Université de Montréal

IFT3225 (Techno. Web)
Projet 1
Shell/CSS/HTML5/bootstrap
pour le scraping de pages HTML de tests CSS de W3C

Contact :
Philippe Langlais
RALI/DIRO
Université de Montréal

+1 514 343 61 11 ext: 47494
felipe@iro.umontreal.ca
http://www.iro.umontreal.ca/~felipe/

▮ dernière compilation : 28 janvier 2022 (11:50)

Contexte

W3C offre une suite de tests CSS destinée à tester la compatibilité de votre navigateur à certains aspects de la norme CSS érigée par cet organisme. Il s'agit de 172 tests, chacun d'eux correspondant à une ressource HTML à bord de laquelle sont embarqués les codes HTML et CSS du test. Votre but dans ce projet est de transformer ces pages en une suite de pages abritées au DIRO de façon à ce que la réponse d'un test ne soit pas affichée par défaut, comme c'est le cas actuellement. Vous en profiterez pour bonifier la présentation visuelle des pages de test.

Notez que dans ce projet, il ne vous est pas demandé de créer des pages dynamiques, mais bel et bien de pré-traiter les 172 pages de test de W3C et de les abriter sur le serveur Web du DIRO. Vous ne devez en particulier pas faire usage de javascript qui sera retiré lors des tests. L'aspect visuel des pages abritées au DIRO pourra bénéficier de bootstrap. La transformation des pages de W3C en pages abritées au DIRO peut se faire à l'aide d'une librairie python comme beautiful soup, ou par scriptage shell (plus audacieux ici).

Toutes les ressources HTML et CSS déployées au DIRO doivent être aux normes HTML5 et CSS3 respectivement. Pour mémoire, le serveur Web du DIRO sert les ressources qui sont rangées dans le répertoire (et ses descendants) `/home/www-ens/USAGER/public_html` (où `USAGER` est votre nom d'utilisateur). Plus à ce sujet ici.

À faire

1. Écrire un **script shell**[1] `scrap` qui aspire les pages de test de W3C. Ce script doit prendre en argument le répertoire (qui peut ne pas encore exister) où doivent être stockées les pages. Ce script sera testé sur une machine du DIRO à l'aide par exemple des instructions suivantes :

   ```
   cd /u/felipe/test/projet-etudiant1   % repertoire fictif
   scrap monrep                         % où se trouve votre solution
   ```

   À la fin de l'exécution, 172 pages HTML devraient être crées dans le répertoire indiqué (`monrep` dans l'exemple). Mettez un délai d'au moins 2 secondes entre chaque extraction de façon à ne pas saturer le site web abritant les pages de W3C.

2. Écrire un **script shell** `css` qui passe en revue toutes les ressources HTML de W3C (votre copie locale), en extrait le code CSS (vous pouvez utiliser un extracteur à l'aide de beautiful soup ou équivalent) de test et vérifie sa validité. Vous utiliserez pour cela le compilateur sass que vous installerez au besoin en suivant les instructions (option GitHub). Indiquez dans votre rapport les fichiers HTML qui contiennent des règles CSS qui ne sont pas aux normes en regroupant ces fichiers par type d'erreur rencontrée. Votre rapport devrait contenir quelque chose ressemblant à ceci (une même ressource CSS peut rencontrer plusieurs problèmes) :

   ```
   Error: Top-level selectors may not contain the parent selector "&".
   files: css3-modsel-144.html, ...

   Error: Expected identifier.
   Files: ....

   Error: expected "|".
   Files: ...
   ```

1. Utilisez le language shell de votre choix : csh, bash, etc. Le script ne doit faire appel qu'à des commandes shell : pas de javascript, perl, python, etc.

3. Écrire un programme (ou script shell) `transf` qui prend en entrée (au moins) une page HTML correspondant à un test W3C. Ce programme doit produire les ressources nécessaires à la réalisation d'une page HTML qui présente le test de façon plus élégante et sans que la réponse ne soit affichée. Un exemple est fourni en figure 1 où on observe les 3 zones importantes du test (le code CSS, le code HTML et son rendu sans le stylage). La page HTML produite :
   — doit être écrite en HTML5,
   — doit être valide selon le validateur W3C :
   — ne doit pas mélanger commandes CSS et HTML,
   — ne doit pas contenir de script javascript,
   — doit de préférence faire usage de bootstrap et doit visuellement être soignée.

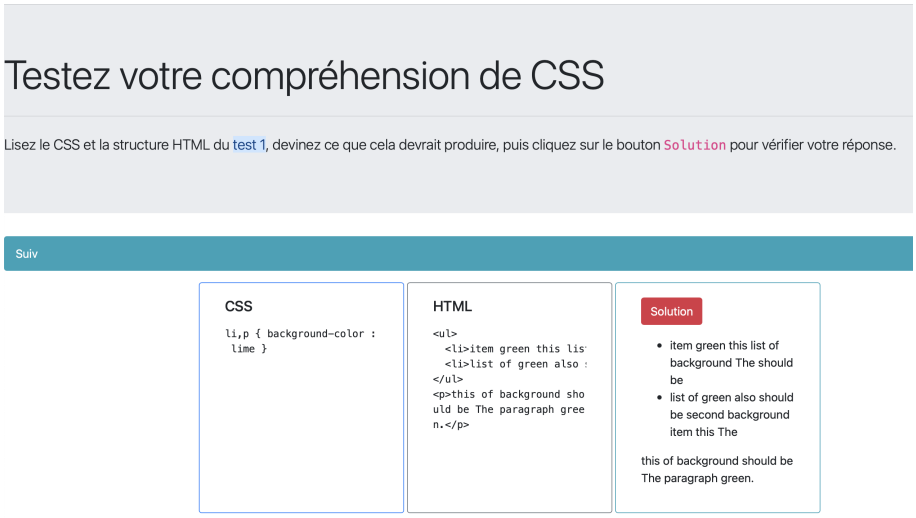

FIGURE 1 – Exemple de page proposant le premier test de W3C. Notez que la 3è boite affiche le visuel du code HTML mais pas le résultat du *stylage CSS* que l'on peut voir si l'on clique sur le bouton `Solution` dont le résultat sera rendu par votre navigateur par une autre page illustrée en figure 2. Vous observerez également que la seconde boite brouille le texte dans le HTML (sans perturber la lecture du code HTML) de façon à ce que la réponse ne soit pas lisible facilement.

Votre programme devra composer une ou plusieurs ressources à partir

de la page de test. Chaque page HTML doit contenir au minimum les informations visualisées dans les figures données en exemple, à savoir : le code CSS et HTML du test, ainsi qu'un mécanisme pour passer au test suivant ou précédent. [2] Les éléments textuels révélant la réponse devront être *brouillés*. Dans l'exemple fourni, les mots des textes dans le balisage HTML ont été mélangés. Vous pouvez introduire une autre transformation moins facile à lire, pour autant que la structure HTML affichée permette encore de répondre au test.

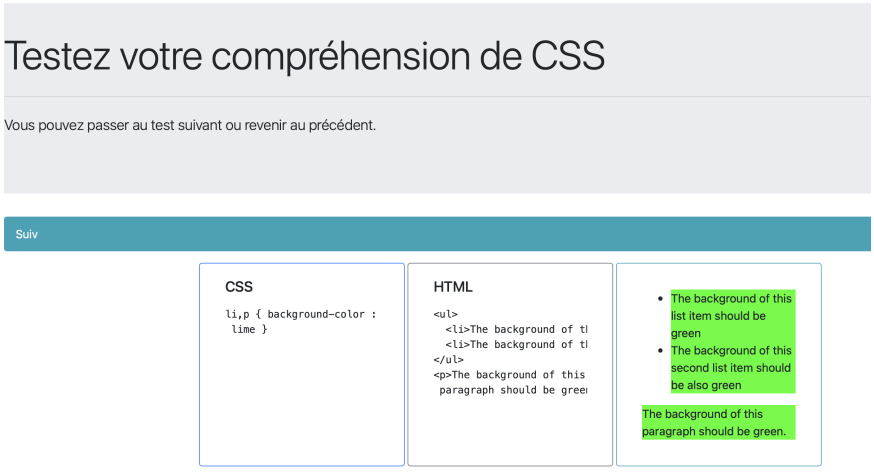

FIGURE 2 – Exemple de page proposant le premier test de W3C et sa solution (3è boite). Vous observerez également que la seconde boite ne brouille plus le texte HTML du test. En cliquant sur `Suiv` l'utilisateur se rend à la page illustrée en figure 3 qui offre le deuxième test de W3C.

Note : tentez d'éviter que le CSS de test n'impacte votre page autrement que dans la boîte où la solution est affichée. Par exemple une règle qui s'appliquerait sur un paragraphe pourrait influencer le rendu d'un paragraphe si vous utilisez un tel élément. Indiquez dans votre rapport comment vous avez le cas échéant réglé le problème.

4. Écrire un script shell `run` qui prend en argument un répertoire contenant les tests W3C téléchargés et un répertoire de sortie où seront générées les ressources nécessaires (HTML/CSS) pour donner accès

2. Les liens Prev et Suiv dans les exemples.

aux tests bonifiés. Ce script fera appel au programme `transf`. Voici un exemple d'usage de votre commande :

```
cd  /u/felipe/test/projet-etudiant1
run monrep /home/www/felipe/public_html/yEW5g93BahmCcKVB
```

dont l'exécution doit produire (dans `yEW5g93BahmCcKVB`) les ressources prêtes à être servies par le serveur du DIRO. Une fois ce script exécuté, il vous suffira d'exposer une url à partir de laquelle tout est accessible sans jamais devoir saisir une autre URL dans le navigateur. Ainsi si vous exposez le premier test, sa solution doit être accessible (en cliquant sur le bouton `Solution` de la page) et il doit être également possible de passer aux autres tests.

FIGURE 3 – Exemple de page proposant le deuxième test de W3C.

5. Vous devez produire un rapport au **format pdf** d'au plus 3 pages dont le nom est constitué du nom de chaque membre de l'équipe, comme dans cet exemple : `Pierre_Lapointe+Jérome_Minière.pdf`. Ce rapport doit au moins contenir :
   — l'url du premier test que vous avez généré. Cet url doit être cliquable (depuis le pdf) de façon à simplifier les tests.
   — la liste des langages utilisés pour coder votre projet.
   — très brève description des entrées de votre programme `transf`.
   — la liste des ressources CSS qui ne sont pas aux normes (selon `sass`)
   — le rôle de chaque membre de l'équipe.

Mises en garde

Dans la mesure où je vous demande de créer des pages HTML abritées au DIRO, afin de pouvoir les consulter, vous devrez les mettre en lecture pour tout le monde (`chmod og+r`). Ce faisant, rien n'empêcherait une personne de récupérer vos pages. Cette personne pourrait être un étudiant du cours qui pourrait ainsi vous mettre (malgré vous) en situation de plagiat. Afin de réduire ce risque, je vous recommande de créer un `path` depuis l'endroit où sont abritées vos pages au DIRO avec un ou plusieurs répertoires au nom improbable (ex : `yEW5g93BahmCcKVB`[3]) et dont les seuls droits d'accès sont en eXécution pour les autres (`o`) et le groupe (`g`) (ex : `chmod 711 yEW5g93BahmCcKVB`).

Vos programmes doivent fonctionner sur les machines du DIRO, sans imposer au correcteur une quelconque installation de librairie ou autre ressource, pas plus que de compilation. Donc si vous êtes tenté de proposer une solution en Java, assurez vous que le correcteur n'aura pas à se soucier de compiler les sources. Vérifiez bien que toutes les ressources demandées sont accessibles.

Notation

Ce projet compte pour 15 points répartis comme suit :

2 programme `scrap`

5 programme `transf`

2 programme `run`

2 tests d'exécution (en jouant à partir de l'url cliquable)

2 aspect visuel des pages générées

2 rapport

Le respect des consignes pour chacun des programmes à remettre fera partie de la note donnée à chaque programme, tout comme leur bonne exécution.

3. N'utilisez pas ce nom ou un nom proche !

Remise

La remise est à faire sur Studium sous le libellé `projet1`. Vous devez remettre votre code et votre rapport (format pdf) dans une archive (gzip, tar, tar.gz) dont le nom est préfixé de `projet1-noms`, où `noms` est à remplacer par l'identité des personnes (`prénom_nom`) impliquées dans le projet. Donc si j'avais à remettre seul mon solutionnaire au projet1, je le ferais sous le nom `projet1-philippe_langlais.tar.gz`. Aucune ressource HTML/CSS que vous devez produire ne doit être remise (par soucis de bande passante). Le projet est à remettre en groupe d'au plus deux personnes **au plus tard dimanche 20 février à 23h59**.

Questions

Posez vos questions via le slack du cours sur le canal `demo` si votre question peut être d'intérêt pour tout le monde, ou en message individuel direct (au démonstrateur Yifan Bai (Andy)) sinon. Prenez le temps de lire le sujet avant de poser des questions. Je ne répondrai plus aux questions de compréhension trois jours avant la date butoir.