



Université de Montpellier

Master Informatique Génie Logiciel

TER L3

par

Martin SANCHEZ
Matthieu TRINQUART
Mathis CAPISANO

Encadrants universitaires : Eric Boureau, Yoann Bonavero

Table des matières

Introduction	1
Chapitre 1 Présentation du sujet	2
1.1 Le sujet	2
1.2 L'enjeu	3
1.3 Notre approche du problème	3
1.4 Cahier des charges	4
Chapitre 2 Organisation du projet	5
Chapitre 3 Etude de Marché	6
3.1 W3C	6
Chapitre 4 Mise en application	8
4.1 Choix des solutions	8
4.2 Le Crawler	9
4.2.1 Les solutions trouvés	9
4.2.2 Notre Crawler	9
4.3 LightHouse	10
4.3.1 Le script	11
4.3.2 Les flags LightHouse	11
4.4 Selenium	11
4.4.1 Principe	12
4.4.2 Le but / Pourquoi utiliser un outil d'automatisation ?	12
4.5 Visualisation Web	13
4.5.1 Résultat	13
4.6 Installation	14

4.6.1	Installation du Crawler	14
4.6.2	Installation de Lighthouse	15
4.6.3	Installation de Selenium	15
4.6.4	Lancement	15
Chapitre 5 Analyse des résultats		16
5.1	Complexité théorique	16
5.2	Exécutions	17
5.3	Exemple d'exécution	17
5.3.1	Crawler	17
Chapitre 6 Conclusion		20
6.1	Fonctionnalités mises en oeuvre	20
6.2	Compétences apportées	20
6.3	Ouverture	21
6.4	Limites du projet	21
Chapitre 7 Définitions		22
Chapitre 8 Annexes		23
8.1	Code	23
8.1.1	Boutenbout.sh	23

Introduction

Pour notre projet de TER, notre groupe, composé par CAPISANO Mathis, TRINQUART Matthieu, HAJJAJ Jillali et SANCHEZ Martin, a été affecté au projet Enhanced Web Page Adaptation (EWPA) encadré par BOURREAU Eric, BONAVERO Yoann. EWPA est un plugin s'incrutant sur les navigateurs les plus connus (Google Chrome, Mozilla Firefox, Opera, . . .) et permettant aux utilisateurs d'ajuster le niveau de contraste entre les couleurs de fond et celles du texte. Il a été imaginé pour aider principalement les personnes handicapées visuellement. Le projet consistera donc à utiliser ce plugin et allié à d'autres logiciels, l'objectif final sera de créer des audits permettant de déterminer quelles pages d'un site sont visitables pour les utilisateurs sans le plugin mais aussi de trouver les éléments posant problème dans les pages à erreur. Une fois l'audit créé, celui-ci pourra être vendu aux propriétaires du site pour mettre en place des correctifs.

1

Présentation du sujet

1.1 Le sujet

L'accessibilité numérique est une des options méconnues de nos téléphones (sur iPhone : Réglages/Général/Accessibilité) et du web en général. Savez-vous combien de sites sont prévus pour être utilisables par des personnes sans souris (handicap moteur) ou malvoyantes (handicap visuel) ? Seulement 10% des sites web sont réellement accessibles (Réf : K. Cullen et al, L'accessibilité Web en Europe, 2010), et moins de 4% des sites des administrations (étude réalisée en 2014 par accessite-audit.com).

Actuellement un plug-in, réalisé de 2013 à 2016, est en place sur un premier prototype proof of concept disponible sur ewpa.lirmm.fr. Il permet de modifier à la volée au sein d'un navigateur, la palette de couleur pour s'adapter aux limites de la personne souffrant de handicap visuel. Il se compose de deux blocs, le premier analyse la page html via son arbre DOM (Document Object Model) et en extrait les infos géométriques et spatiales, le second résout un problème combinatoire optimisant le rendu chromatique et lumineux. Il injecte alors une palette de couleur adaptée dans le CSS.

Après une étude de marché sur les outils d'accessibilité orientés clients ou orientés serveur, le projet consiste à utiliser la version actuelle (« cliente » au sein d'un navigateur) dans une déclinaison « serveur » sans modifier l'architecture. Le but est de qualifier et quantifier un site pour fournir des métriques à un audit et ainsi proposer des solutions pour « corriger » les pages défaillantes du serveur.

1.2 L'enjeu

Ce sujet est particulièrement intéressant pour nous, car au-delà de contribuer à rendre l'internet plus accessible aux personnes souffrant d'un handicap visuel, il nous permet à nous, étudiants, de travailler dans un contexte assez réaliste. En effet, dans un contexte de travail en entreprise, le personnel est souvent amené à travailler avec du code qui est fait par quelqu'un d'autre. Dans notre cas nous avons dû comprendre, apprendre à utiliser, et nous adapter à l'extension EWPA déjà existante pour s'en servir correctement et pouvoir créer notre programme autour.

1.3 Notre approche du problème

Différentes approches du problèmes se sont proposées à nous.

En premier, nous aurions pu directement commencer à coder dès que le sujet nous a été donné. En effet, les compétences techniques ne nous semblaient pas insurmontables et nous avions déjà quelques idées de comment faire certaines parties du programme, notamment le crawler.

Deuxièmement, une possibilité aurait été de n'utiliser que des programmes ayant déjà été écrits par d'autres et Open Source. Cette solution aurait pu nous faire gagner du temps si nous avions su comment installer les logiciels nécessaires.

Enfin, la solution que nous avons adoptée, à mi-chemin entre les deux précédentes : certains programmes ont été réutilisés tels quels (LightHouse) et d'autres ont été écrits selon nos besoins (crawler).

1.4 Cahier des charges

L'objectif étant de quantifier l'accessibilité d'un site, c'est-à-dire, être capable d'évaluer quelles pages sont adaptées aux contraintes de contraste et celles qui ne le sont pas, et ainsi pouvoir proposer des corrections pertinentes. Nous devons donc être capable d'obtenir la liste des pages accessibles d'un site web, et de garder certaines données les concernant.

Nous devons être capable d'analyser ces pages, pour définir si elles sont adaptées aux contraintes de contraste, et dans le cas où elles ne le sont pas, pouvoir définir quels éléments sont à modifier.

Pour finir, pouvoir fournir des modifications pour se conformer aux contraintes, cette partie là sera assurée par l'extension EWPA, nous devons alors l'automatiser. Le logiciel final doit s'exécuter le plus rapidement possible pour analyser un site entier.

2

Organisation du projet

Dans un premier temps, nous avons pris une semaine pour chercher des solutions existantes sur internet ou des idées qui pourraient nous aider à avancer dans le projet tout en faisant des études de marchés. Les sujets de recherches étaient principalement les crawlers et la détection de contrastes.

Suite à cette semaine de recherches, nous avons alors décidé des tâches et sélectionné certains outils trouvés sur internet pour commencer à développer le projet.

Nous nous sommes alors réparti les missions suivantes :

- Analyser les niveaux de contrastes d'une page.
- Lister les liens accessibles depuis une page
- Proposer des corrections de EWPA pour les erreurs trouvées
- Pouvoir visualiser les résultats

Ensuite, nous avons mis en place un dépôt GitHub pour mieux organiser le travail individuel de chacun et avancer de manière coordonnée sur les différentes parties du programme principal.

Tout au long de ce projet nous nous sommes appuyés sur les méthodes agiles pour nous organiser autour de réunions hebdomadaires pour avancer nos tâches respectives.

Lors de ces réunions en visioconférence avec nos tuteurs, nous faisons le point sur l'avancée du projet et discussions des fonctionnalités à ajouter mais aussi des problèmes à corriger pour la semaine suivante.

Pour finir, les dernières semaines du projet ont été consacrées à l'optimisation des programmes pour gagner en temps d'exécution, car même si nous n'avons pas de contraintes particulières en temps, le programme doit s'exécuter en temps raisonnable.

3

Etude de Marché

L'objectif de l'étude de marché est de rechercher sur internet des outils numériques qui pourraient au moins partiellement nous aider à créer le logiciel que nous voulons.

3.1 W3C

W3C (World Wide Web Consortium) est une organisation internationale de normalisation du World Wide Web fondée en 1994. Un de leurs objectifs est donc aussi de rendre le Web le plus accessible possible dans le monde entier.



W3C possède une rubrique sur son site regroupant de nombreux outils numériques permettant de rendre internet plus accessible pour différents types de handicap.

Nous avons donc trouvé différent crawlers :

- <https://www.experte.com/accessibility/contrast>

Ce site est un crawler qui analyse le contraste de chaque page qu'il trouve et repère la ligne html qui pose problème. On peut ensuite exporter toutes les données en CSV. Malheureusement il est limité à 500 URL.

- <https://www.screamingfrog.co.uk/seo-spider/>

Il s'agit d'un logiciel à télécharger qui parcourt un site internet à partir de l'adresse donnée en entrée et retourne un fichier Excel. Malheureusement, celui-ci est limité

à 500 URL et pour pouvoir dépasser ces 500 URL il faut acheter la version payante.

Lors de nos recherches, nous avons aussi découvert différents outils permettant d'analyser les pages :

- <https://userway.org/scanner>

Ce site permet aussi d'analyser le contraste de toutes les pages d'un site mais celui-ci est payant.

- <https://color.a11y.com/?wc3/https://www.checkmycolours.com/>

Ce site analyse le contraste d'une page web.

- <https://github.com/gdkraus/color-contrast-chrome>

Une autre extension permettant de transformer la page à la volée (ne semble plus être maintenue).

- <https://pypi.org/project/lighthouse/>

Librairie python permettant d'utiliser LightHouse au sein du code.

- <https://github.com/GoogleChrome/lighthouse>

LightHouse en ligne de commande (bash).

- <https://webaim.org/resources/contrastchecker/>

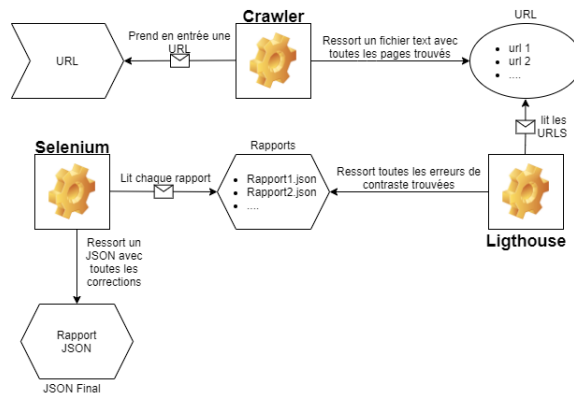
Site permettant de tester le ratio de contraste entre plusieurs éléments.

4

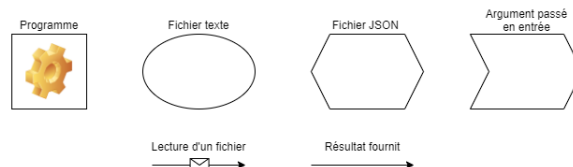
Mise en application

4.1 Choix des solutions

Le programme est constitué d'un crawler qui récupère une URL et parcourt toute l'arborescence du site et récupère ainsi tous les liens et les ajoute dans un fichier. Un programme avec lighthouse récupère toutes ces URL, calcule le ratio de contraste et crée un rapport en JSON qui est ensuite analysé par le dernier programme qui ouvre un navigateur, déclenche EWPA et récupère les corrections apportées par EWPA. Nous avons donc développé différents outils pour qu'ils s'organisent et puissent être automatisés du lancement du crawler jusqu'au rendu des correctifs de EWPA.



Légende



4.2 Le Crawler

En ce qui concerne le crawler, c'est le premier élément de la structure du projet, son rôle est de fournir à LightHouse une liste de liens valides à analyser en partant de la racine d'un site web.

4.2.1 Les solutions trouvés

Dans un premier temps, nous avons exploré les programmes déjà existants. Nous avons trouvé plusieurs crawlers : certains étaient des logiciels, d'autres des petits programmes trouvés sur github, mais dans la plupart des cas, ils étaient soit payants, soit ne faisaient pas exactement ce que nous voulions.

Nous avons donc décidé de l'écrire nous même.

4.2.2 Notre Crawler

Pour notre crawler, nous avons choisi de le programmer en python pour plusieurs raisons.

Premièrement, la facilité : il nous paraissait plus facile de le programmer avec ce langage compte tenu du temps que nous avons et de la multitude d'exemples et de tutoriels disponibles.

Deuxièmement, pour limiter le nombre de langages utilisés. En effet, au début de ce projet, python était le seul moyen à notre connaissance pour utiliser Selenium, qui nous sert par la suite à faire les rapports.

Enfin par curiosité, nous programmons souvent en C, C++ et peu en python. Nous en avons profité pour approfondir ce langage.

En ce qui concerne la programmation, nous nous sommes d'abord inspirés de programmes trouvés sur github, puis petit à petit nous avons greffé les fonctionnalités dont nous avions besoin :

- La mémorisation du nombre de visites par page.
- La création d'un fichier Excel pour visualiser les données obtenues.

- La vérification de la validité des pages (Erreurs 400-499).
- L'exclusion des formats de pages qui ne nous intéressaient pas (pdf, images, ...).

Le programme se découpe en trois parties principales :

- Vérifier la validité des liens

Pour vérifier la validité d'un lien, nous vérifions qu'il soit bien au format "leSous-domaine.leSite.leDomaine", ensuite, nous excluons les formats qui ne nous intéressent pas, les PDF, les Images, les redirections vers des mail etc..

Si toutes les vérifications passent, le lien est valide.

- Lister tous les liens dans une page

C'est la partie la plus importante du crawler, il s'agit de prendre une page valide et d'en extraire tous les liens. Pour cela nous faisons une requête à la page, pour analyser son contenu et en extraire les balises HTML `<a />`, qui contiennent les liens.

Ensuite nous vérifions si le lien est valide et si il ne l'est pas nous essayons de le corriger. Nous excluons les liens qui sortent du site, par exemple si nous sommes sur "www.lirmm.fr" nous ne garderons pas le lien "www.paslelirmm.fr".

- Effectuer un traitement avec ces liens

Cette dernière partie rassemble tout le reste, c'est ici que nous gardons en mémoire tous les liens déjà visités et à visiter, que nous créons les fichiers de sortie.

A l'exécution, ce programme prend en entrée un lien, la racine du site, et un entier, déterminant le nombre de requêtes à faire au site, "-1" étant un cas spécial pour dire de ne pas s'arrêter tant qu'il reste des pages à parcourir. Plus cet entier est grand, plus il y aura de liens en sortie de programme et plus son exécution sera longue.

4.3 LightHouse

LightHouse est un outil créé par Google dans le but de faire des audits de pages web. Cet outil récupère des informations sur les performances, l'accessibilité mais aussi sur l'optimisation du site pour les moteurs de recherches. Nous ne nous sommes intéressés qu'à la partie accessibilité et plus précisément celle gérant le contraste entre le fond et la couleur du texte.

Cet outil a été utilisé en ligne de commande (`bash`) et dans un script. Il générera des

rapports dans un fichier “reports” qu’utilisera ensuite le programme servant à comparer les erreurs et les modifications de EWPA.

4.3.1 Le script

Pour lancer LightHouse nous avons utilisé un script bash. Il nous a permis de créer le dossier “reports” s’il n’existait pas. Par la suite, il lit chaque ligne du fichier possédant les liens du site (généré par le crawler) et lance LightHouse avec diverses options sur ce lien.

4.3.2 Les flags LightHouse

`-chrome-flags='-headless -no-sandbox'` : permet de lancer le navigateur “sans tête” et sans sandbox. C’est-à-dire que l’interface graphique ne sera pas affichée et que les scripts pourront s’exécuter simplement sans entrer en conflit avec les limitations du navigateur.

`-only-audits=color-contrast` : lance LightHouse pour ne faire que des audits concernant les contrastes de couleur.

`-output=json` : demande un fichier de rendu en JSON.

`-output-path=./reports/report$siteNumber.json` : créer le rendu et l’envoie dans le fichier report[NumeroDuReport].

`-disable-storage-reset` : désactive la réinitialisation du stockage. Dans notre cas, ça nous fait gagner du temps puisqu’à chaque fin d’exécution de LightHouse, celui-ci est censé supprimer les données de navigation. Si on choisit de ne pas supprimer ces données, cela peut accélérer le déroulement du programme.

`-quiet` : désactive l’affichage dans la console

4.4 Selenium

Selenium est un outil d’automatisation open source pour tester des applications web. On peut le faire de différentes manières et avec différents langages tels que Java, C, Py-

thon, Ruby

Selenium utilise un composant spécial des navigateurs : le WebDriver. Ils sont mis à disposition par les navigateurs web. Par exemple Google Chrome à ChromeDriver et Firefox GeckoDriver. Ces composants permettent une liaison entre le navigateur et le code Selenium.

4.4.1 Principe

Le principe est simple, Selenium nous met à disposition plusieurs fonctions qui permettent d'interagir avec le navigateur, ici nous avons utilisé Python pour programmer notre code.

4.4.2 Le but / Pourquoi utiliser un outil d'automatisation ?

L'utilisation de Selenium (ou autre outil d'automatisation) était obligatoire au vu du projet demandé. Nous devons, grâce aux liens récoltés par le crawler, établir un audit afin de récolter les pages présentant un défaut de contraste.

LightHouse permet de parcourir les liens un par un et de voir si la page présentait un défaut de contraste (qui est de 21,5%). Nous devons après ça apporter une correction à cet audit et c'est là que Ewpa et Selenium interviennent. Ewpa étant une extension il nous fallait donc un navigateur afin de le faire fonctionner.

Dans un premier temps, après avoir chargé la page, nous laissons EWPA analyser et traiter la page. Lorsqu'il a fini, il nous prévient en ajoutant à la balise body du DOM l'attribut ewpa-finished, il suffit donc de dire à Selenium d'attendre cet attribut pour commencer à travailler sur les changements apportés. Après avoir chargé Ewpa, le programme parcourt tous les défauts détectés par LightHouse (qui sont donnés par un sélecteur de type "div.col > div.content-container > h5.fusion-responsive-typography-calculated > strong") et recherche grâce au selecteur la balise présentant un défaut de contraste. Lorsqu'il a trouvé la balise présentant le défaut, il récupère la couleur de l'arrière-plan et du premier plan afin de comparer dans l'audit l'ancienne couleur (qui est stockée dans LightHouse) et la nouvelle que Selenium vient de récupérer.

Après avoir récupéré ces informations, le programme met en forme tous les résultats obtenus en JSON (figure 1). Nous avons les informations qui suit :

- L'ancienne couleur de l'arrière plan et du premier plan (avant la correction de Ewpa)
- Le lien
- Le sélecteur où le défaut est présent
- La correction apportée

```
▼ correction {2}
  color : □ rgb(254, 254, 254)
  backgroud-color : ■ rgb(176, 41, 43)
  forground-color : □ #ffffff
  selector : div.fusion-column-wrapper > div.fusion-aligncenter > a.fusion-
             button > span.fusion-button-text
  link : https://sciences.edu.umontpellier.fr/offre-de-formation/
  explanation : Expected contrast ratio of 4.5:1
  background-color : ■ #ff6d77
```

figure 1

Il faut noter que le fichier JSON stocke toutes les erreurs de toutes les pages.

Après avoir rendu le JSON au format ci-dessus, vient la partie de visualisation que nous avons réalisée afin de mieux comprendre les changements effectués.

4.5 Visualisation Web

Pour faire la visualisation du projet nous avons utilisé le langage HTML/CSS et JavaScript.

4.5.1 Résultat

Pour analyser et mettre en page l'audit, nous avons utilisé JavaScript qui dans un premier temps charge le rendu JSON du programme Python (Selenium). Le JSON recensant tous les défauts du site web, il nous suffit donc de parcourir tous les défauts et de les mettre en forme. Par exemple pour mettre en page tous les défauts d'une page (d'un lien) on va parcourir tout le JSON et ne prendre que les défauts présentant ce lien. Après

avoir mis en relation les reports et les liens nous obtenons le résultat qui suit :

Reports		
https://sciences.edu.umontpellier.fr/diplomes-a-la-fds-disponibilite-retrait-envoi/	VOIR LE RAPPORT	20 reports
https://sciences.edu.umontpellier.fr/espace-etudiants/certificat-informatique-et-internet-c2i/	VOIR LE RAPPORT	21 reports
https://sciences.edu.umontpellier.fr/la-cesure/	VOIR LE RAPPORT	4 reports
https://sciences.edu.umontpellier.fr/2021/02/09/10-mars-2021-atelier-strategie-et-techniques-de-recherche-dentreprises/	VOIR LE RAPPORT	17 reports

Si nous voulons voir les reports du premier lien nous obtenons :

<code>div.fusion-text > div.fusion-aligncenter > a.fusion-button > span.fusion-button-text</code>	après avant
<code>ul.nav-tabs > li.active > a#mobile-fusion-tab-diplômésrésidentenfrance > h4.fusion-tab-heading</code>	après avant

4.6 Installation

4.6.1 Installation du Crawler

Pour pouvoir lancer le Crawler, vous aurez besoin de Python, dans une version ultérieure à 3.5, pour l'installer sous Linux :

— `apt install python3.7`

Ensuite vous aurez besoin de “pip” qui permet d'installer des modules python, pour l'installer :

— `curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py`

— `python3.7 get-pip.py`

Et pour finir, vous devez installer les extensions suivantes, à noter que certaines sont installés par défaut avec python3.7 :

- pip3.7 install requests
- pip3.7 install time
- pip3.7 install socket
- pip3.7 install urllib3
- pip3.7 install bs4
- pip3.7 install sys
- pip3.7 install csv

4.6.2 Installation de Lighthouse

En premier lieu, Google Chrome doit être installé. Ensuite, installer Node (<https://nodejs.org/en/>) et enfin exécuter la commande “npm install -g lighthouse”.

4.6.3 Installation de Selenium

Pour installer Selenium sur python il suffit de suivre les étapes suivantes :

- Si vous avez pip : pip install -U selenium
- Sinon : python setup.py install

Pour utiliser l’extension dans Selenium veuillez vérifier que le path doit être un chemin absolu dans ”extension_path”. Afin de récolter les reports, il faudra aussi changer le path de la variable “path” qui stocke le chemin du répertoire ‘reports’.

4.6.4 Lancement

Pour lancer le programme, il suffit d’exécuter la commande shell suivante dans le répertoire du projet :

./boutenbout url du site nombre de requêtes du crawler

Exemple :

./boutenbout http ://www.lirmm.fr/ 5

5

Analyse des résultats

5.1 Complexité théorique

p : nombre de requête

n : nombre de liens dans une page

pn : nombre total de liens

Complexité du crawler : $O(pn^2)$

Complexité du programme scriptLighthouse.sh : $O(pn)$

m : nombre de sélecteurs

i : nombre d'éléments ayant le sélecteur

Complexité du programme selenium.py : $O(npmi)$

Complexité totale : $O(pn + pn^2mi)$

5.2 Exécutions

Site du LIRMM (<http://www.lirmm.fr>) :

	Requêtes 5	Requêtes 10	Requêtes 20	Requêtes 30
durée d'exécution	3min42s	5min41s	7min47s	10min50s
nombre de liens analysés	15	27	38	52
Nombre de pages avec erreurs	13	21	27	40
Nombre d'erreurs	112	189	237	363

Site de la faculté des sciences (<https://sciences.edu.umontpellier.fr/>) :

	Requêtes 5	Requêtes 10	Requêtes 20	Requêtes 30
durée d'exécution	24min38	24min51s	32min48s	35min18s
Nombre de liens analysés	78	77	84	87
Nombre de pages avec erreurs	73	72	79	75
Nombre d'erreurs	975	972	1031	1025

En voyant ces résultats, on peut se rendre compte de plusieurs choses.

- La durée d'exécution peut complètement différer d'un site à l'autre pour un même nombre de requêtes. Cela s'explique assez facilement : le nombre de liens sur une page peut être très différent.
- Le nombre de pages analysées pour un même site peut être presque le même, même si le nombre de requêtes diffère. En effet, si sur un site on retrouve les mêmes liens sur toutes les pages, ils seront presque tous trouvés dès les premières requêtes.

5.3 Exemple d'exécution

Pour montrer le déroulement de l'exécution du programme, nous allons le faire fonctionner sur le site du lirmm(www.lirmm.fr) avec 3 requêtes.

La commande pour lancer ce programme est donc :

```
./boutenbout https ://www.lirmm.fr/ 3
```
















5.3.1 Crawler

Dans un premier temps, le script va lancer le crawler sur l'adresse et avec le nombre de requêtes données.

Nous obtenons la liste de liens suivant :

<http://www.lirmm.fr/>
<http://www.lirmm.fr/recherche/plateformes/plateforme-robotique>
<http://www.lirmm.fr/recherche/departements/mic>
https://www.lirmm.fr/lirmm_admin
<http://www.lirmm.fr/le-lirmm/emplois>
<http://www.lirmm.fr/recherche/departements/rob>
<http://www.lirmm.fr/le-lirmm/presentation>
<http://www.lirmm.fr/recherche/plateformes/plateforme-microelectronique>
<http://www.lirmm.fr/recherche/plateformes/plateforme-informatique>
<http://www.lirmm.fr/recherche/departements/info>
<http://www.lirmm.fr/recherche/equipes>
<http://www.lirmm.fr/users/utilisateurs-lirmm/robin-passama>
<http://www.lirmm.fr/users/utilisateurs-lirmm/pascal-lepinay>
<http://www.lirmm.fr/users/utilisateurs-lirmm/olivier-tempier>
<https://www.lirmm.fr/users/utilisateurs-lirmm/marc-gouttefarde>

Ensuite, le script lance Lighthouse sur chacun de ces liens pour générer des rapports suivant :

Nom	Modifié le	Type	Taille
 report1.json	26/04/2021 14:42	Fichier JSON	31 Ko
 report2.json	26/04/2021 14:42	Fichier JSON	23 Ko
 report3.json	26/04/2021 14:42	Fichier JSON	25 Ko
 report4.json	26/04/2021 14:43	Fichier JSON	19 Ko
 report5.json	26/04/2021 14:43	Fichier JSON	23 Ko
 report6.json	26/04/2021 14:43	Fichier JSON	25 Ko
 report7.json	26/04/2021 14:43	Fichier JSON	26 Ko
 report8.json	26/04/2021 14:43	Fichier JSON	24 Ko
 report9.json	26/04/2021 14:43	Fichier JSON	23 Ko
 report10.json	26/04/2021 14:43	Fichier JSON	25 Ko
 report11.json	26/04/2021 14:43	Fichier JSON	42 Ko
 report12.json	26/04/2021 14:43	Fichier JSON	33 Ko
 report13.json	26/04/2021 14:43	Fichier JSON	28 Ko
 report14.json	26/04/2021 14:44	Fichier JSON	28 Ko
 report15.json	26/04/2021 14:44	Fichier JSON	29 Ko

Après, Selenium utilise l'extension EWPA pour corriger les pages qui contiennent des erreurs et génère le rapport final.



Nous pouvons maintenant visualiser les corrections sur une page web

Reports		
http://www.lirmm.fr/recherche/departements/rob	VOIR LE RAPPORT	7 reports
https://www.lirmm.fr/users/utilisateurs-lirmm/marc-gouttefarde	VOIR LE RAPPORT	12 reports
http://www.lirmm.fr/recherche/plateformes/plateforme-robotique	VOIR LE RAPPORT	5 reports
http://www.lirmm.fr/recherche/plateformes/plateforme-informatique	VOIR LE RAPPORT	5 reports
http://www.lirmm.fr/recherche/plateformes/plateforme-microelectronique/mic-autres	VOIR LE RAPPORT	5 reports
http://www.lirmm.fr/users/utilisateurs-lirmm/olivier-tempier	VOIR LE RAPPORT	11 reports
http://www.lirmm.fr/	VOIR LE RAPPORT	14 reports
http://www.lirmm.fr/le-lirmm/presentation	VOIR LE RAPPORT	8 reports
http://www.lirmm.fr/recherche/equipes	VOIR LE RAPPORT	27 reports
http://www.lirmm.fr/recherche/plateformes/plateforme-microelectronique/securite-cyber-physique	VOIR LE RAPPORT	5 reports
http://www.lirmm.fr/users/utilisateurs-lirmm/pascal-lepinay	VOIR LE RAPPORT	11 reports
http://www.lirmm.fr/recherche/departements/mic	VOIR LE RAPPORT	7 reports
http://www.lirmm.fr/recherche/departements/info	VOIR LE RAPPORT	7 reports
http://www.lirmm.fr/le-lirmm/emplois	VOIR LE RAPPORT	5 reports
http://www.lirmm.fr/recherche/plateformes/plateforme-microelectronique	VOIR LE RAPPORT	5 reports
http://www.lirmm.fr/users/utilisateurs-lirmm/robin-passama	VOIR LE RAPPORT	16 reports

Si on clique sur un des liens "voir le rapport", nous arrivons sur une page expliquant les différentes erreurs et comment les modifier :

div.fusion-text > div.fusion-aligncenter > a.fusion-button > span.fusion-button-text

après

avant

ul.nav-tabs > li.active > a#mobile-fusion-tab-diplômésrésidentenfrance > h4.fusion-tab-heading

après

avant

6

Conclusion

6.1 Fonctionnalités mises en oeuvre

Le but du projet était de pouvoir évaluer le contraste d'un site et de voir s'il rentre dans les normes et de proposer une correction grâce au plugin du lirmm EWPA si ce n'était pas le cas.

Notre programme peut analyser tous les liens d'une page web et repérer les problèmes de contraste. Ensuite un fichier JSON est créé avec toutes les corrections à apporter. Pour une meilleure visualisation des problèmes de contraste une interface web permet de mieux voir les problèmes de couleur. Par ailleurs, le programme peut mettre du temps à analyser des sites contenant beaucoup de pages et c'est pour cette raison que nous pouvons limiter le nombre de requêtes (depuis la racine donnée) des pages à analyser pour ainsi diminuer le temps de calcul.

Ainsi, nous avons réussi à réaliser les tâches proposées par le cahier des charges du projet TER.

6.2 Compétences apportées

Tout d'abord, ce projet nous a permis de renforcer nos habitudes et manières de travailler puisque nous nous sommes largement inspirés des méthodes agiles. Nous avons ainsi pu reprendre les marques prises à l'IUT où ce genre de projets étaient communs. Cette organisation nous a donc habilité à réagir rapidement aux demandes de nos tuteurs.

Nous avons aussi découvert d'autres manières de rechercher des programmes existant déjà (notamment sur le site du W3C mais aussi sur GitHub en appliquant des filtres de

recherche).

Enfin, pour la première fois, nous avons pu développer en python. Ce fut une expérience enrichissante de développer avec un langage de si haut niveau. De plus, afin de lier les parties entre elles, des scripts bash ont été écrits. Une fois de plus, c'était une première pour nous et nous en sommes ravis.

6.3 Ouverture

Pour notre projet nous avons simplement analysé les contrastes de chaque pages Web d'un site et montré les erreurs ainsi que la correction appropriée .Cependant nous aurions pu faire des statistiques sur différents sites pour ainsi, lorsqu'on analyse le contraste d'un site web, le comparer à une base de données et voir si son niveau de contraste est dans la moyenne de celle-ci ou si le site analysé fait partie des "mauvais élèves".

Il était aussi proposé par les tuteurs qu'on adapte notre logiciel pour qu'il puisse analyser chaque site et voir si les couleurs sont compatibles avec la vision des daltoniens.

De plus, nous aurions pu aussi ajouter une suppression des données à la fin ou au début de l'exécution du script principal afin de donner un aspect un peu plus rangé. Enfin, nous aurions pu donner différents paramètres au crawler pour, par exemple, l'exécuter sur une durée ou avec une limite de liens, etc

6.4 Limites du projet

Pour le temps d'exécution nous sommes limités par le framework et EWPA. Lighthouse, Selenium et EWPA ont un temps d'exécution qu'on ne peut pas réduire donc le temps total d'exécution de notre logiciel sera borné par le temps d'exécution de ces outils.

Malheureusement, le temps d'exécution sera aussi borné par le temps de téléchargement de la page web qui dépend de la vitesse de connexion du client.

Une autre limite est atteinte quand le background de la page HTML est transparent. En effet nous avons eu plusieurs fois le problème où le background était transparent et qu'il fallait additionner toutes les transparences de tous les backgrounds afin de trouver la bonne couleur. Cependant, l'algorithme pour représenter la vraie couleur est compliqué et nous n'avons pas eu le temps de le faire.

7

Définitions

Contraste : Le contraste de couleurs correspond à la juxtaposition de couleurs opposées sur le cercle chromatique.

Crawler : Il s'agit d'un programme permettant de parcourir l'arborescence d'un site.

Selenium : API permettant de faire faire des actions à un navigateur grâce à un langage de programmation.

LightHouse : API permettant de créer des audits de pages web. Ces audits se font sur plusieurs critères : performance, qualité et bien d'autres. Dans notre cas, nous nous sommes intéressés à la partie traitant la qualité de la page et en particulier, le contraste entre les couleurs.

JSON : Format de données textuelles. Il permet de représenter des informations de manière structurée.

8

Annexes

8.1 Code

8.1.1 Boutenbout.sh

```
#!/bin/bash
if [ $# -ge 1 ]
then
    eval "python3.9 ./Crawler/MyCrawler.py $1 $2"
    ./scriptLighthouse.sh

    numberOfErrors=$(cat ./reports/* | grep -c "Element
        has insufficient color contrast")
    echo "-----$numberOfErrors erreurs au total
        -----"

    declare -i pagesWithErrors=0
    for FILE in ./reports/*;
    do
        errorsInThisFile=$(cat $FILE | grep -c "
            Element has insufficient color
```

```
        if [ $errorsInThisFile -ne 0 ]
        then
            let "pagesWithErrors += 1"
        fi
    done
    numberOfFiles=$(ls -l reports/ | wc -l)
    echo "-----$pagesWithErrors/$numberOfFiles
        pages avec erreur(s)-----"

    export PATH=$PATH:$PWD
    eval "python3.9 jillalyNewest.py"

else
    print "Error : syntaxe ./boutenbout.sh <url> [deep]"
fi
```