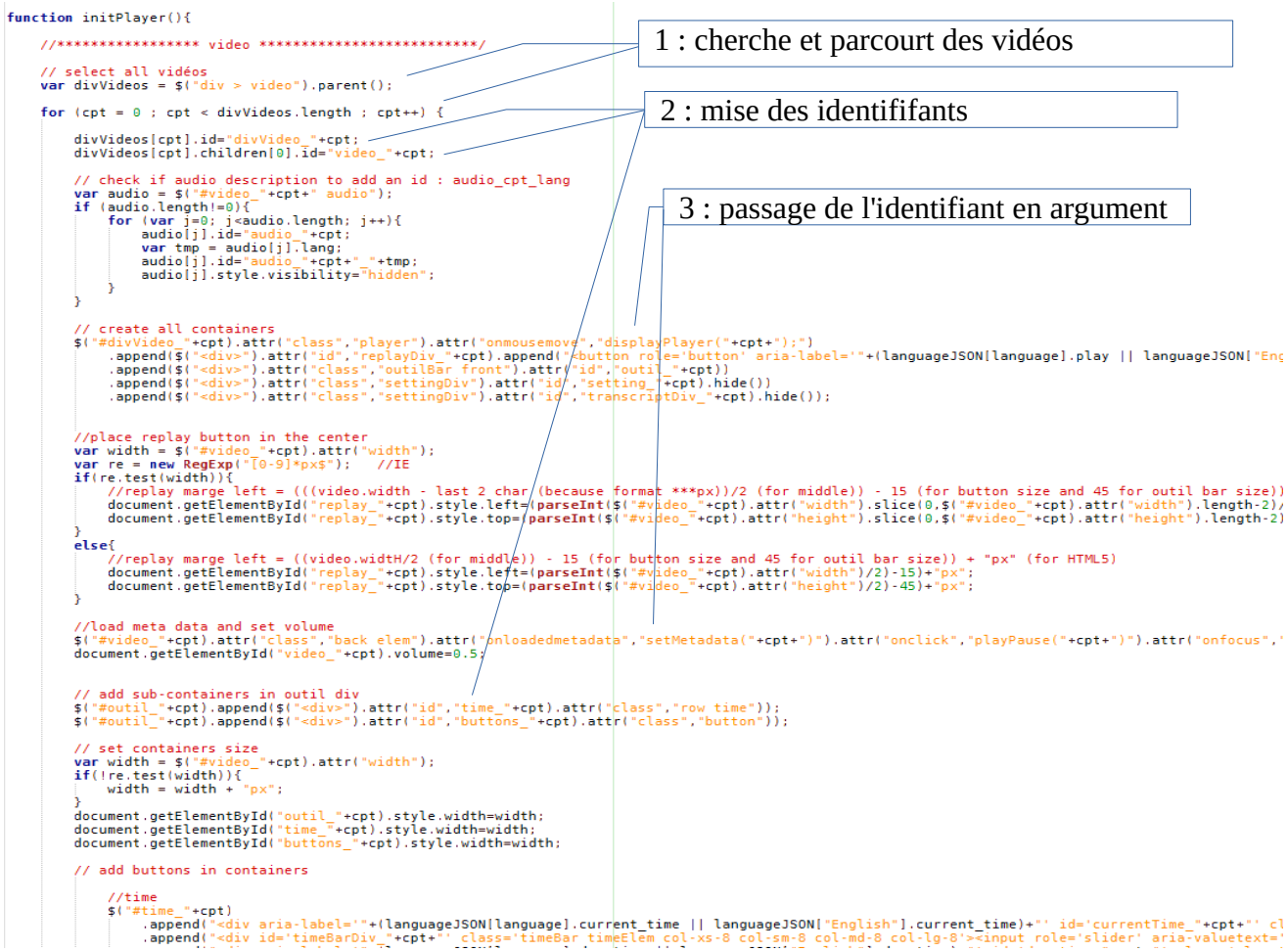


Figure 8 : Extrait de la fonction d'initialisation

Par la suite, il a fallu implémenter la barre de défilement du temps ainsi que le temps actuel et le temps maximum de la vidéo. Pour se faire l'élément *input* avec l'attribut *type='range'* présent dans HTML a été choisi, ce choix se porte sur le fait que l'utilisateur peut très facilement en changer les données car ces actions sont déjà prévues par HTML sans rien à avoir à implémenter en plus – excepté le CSS qui n'est pas géré par Bootstrap, il a donc fallu créer une classe CSS spécifique, pour que cette barre s'intègre correctement dans le style du lecteur.

Une méthode d'actualisation du lecteur a donc été créée pour mettre à jour cette barre en fonction du temps courant de la vidéo, celle-ci est automatiquement appelée toute les 5000 millisecondes à partir du moment où on lance la lecture sur un des lecteurs. Il a donc fallu, pour cette partie, connaître la durée maximum de la vidéo pour pouvoir l'afficher avant l'appui sur « lecture » mais également pour mettre un maximum sur la barre de temps. Ceci a posé un certain nombre de problèmes du fait que cette donnée n'est pas disponible au moment du chargement de la page. Il faut attendre quelques millisecondes avant que ce soit le cas. Il a donc fallu implémenter une méthode qui sera appelée lors du chargement des métadonnées pour pouvoir mettre à jour toutes les données de la vidéo. Les métadonnées sont littéralement les données des données. Si l'on prend l'exemple d'une vidéo, celle-ci est la donnée principale, les métadonnées sont tout ce qui la concerne en plus : le temps courant, la durée, le volume actuel, les sous-titres disponible, etc.

En fonction du navigateur, cette méthode n'est pas appelée au même moment dans le chargement de la page. Il a donc fallu faire une vérification pour savoir si tout était bien chargé et si ce n'était pas le cas, rappeler cette méthode quelques secondes plus tard. Le chargement de la vidéo pouvant échouer, il y a une vérification qui se fait à chaque appel. Si le nombre d'appel de la fonction est supérieur à 5, un message d'erreur est affiché et la fonction JavaScript n'est plus appelée pour éviter de surcharger le navigateur.

Il a ensuite fallu implémenter les sous-titres, la transcription textuelle et l'audio-description de la vidéo. Les deux premières se sont faites de manière très simple grâce à la sous-balise *track* de HTML5 ainsi qu'à la méthode du chargement des métadonnées. Pour le moment le lecteur gère uniquement les sous-titres au format

web VTT. C'est un format de fichier permettant d'écrire des sous-titres pour les balises *video* disponible avec HTML5. Leur ajout au sein du lecteur se fait de la manière suivante :

```
<div>
    <video width="42" height="42">
        <source src="./exemple.mp4" type="video/mp4">
        <track label="Klingon" kind="subtitles" srclang="kg" src="./your/klingon/subtitles/file.vtt">
        <track label="Primitive Quendian" kind="subtitles" srclang="qd" src="./your/quendian/subtitles/file.vtt">
    </video>
</div>
```

Figure 9 : Implémentation sous-titres

La transcription textuelle d'une vidéo est un fichier texte qui décrit tout ce qui se passe durant la vidéo permettant ainsi aux personnes ne pouvant pas voir ou entendre celle-ci de la comprendre dans les meilleures conditions possibles. Elle s'implémente de la façon suivante :

```
<div>
    <video width="42" height="42">
        <source src="./exemple.mp4" type="video/mp4">
        <track label="Gallifreyan (txt)" kind="descriptions" src="./your/gallifreyan/transcript/file.txt">
        <track label="Vulcan (html)" kind="descriptions" src="./your/vulcan/transcript/file.html">
    </video>
</div>
```

Figure 10 : Implémentation transcription textuelle

L'audio-description d'une vidéo est presque identique à la transcription textuelle de celle-ci sauf que c'est un fichier audio. Ce dernier décrit tout ce qui se passe durant la vidéo – excepté les dialogues déjà audio – permettant ainsi aux personnes ne pouvant pas la voir de la comprendre dans les meilleures conditions possibles. Elle a été quant à elle plus compliquée à mettre en œuvre puisque HTML5 n'a pas de balise spécifique pour ajouter une audio description sur un fichier vidéo. Nous avons donc opté pour intégrer des balises *audio* – servant à l'origine à lire des fichiers audio seul – directement dans la balise *video* sous la forme suivante :

```
<div>
    <video width="42" height="42">
        <source src="./exemple.mp4" type="video/mp4">
        <audio lang="dz">
            <source src="./your/dovahzul/subtitles/file.mp3" type="audio/mpeg">
            <source src="./your/dovahzul/subtitles/file.ogg" type="audio/ogg">
        </audio>
        <audio lang="dk">
            <source src="./your/dothraki/subtitles/file.mp3" type="audio/mpeg">
        </audio>
    </video>
</div>
```

Figure 11 : Implémentation audio-description.

Cette forme paraît logique du fait que le fichier audio fait partie de la vidéo, ce serait très étrange de la mettre en dehors. De plus cela permet de récupérer très facilement toutes les descriptions audio ainsi que tous les types de fichiers disponibles pour celle-ci. Les identifiants pour les fichiers audio sont générés de la façon suivante : « audio_numeroDuLecteur_contenuDeL'attributLang » cela permet grâce à une découpe de cet identifiant, de retrouver facilement la langue du fichier et le lecteur auquel il appartient. Le fichier audio se lance ensuite au même moment que le fichier vidéo, ce qui synchronise automatiquement les deux fichiers – bien évidement si le lecteur est mis en pause ou si le son est coupé, cela a lieu sur les deux à la fois. Depuis les options du lecteur – celles-ci changent automatiquement en fonction des éléments qui sont disponibles – il est possible de changer la langue des sous-titres et de l'audio-description en instantané pendant la lecture.

Le dernier grand problème de programmation a été l'implémentation de la langue de l'interface utilisateur – la langue écrite dans la fenêtre d'option et les mots qui seront restitués vocalement à l'aide des attributs *Accessible Rich Internet Applications* (ARIA) qui seront implémentés par la suite. ARIA est une spécification technique du W3C qui permet d'ajouter de nouveaux attributs dans le code HTML d'une page web. Les lecteurs d'écran les analysent ensuite pour les retranscrire. Sachant qu'il faut – comme les sous-titres et l'audio-description – changer la langue de manière simple et instantanée. Pour implémenter ceci, Il y avait deux solutions :

- sauvegarder certaines langues à l'intérieur même du code JavaScript – ce qui aurait pour conséquence que l'intégrateur pourrait très difficilement les modifier

- trouver une méthode pour faire en sorte que la personne qui va mettre ce lecteur dans son site web puisse ajouter très facilement une nouvelle langue.

Au final les deux solutions ont été implémentées, la première est uniquement là en tant que solution de secours si pour telle ou telle raison le serveur n'arrive pas à charger le fichier de langue – un message est affiché pour dire que celui-ci a échoué. Il a été choisi d'utiliser un fichier au format JSON pour gérer les différentes langues. C'est un format de données permettant de les organiser de manière simple et compréhensible par des personnes qui n'ont pas une formation de développeur, par exemple pour l'intégrateur qui ne connaît pas forcément très bien le JavaScript. Ce fichier JSON se présente sous la forme suivante :

```
"Name of your language" : //Display in setting box
{
    "close" : "Translation of close"    // without translation the player use English translation
    "language" : "Translation of language",

    "subtitle" : "Translation of subtitle",
    "subtitle_language" : "Translation of subtitle language",

    "audio_description" : "Translation of audio description",
    "audio_description_language" : "Translation of audio description language",

    "text_transcript" : "Translation of text_transcript",

    "general" : "Translation of general",
    "general_language" : "Translation of general language",

    "play" : "Translation of play",
    "pause" : "Translation of pause",
    "replay" : "Translation of replay",

    "mute" : "Translation of mute",
    "volume" : "Translation of volume",

    "setting" : "Translation of setting",

    "fullscreen" : "Translation of fullscreen",

    "time_bar" : "Translation of time bar",
    "current_time" : "Translation of current time",
    "duration" : "Translation of duration",
    "hour" : "Translation of hour",
    "minute" : "Translation of minute",
    "second" : "Translation of second",
},
```

Figure 12 : Organisation du fichier JSON contenant les langues.

Pour ajouter une langue il suffit d'ajouter une catégorie dans le fichier JSON. Le JavaScript va ensuite charger ce fichier et scanner toutes les langues disponibles et les ajouter dans le menu d'options, il sélectionnera par défaut le langage qui est

indiqué dans une variable JavaScript. Si cette langue est introuvable ou que la traduction d'un mot est manquante, le lecteur utilisera automatiquement l'anglais grâce à cette ligne de code placée à chaque appel du fichier de langage :

```
$("#subtitle_"+id).attr("aria-label",languageJSON[language].subtitle || languageJSON["English"].subtitle);
$("#audioDescription_"+id).attr("aria-label",languageJSON[language].audio_description || languageJSON["English"].audio_description);
$("#transcript_"+id).attr("aria-label",languageJSON[language].text_transcript || languageJSON["English"].text_transcript);
```

Figure 13 : Exemple de l'appel du fichier de langage.

3.5. Savoir se faire connaître

Voyant que le projet avançait bien, nous avons décidé d'en faire un peu de publicité ainsi que de le présenter durant différentes réunions. Nous sommes, tout d'abord, allés voir la web-designer de l'université d'Orléans pour lui demander son avis sur le lecteur. Elle a été très enthousiasmée par celui-ci et nous a proposé de l'implémenter dans le futur site web de l'université d'Orléans – pour le moment toujours à l'état de maquette nous n'avons donc pas d'autres nouvelles. Elle a demandé si l'apparence du lecteur – principalement la couleur – était facilement modifiable pour pouvoir adapter le lecteur en fonction de la couleur de chacune des composantes de l'université. L'organisation du fichier CSS a donc été changée pour faire en sorte que celui-ci soit le plus compréhensible possible et le plus facile à utiliser.Voici un petit exemple de l'apparence de celui-ci :

```
**************  General color **********************/

  .outilBar{ /* Bar with all button */
      background-color:lightgrey;
      opacity:1;
      color:black; /* Text color */
  }

  .settingDiv{ /* Setting box */
      background-color:lightgrey;
      opacity:1;
      color:black; /* Text color */
      border: black solid 2px;
      border-radius: 4px;
  }
```

Figure 14 : Extrait fichier CSS.

Mon maître de stage, ayant déjà écrit des articles pour Linux Magazine, m'a également proposé que nous y écrivions un article sur le projet, ce qui permettrait de promouvoir le plus possible le lecteur auprès des intégrateurs et des développeurs. Puisque c'est un magazine francophone mensuel spécialisé dans Linux et les logiciels libres. Il est publié par les éditions Diamond et est à destination de professionnels, mais également d'amateurs éclairés dans le domaine du logiciel libre. Nous avons donc envoyé un mail au rédacteur en chef pour lui demander si le sujet pouvait l'intéresser, il nous a répondu positivement et nous lui avons donc envoyé un premier plan pour notre article. En voyant ce plan il nous a proposé d'écrire deux articles au lieu d'un seul : un premier sur l'accessibilité web et un second pour savoir comment a été développé le lecteur et comment l'installer sur son site web. Nous ne savons pas encore dans combien de temps nos articles seront rédigés, mais nous espérons lui envoyer avant le 1er août pour qu'ils soient publiés dans le numéro de septembre voire d'octobre.

3.6. Ajout du ARIA

Il a ensuite fallu implémenter la restitution audio à l'aide des lecteurs d'écran. Pour ce faire nous avons utilisé les propriétés de la spécification ARIA du WAI, c'est une série d'attribut (voir schéma en annexe p35) que les développeurs peuvent ajouter sur les éléments HTML de la page web, ces attributs ne changent strictement rien à l'organisation de la page web. Ils sont uniquement interprétés par les lecteurs d'écran, ce qui leur permet de procéder à un retour audio. Par exemple si on a : *<input type='range' step='0.1' value='0.5' min='0' max='1' />* le retour audio sera : « Potentiomètre 0,5 » mais on ne sera pas de quoi il s'agit vraiment alors que si on a : *<input role='slider' aria-valuetext='50%' aria-label='Volume' type='range' step='0.1' value='0.5' min='0' max='1' />* la restitution sera : « Volume potentiomètre 50 % » ce qui sera beaucoup plus explicite. Il a donc fallu ajouter du ARIA sur tous les éléments du lecteur de façon à avoir assez de description – pour identifier correctement l'élément – sans en avoir trop – pour éviter de surcharger la restitution afin que celle-ci ne devienne pas incompréhensible. Voici un exemple du code du lecteur après

intégration des attributs ARIA.

Contrairement à JavaScript, le ARIA est très simple à implémenter car tous les navigateurs le comprennent. Il y a uniquement quelques différences de restitution très mineures entre les lecteurs d'écran – par exemple **NVDA** dira « Volume potentiomètre 50 % » alors que **VoiceOver** dira « Potentiomètre volume 50 % ». Le seul petit problème a été de trouver les attributs qu'il faut utiliser pour faire telle ou telle restitution, mais dans l'ensemble cette partie a été très rapide à implémenter. Car une fois que l'on a trouvé la solution pour un élément cela reste la même pour tous les autres du même type.

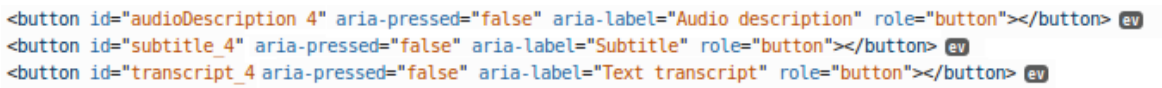


Figure 15 : Exemple de l'implémentation du ARIA.

3.7. Page de présentation du projet

Ayant un projet presque fini, nous sommes passés à l'implémentation dans une véritable page web – bien évidement toujours en respectant les règles de l'accessibilité web. Nous allons fournir cette dernière à Linux Magazine ainsi que la mettre sur le GitLab du projet. Elle se situe actuellement à l'adresse suivante : http://www.dev-and-a11y.com/projects/MediaPlayerA11y/mediaplayera11y.html. Nous avons donc mis une démonstration pour chacune des possibilités du lecteur, mais nous y avons également rédigé – en anglais – une notice explicative pour l'installer – les inclusions à faire ainsi que la syntaxe à respecter pour que celui-ci fonctionne dans les meilleures conditions possibles – le configurer – si l'on veut changer la langue par défaut, l'apparence, ajouter une langue, etc… – et l'utiliser – principalement les raccourcis disponibles sur celui-ci.

Nous y avons également ajouté une foire aux questions ainsi que la licence sous laquelle le projet est disponible. Pour cette dernière nous avons choisi la licence *« Creative Commons : Attribution-NonCommercial 3.0 Unported »* (https://creativecommons.org/licenses/by-nc/3.0/) qui permet la modification – en

citant les contributeurs ainsi qu'en remettant cette nouvelle version sous la même licence – la distribution – sous réserve de citer les auteurs – mais qui interdit toute utilisation dans un but commercial. Il n'y a donc pas possibilité de télécharger ce projet et de le vendre.

3.8. Validation et correction du code

Le projet ainsi que la page de présentation de celui-ci étant fini, il a fallu passer aux tests pour vérifier que tout fonctionne bien. Nous avons donc testé le projet sur les différents lecteurs d'écran – même si cela a été fait tout au long du stage, cette fois-ci était beaucoup plus poussée et méthodique. Ayant travaillé sous Firefox, il a fallu effectuer quelques corrections de style pour les autres navigateurs, voulant réaliser quelque chose le plus simple et le plus stable possible. Il y a un seul fichier CSS qui est utilisé par tous les navigateurs bien qu'aucun *hacks* CSS n'ait été utilisé dans celui-ci. Un *hack* CSS est une manière d'écrire une propriété de façon à utiliser un problème d'interprétation et permettre à un navigateur défectueux d'aboutir à nos fins, soit en détournant une propriété de son usage, soit en utilisant des codes supplémentaires pour pallier des manques. Il ne faut bien évidemment pas les utiliser, car nous ne savons pas si le lendemain ces problèmes d'interprétation seront corrigés.

Actuellement le lecteur a été testé avec Firefox 38.0.x, Internet Explorer 11.0 – il y reste actuellement encore un bug avec la barre de défilement du temps que l'on ne peut pas déplacer à la souris –, Google Chrome 43.0, Safari 8.0.6 et Opera 29.00. Ainsi qu'avec les lecteurs d'écran NVDA 2015.1, VoiceOver 7.03 et Orca 3.14.

Après s'être assuré de la compatibilité avec les navigateurs et les technologies d’assistance, il a fallu s'assurer que la page de présentation et le lecteur en lui-même répondaient aux critères du HTML5 et de l'accessibilité définis par le W3C. Pour tester les premiers critères nous avons utilisé les validateurs HTML5 et CSS3 mis en place par le W3C (respectivement https://validator.w3.org/ et https://jigsaw.w3.org/css-validator/). Bien évidement pour mettre en place des tests

vérifiant l'accessibilité du site et du lecteur, nous n'avons pas tout vérifié à la main. Plusieurs outils existent nous avons principalement utilisé les barres d'outils pour Firefox telles que :

- Web Developer : c'est un outil gratuit, à installer et disponible sur les navigateurs, qui permet d'effectuer différentes vérifications sur sa page web, par exemple : retirer le CSS, le JavaScript, faire appel à des vérificateurs de code, etc…

- WCAG Contrast checker : c'est un encore un outil gratuit et disponible également sur navigateur. Il permet d'effectuer des vérifications liées au contraste d'une page web, cela permet de savoir si cette dernière respecte les règles définies dans par le WCAG

- Opquast : Ce n'est pas un outil mais carrément un *add-on* à installer qui permet d'effectuer les différentes vérifications sur sa page web pour savoir si celle-ci respecte les règles définies par le RGAA. Il vérifie tous les critères de manière méthodique, s'il en manque un, il indique la ligne et ce qu'il faut faire pour le corriger.

Ayant respecté les contraintes d’accessibilité tout au long du projet, après ces tests, il a principalement fallu changer la couleur pour changer les contrastes ainsi qu'ajouter quelques attributs oubliés.

3.9. Apparence finale

Au bout de ces dix semaines le projet a finalement été mené à bien. Toutes les fonctionnalités primaires ont été implémentées, à savoir : la lecture, couper le son et régler le volume, le passage en plein écran et le changement du temps à l'aide de la barre de défilement. En plus de cela des fonctions spécifiques à l'accessibilité ont également été ajoutées à savoir : compatibilité avec les lecteurs d'écran, l'ajout de sous-titres, d'audio-description et d'une transcription textuelle.

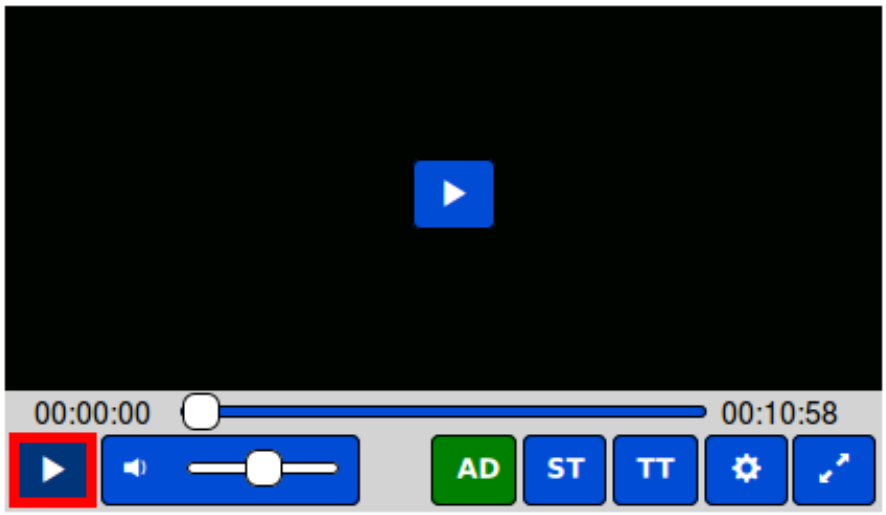


Figure 16 : Apparence du lecteur avec toutes les options disponibles

De plus l'apparence du lecteur en elle-même respecte les critères de niveau AAA qui concernent le contraste dans la catégorie « Couleur » du WCAG. Car les couleurs choisies, à savoir bleu et blanc sont des couleurs qui sont très visibles même par des personnes atteintes de trouble de la vision qui altèrent les couleurs. En plus de cela, comme on peut le voir sur la capture d'écran juste au-dessus, un rectangle rouge a été ajouté pour que l'utilisateur puisse voir très facilement et très rapidement l'endroit de son *focus*. Nous pouvons également voir que lorsqu'une fonctionnalité est active – par là l'utilisateur l'a mise en route, si celle-ci n'est pas disponible le bouton qui lui est associé n'apparaît pas – le bouton servant à l'activer change de couleur pour bien montrer que celle-ci est active. Comme demandé par la web-designer toutes ces couleurs sont bien évidement modifiables à volonté via le fichier CSS, dans ce cas c'est donc à l'intégrateur de vérifier que les couleurs qu'il utilise sont bien conformes aux recommandations du WCAG.

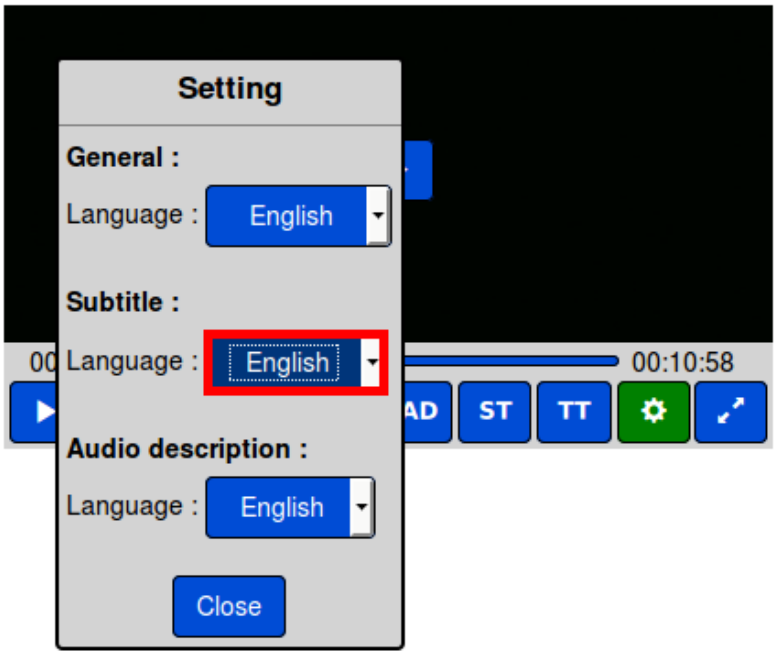


Figure 17 : Apparence de la boîte d'option avec toutes les fonctionnalités

Comme nous pouvons le voir sur cette deuxième capture d'écran du lecteur, la fenêtre d'option contient toutes les langues qui sont mises à disposition par l'intégrateur. L'utilisateur peut choisir la langue qu'il veut et pourra en voir les effets instantanément, par exemple s'il change la langue de l'audio-description, le fichier en cours de lecture va s'arrêter et la nouvelle langue va prendre le relais de manière totalement fluide et naturelle.

De plus nous avons spécialement veillé à ce que lors de l'utilisation du lecteur uniquement au clavier – avec les combinaisons de touche *tab* et *shift+tab* – il n'y ait pas de « piège au clavier » ni de *focus* perdu. C'est pour cette raison que, par exemple, lors du passage en plein écran le focus va rester sur le lecteur et ne va pas pouvoir en sortir, cela évite qu'il se retrouve « derrière » la vidéo et que l'on ne puisse pas le voir. Si malencontreusement le *focus* venait quand même à aller autre part que sur les boutons du lecteur en plein écran, une vérification faite à chaque mouvement replacerait le *focus* sur le bouton *play* de celui-ci.

Comme le prévoit le RGAA pour qu'un site web soit accessible, il faut que celui-ci contienne des raccourcis clavier. Nous en avons donc ajouté dans le lecteur de façon

à faire en sorte que beaucoup d'actions soient réalisables uniquement avec une seule touche. Voici la liste des raccourcis qui sont disponibles sur le lecteur :

Touche	Effet
Barre espace ou *p*	Lecture / Pause
m	Couper le son
f	Mettre en plein écran
o	Ouvrir / fermer la boîte des options
t	Ouvrir / fermer la boîte des transcriptions textuelles
Echap	Fermer toute boîte et sorti du plein écran
Flèche du haut	Monter le son
Flèche du bas	Baisser le son
Flèche de droite	Avancer d'une seconde
Flèche de gauche	Reculer d'une seconde

Avancer ou reculer seconde par seconde peut-être très pratique lorsque l'on recherche un moment précis de la vidéo, mais avec un média qui dure deux heures si l'on veut aller directement à 1h30 au clavier cela devient vite très long et très contraignant. Il a donc fallu trouver une solution simple et efficace, hors de question d'ajouter de nouveaux raccourcis cela compliquerait trop l'utilisation. La solution finalement adoptée a été une accélération progressive en fonction du temps que l'on maintient enfoncée la touche et du temps total de la vidéo.

3.10. Perspectives

Bien que le projet ait bien avancé durant ces dix semaines, beaucoup de choses pourraient encore être faites par la suite, par exemple :

- la gestion des sous-titres au format SRT : ce dernier est actuellement le format le plus répandu dans le monde mais n'est pas encore supporté nativement par les navigateurs web. De plus après quelques recherches j'ai pu constater que très peu de différences existent entre le format web VTT et SRT : ce ne serait donc pas un ajout très long à faire. Mais actuellement, par manque de temps, cela n'a pas été fait. Malgré cela des liens vers des projets qui effectuent ce travail ont été ajoutés

à la page de présentation.

- pouvoir personnaliser les sous-titres : faire en sorte que l'utilisateur puisse, à partir des options, paramétrer les sous-titres comme il le souhaite, par exemple en modifiant : la taille, la police d'écriture, les couleurs du texte et du fond, etc…
- pouvoir déplacer les boites des transcriptions textuelles et des options : actuellement celles-ci sont figées et ne peuvent être déplacées par l'utilisateur, à terme ce serait bien de pouvoir lui donner cette liberté.

Une autre idée un peu « extrême » consisterait à reprendre le code de ce lecteur pour le transformer en *add-on* Firefox. Celui-ci permettrait, à l'utilisateur l'ayant installé sur son navigateur personnel, d'avoir accès à toutes les vidéos HTML5 via ce lecteur. Il modifierait alors le code de la page web où se situe l'utilisateur en replaçant tous les lecteurs existant par celui-ci. Cette idée demandant beaucoup de travail ce ne serait pas vraiment une évolution du projet mais plutôt un deuxième basé sur celui-ci.

De plus une adresse mail de contact a été mise à disposition sur le site web servant de présentation du projet, si une communauté se crée autour de ce dernier peut-être que d'autres idées d'évolution pourront être proposées par celle-ci.

4. Bilan humain et technique

Durant ce stage j'ai appris beaucoup de choses, autant au niveau technique que humain. Tout d'abord au niveau technique, j'ai appris précisément les différences entre HTML et HTML5 et celle entre CSS et CC3, au début du stage je ne savais pas vraiment ce que l'on entendait avec ces appellations maintenant je comprends beaucoup mieux.

De plus bien qu'ayant réalisé un certain nombre de projets personnels en JavaScript, j'ai quand même appris beaucoup de choses dessus ainsi que sa bibliothèque la plus connue JQuery – que nous avions très peu utilisée en cours. Malgré l'avis général j'ai remarqué que cette dernière n'est pas forcément bien adaptée pour faire certaines choses ce qui m'a poussé à utiliser parfois les fonctions native de JavaScript. Par exemple les fonctions *$("#exemple")* et *document.getElementById("exemple")*, sont identiques à un tout petit détail près, lorsque l'élément avec l'identifiant « exemple » n'est pas trouvé, la version JavaScript retournera *null* alors que la version JQuery retourna quant à elle un objet JavaScript de type *selector* même si celui-ci ne pointe sur rien. Or lors de leur utilisation dans une conditionnelle, par exemple *if($("#exemple"))* celle-ci sera toujours vraie puisque la fonction ne retournera jamais *null* mais avec la version JavaScript nous savons lorsque c'est le cas et nous pouvons utiliser un *else*.

J'y ai, bien évidemment, appris également comment fonctionne l'accessibilité sur le web, tous les critères qui doivent être respectés – normalement selon la loi française tous les sites des administrations françaises doivent les respecter depuis 2011 – ainsi que comment faire en sorte que ceux-ci soit respectés. Voulant par la suite administrer mon propre site, mettant à disposition des logiciels permettant aux étudiants en situation de handicap de pouvoir suivre leurs études dans les meilleures conditions possibles, ce site pourra également accueillir un forum ainsi qu'un chat sur lequel les étudiants pourront directement échanger. Or pour que ce site soit utilisable

par ces derniers il faudra y respecter toutes les règles apprises durant ce stage.

En ce qui concerne la partie humaine, j'ai appris à travailler sur un projet avec des personnes qui n'ont pas de formation de développeur et qui donc ne comprennent pas forcément le code écrit derrière, il a donc fallu que je m'adapte en fonction de chacun. J'ai également appris à mener à bien un projet de l'idée à la production en passant par toutes les étapes de celui-ci. Contrairement à l'IUT, où nous avons presque tous la même culture informatique, là j'ai pu vraiment voir comment agissent des utilisateurs non informaticiens face au projet et adapter certaines parties en conséquence.

Le stage étant centré sur l'accessibilité, je suis donc maintenant sensibilisé encore plus à celle-ci et j'essayerai par la suite de la promouvoir dans mes projets futurs. Mon maître de stage et moi-même étant tous les deux concernés par celle-ci, nous avons donc décidé de rester en contact pour mettre en place le site http://www.dev-and-a11y.com/ sur lequel nous allons héberger différents projets open-source concernant le développement et l'accessibilité. Ces projets pourront aussi bien être sur le web, une application pour téléphone ou encore un jeu vidéo. Du moment que celui-ci à un rapport avec l'accessibilité. Nous espérons ainsi promouvoir le possible l'accessibilité qui est quelque chose de très important et pas forcément très compliqué à mettre en place. Uniquement 5min de temps de développement peut économiser plusieurs heures de temps perdu à une personne handicapée. C'est pour cela que je vous encourage dès à présent à y penser. Et n'hésitez surtout pas à installer ce projet sur votre site web s'il y a des fichiers audio ou vidéo sur celui-ci, ou même à en parler autour de vous pour que d'autres prennent conscience de son importance.

Annexes

A.1. Fiche de comparaison des lecteurs existants

Acorn :

Général :

+ Avance *frame* par *frame*

+ sous-titre (++ script en dessous)

+ Affichage d'une boite d'information qui permet de savoir le rôle du bouton

- le bouton *fullscreen* ne change pas lors du passage en plein écran

- pas de raccourci

- pas dans le bon ordre

- la barre disparaît et tout devient inaccessible en plein écran (mais toujours possibilité de se déplacer dessus au clavier)

- changement de son qui accélère

- le son est au maximum dès le démarrage

NVDA :

+ NVDA reconnaît les boutons du lecteur (donne le temps à chaque déplacement sur le curseur)

+ le volume est énoncé à l'oral lors d'un changement

- le temps ne s'actualise pas lors de l'avance de la vidéo (il reste au dernier "emplacement de déplacement")

Vibeo :

Général :

+ raccourci avance rapide/retour + son

+ la barre est toujours là en plein écran

+ bouton volume et mute à la même place

+ 2 versions graphiques

- pas de raccourci *play/pause*

- espace n'est pas désactive

- en plein écran la barre ne disparaît pas au bout d'un moment

- pas de sous titre

- quand on est en plein écran et que l'on arrive a la fin des boutons, la vidéo repasse en écran "normal" et le focus retourne au début de la liste des boutons

NVDA :

+ NVDA reconnaît les boutons du lecteur (donne le temps à chaque déplacement sur le curseur)

Access42 :

Général :

+ la barre disparaît au bout d'un moment (peut-être trop vite)

+ sous-titre

+ audio description

+ menu de paramétrage des sous titre (langue, couleur, ombre, etc)

+ menu pour accéder au fichier .txt/.html de la transcription

+ possibilité de quitter les deux menus précédents avec échap

- pas de raccourci

- les boutons se grisent lors du focus. Résultat : on a par moment l'impression qu'ils sont déactivés

- le son ne marche plus après un pause/play en plein écran, il faut retourner en écran "normal"

- le son est au maximum dès le démarrage

- problème lorsque l'on quitte le plein écran avec échap (le bouton ne change pas)

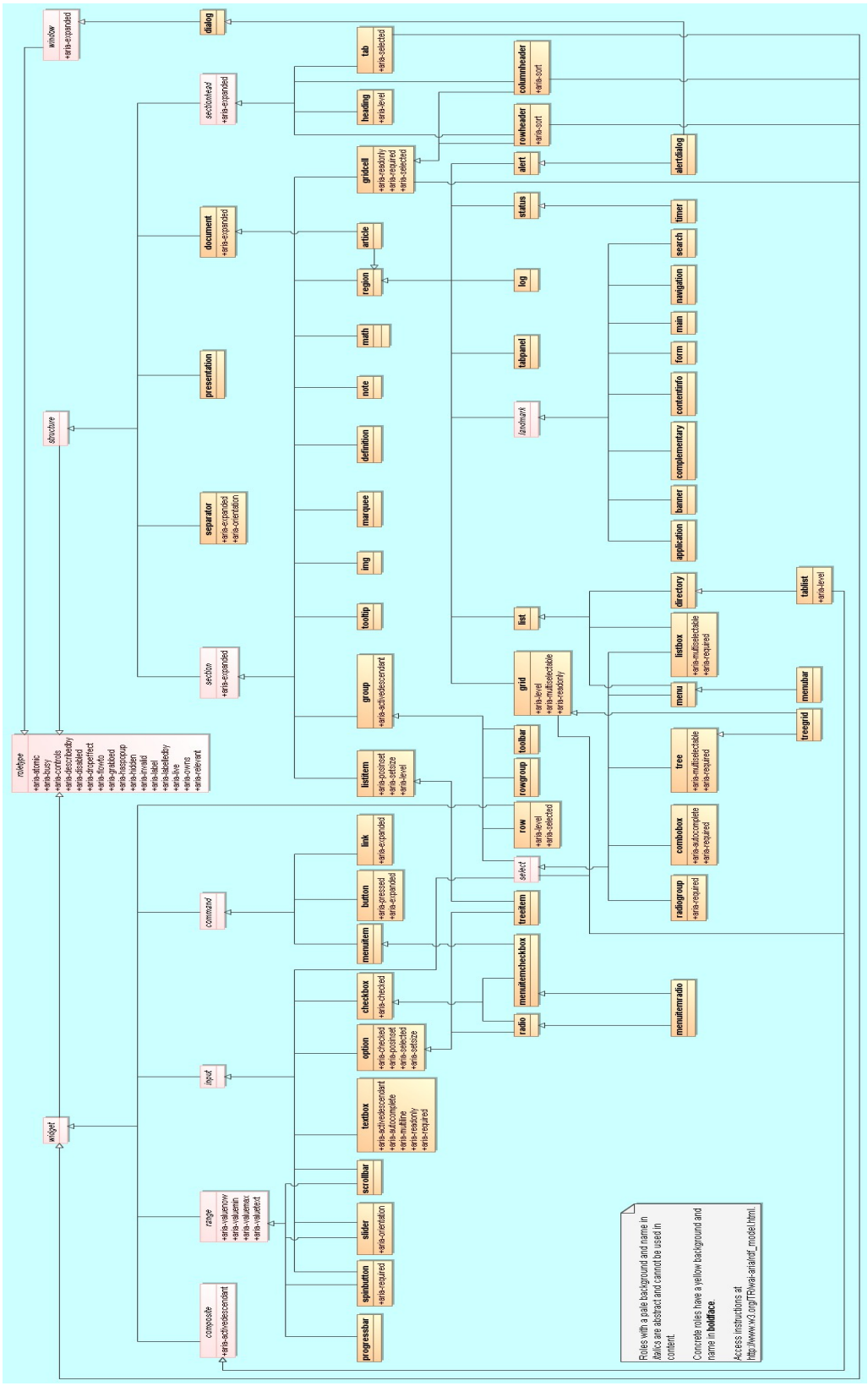
NVDA :

+ NVDA reconnaît les boutons du lecteur (donne le temps à chaque déplacement sur le curseur)

+ le volume est énoncé à l'oral lors d'un changement

- spam le temps à chaque seconde lorsque le focus est sur la barre de temps

A.2. Schéma attribut ARIA



Sitographie

Image clavier ZoomText : http://www.accessolutions.fr/IMG/png/ZoomText.png

Image téléagrandisseur : http://magasin.avh.asso.fr/270-thickbox_default/visio-book-teleagrandisseur-portable.jpg

Image plage braille : http://www.certam-avh.com/sites/default/files/IRIS40_icon.jpg

Image schéma attributs ARIA : http://www.w3.org/TR/wai-aria/rdf_model.svg

Télécharger Web Developer : https://addons.mozilla.org/fr/firefox/addon/web-developer/

Télécharger WCAG constrast Checker : https://addons.mozilla.org/fr/firefox/addon/wcag-contrast-checker/?src=search

Télécharger Opquast : http://opquast.com/fr/

Télécharger NVDA : http://www.nvda-fr.org/

Télécharger BootStrap : http://getbootstrap.com/

Télécharger JQuery : https://jquery.com/