1. Rappels généraux

Tous les TD et TP se dérouleront sur **Linux**. Nous allons travailler avec plusieurs outils qui dépendent de **Docker**.

Le but est de découvrir le test automatique d'interfaces Web avec `Robot Framework` et son plugin `Selenium`.

- `Robot Framework` est un automate logiciel, c'est à dire un moteur pour exécuter des scripts. Ces script permettent de faire des tests automatisés sur tout ce qui est interface graphique, API REST, et aussi bases de données.
- Le plugin `Selenium` relie `Robot Framework` à un navigateur internet comme Firefox ou Chrome. Il se présente sous la forme d'une librairie dans `Robot Framework`.

Avec ces outils, on prépare un script de test qui déroule une séquence d'actions sur un navigateur, comme ouvrir tel URL, chercher tel bouton, cliquer dessus, vérifier qu'on a tel contenu, etc. On crée ainsi une suite de tests fonctionnels qui garantissent les grandes fonctions du logiciel, celles que verra l'utilisateur final.

2. Préparatifs

Nous allons travailler avec une image Docker qui contient Firefox et Chrome en version headless (sans interface visible). Cette image Docker contient aussi `Robot Framework` ainsi que `Selenium`.

1. Ouvrez un shell et tapez les commandes suivantes (utilisez le bouton ci-contre ⟶) :

```
docker pull robot-selenium
cd /Docker/$USER
wget https://perso.univ-rennes1.fr/pierre.nerzic/TestsCI/fichiers/tp6.zip
unzip tp6.zip
chmod +x tp6/*/*.{sh,pyz}
cd tp6/testsRobotFramework
export http_proxy='http://129.20.239.9:3128'
export https_proxy='http://129.20.239.9:3128'
```

Si vous êtes sur votre PC linux personnel, placez-vous d'abord dans un dossier approprié, et voici les commandes à taper :

```
docker pull melnibon/robot-selenium
wget https://perso.univ-rennes1.fr/pierre.nerzic/TestsCI/fichiers/tp6.zip
unzip tp6.zip
chmod +x tp6/*/*.{sh,pyz}
cd tp6/testsRobotFramework
```

3. Découverte de Robot Framework

Il y a plusieurs sous-dossiers dans `tp6`, chacun pour un type de tests. Dans chaque dossier, il y a un script de lancement `run_robot.sh` et un sous-dossier `tests`. Les scripts `*.robot` sont dans ce dernier.

3.1. Salutations

1. Allez dans `tp6/testsRobotFramework`.
2. Affichez le fichier `tests/TestBonjour.robot`. Essayez de comprendre ce qu'il fait.
3. Affichez le fichier `TestFichiers.robot`. C'est un ensemble de tests sur des dossiers et fichiers ; ici, ceux des tests. Attention, dans ce TP, `Robot Framework` tourne dans un *container* Docker, et donc ce sont les dossiers et fichiers à *l'intérieur du container*.

Les tests sont définis à l'aide de mot-clés comme `File Should Not Exist` et `Directory Should Exist` qui viennent de la librairie `OperatingSystem`.

4. On va maintenant exécuter ces tests, en restant dans le dossier `tp6/testsRobotFramework` : 

```
./run_robot.sh
```

Le script `run_robot.sh` contient la commande et toutes les options pour lancer l'image `robot-selenium`, et vous devriez voir à peu près ceci :

```
.../tp6/testsRobotFramework$ ./run_robot.sh
==============================================================================
Tests
==============================================================================
Tests.TestBonjour
==============================================================================
Bonjour du robot                                .Vas voir dans le dossier reports
Bonjour du robot                                                        | PASS |
------------------------------------------------------------------------------
Tests.TestBonjour                                                       | PASS |
1 test, 1 passed, 0 failed

... idem avec Tests.TestFichiers

==============================================================================
Tests                                                                   | PASS |
3 test, 3 passed, 0 failed
==============================================================================
Output:  /projet/reports/output.xml
Log:     /projet/reports/log.html
Report:  /projet/reports/report.html
```

5. Ouvrez le fichier `reports/report.html` avec un navigateur, c'est là que `Robot Framework` a déposé son compte-rendu. En bas de la page, sous `Test Details`, cliquez sur l'onglet `All`, puis sur le nom `Bonjour du robot`. Vous aurez le détail des étapes qui ont réussi et celles qui ont échoué, et les raisons. Relisez le script de test pour tout comprendre et vous imprégner de la syntaxe.

3.2. Tests sur des fichiers

1. Ouvrez le fichier `tests/TestFichiers.robot` dans un éditeur.

2. Ouvrez ce lien sur la doc et lisez un peu. Il faut s'habituer à ce type de documentations. Elle est très complète mais un peu bizarre au premier abord. Cherchez `Example` pour avoir une idée du mode d'emploi, puis lisez la doc du mot-clé `Directory Should Exist`.

Attention à ne pas confondre `Directory Should Be Empty` et `Empty Directory`. Allez voir leur description pour savoir pourquoi.

3. Ajoutez un test qui affirme que le fichier `tests/TestFichiers.robot` doit exister (cherchez le mot-clé adéquat).
4. Ajoutez ce test :

```
Fichier tests/TestFichiers.robot doit contenir au moins 400 octets
    ${taille} =  Get File Size  tests/TestFichiers.robot
    Should Be True  ${taille} >= 400
```

Ce test montre comment on affecte une variable et comment on peut tester des assertions dessus.

- La syntaxe est `${nomvar} =` suivi de deux espaces suivi d'un mot-clé qui retourne une valeur, ici `Get File Size`
- Les assertions `Should Be True` sont prédéfinies (*built-in*). Voici la documentation de ces mots-clés : librarie Builtin.

5. Ouvrez ce lien sur la doc et lisez un peu. Il y a énormément de choses (107 mot-clés). Allez directement voir `Should Be True` et `Should Be Equal`, ce sont les deux mot-clés les plus utiles au début.
6. Ajoutez un test qui affirme qu'il y a exactement deux fichiers (cherchez le mot-clé adéquat). Vous allez vous battre avec la comparaison. La documentation du mot-clé qui compte les fichiers dit : « *The count is returned as an integer, so it must be checked e.g. with the built-in keyword* `Should Be Equal As Integers`. »

4. Test d'un site Web

Pour découvrir les concepts des tests fonctionnels sur interface, on va utiliser Wikipédia.

4.1. Premier test

1. Allez dans `tp6/testsWikipedia`.

Il y a un fichier `tests/TestWikipedia.robot`, et il y a `run_robots.sh` pour lancer l'exécution, et aussi `Wikipedia-proxy-cache.pyz`.

2. Avant de lancer l'exécution, il faut ouvrir un autre onglet dans le terminal, ou un autre terminal dans ce dossier, et taper cette commande :

```
./Wikipedia-proxy-cache.pyz
```

Ce script gère un *cache* pour Wikipedia, ce qui évite de se faire bloquer (*blacklisting*) à cause du trop grand nombre de requêtes en rafale sur les mêmes pages. Il permet au container Docker d'accéder à Wikipedia via `host.docker.internal` sur le port 8000. NB: `host.docker.internal` est le nom interne au *container* de `localhost`.

Si vous êtes chez vous, vous pouvez vous passer de ce script et alors, il faut échanger les commentaires des paires de variables `${URL_...}` pour accéder directement à Wikipedia.

3. Lancez les tests avec `./run_robot.sh`
4. Ouvrez `reports/report.html` pour les résultats. Gardez-le ouvert pour tous les lancements suivants (`F5` pour rafraîchir).
5. Essayez de comprendre le fichier `tests/TestWikipedia.robot`. Il y a plusieurs sections.

- La section `Keywords` permet de définir de nouveau mots-clés. Les mots-clés comme `Open Browser` et `Set Windows Size` sont documentés SeleniumLibrary. Cherchez `Open Browser` dans l'index de gauche. Vous voyez la liste des paramètres (*Arguments*) comme `url`, `browser`, etc. Ici, `browser=headlessfirefox`, un navigateur sans interface, comme un processus d'arrière-plan.
- La section `Settings` définit `Suite Setup` et `Suite Teardown`, ce qui doit être fait au début de tous les tests, et tout à la fin.
- La section `Variables` affecte des variables globales, taille de la fenêtre et URLs à utiliser. Les versions accédant directement à Wikipedia sont commentées.
- La section `Test Cases` contient tous les tests. Cherchez `Go To` et `Get Title`. Le mot-clé `Should Contain` n'est pas documenté ici, car il est prédéfini. La ligne `Wait Until Location Is` permet d'attendre le chargement complet de la page, très lent la première fois. Une fois qu'elle est en cache, c'est très rapide.

6. Dans le *keyword* `Setup Browser` commentez les deux premières lignes et décommentez la 3e, celle de Firefox *headless*, juste pour voir si ça fait une différence entre Firefox et Chrome.
7. Vous pouvez aussi changer la taille de la fenêtre définie par `WINSIZE`.

4.2. Recherche

On va lancer une recherche, comme le ferait un humain. Mais avant, il faut vérifier qu'il y a une zone de saisie pour la recherche et le bouton de lancement.

1. Ajoutez le test suivant dans `TestWikipedia.robot` :

```
Test Zone de recherche et bouton dans la page d'accueil
    Go to  https://www.wikipedia.fr/
    Wait Until Location Is  https://www.wikipedia.fr/
    Page Should Contain Element  //input[@id="search" and @type="texte"]
```

Que fait ce test, en particulier la dernière ligne ? Le mot-clé `Page Should Contain Element` est de `SeleniumLibrary`. Son paramètre est appelé *locator*. Il désigne l'élément concerné par l'action par son identifiant, son titre, etc. Ici, c'est avec une requête XPath (cours n°2). Il existe des simplifications, par exemple `id="search"` aurait suffi.

2. Que signifie la syntaxe XPath `//input[@id="search" and @type="texte"]` ? Elle cherche un élément `<input id="search" type="texte"/>` dans le DOM de la page.
3. Ouvrez manuellement https://www.wikipedia.fr/ et utilisez l'inspecteur pour trouver ce qui caractérise le `<input>` où on tape le texte à rechercher. Est-ce tout à fait correct ? *texte* ?? Corrigez pour que le test réussisse.
4. Changez le mot-clé `Page Should Contain Element` en `Element should be visible` qui est un peu plus strict. Relancez les tests.

5. Ajoutez une ligne pour vérifier qu'il y a bien une balise `<img id="search-icon">` et ajoutez une condition sur l'attribut `alt` de cette image.
6. Maintenant, on va saisir un texte et cliquer sur le bouton. Ajoutez le test suivant dans `TestWikipedia.robot` :

```
Test Recherche de "Robot Framework" sur Wikipédia
    Go to  https://www.wikipedia.fr/
    Wait Until Location Is  https://www.wikipedia.fr/
    Input Text  //input[@id="search"]  Robot Framework
    Click Element  //img[@id="search-icon"]
    Location should be  https://fr.wikipedia.org/wiki/Robot_Framework
    Page Should Contain  Robot Framework est un framework de test automatique
    Capture Page Screenshot
```

7. Ouvrez le fichier `tp6/testsWikipedia/reports/report.html`, affichez le détail de ce test et regardez la copie écran.
8. Allez voir les mots-clés que vous ne connaissez pas dans la documentation.

4.3. Liens hypertexte

On va tester les liens hypertextes : `<a href="...">texte</a>`.

1. Ajoutez et complétez le test suivant dans `TestWikipedia.robot` :

```
Test Clic sur le lien "python" dans la page "Robot Framework"
    Go to  https://fr.wikipedia.org/wiki/Robot_Framework
    Wait Until Location Is  https://fr.wikipedia.org/wiki/Robot_Framework
    Click Element  L'ÉLÉMENT <A> DONT LE TEXTE EST "python"
    Log Location
    Capture Page Screenshot
```

Le principe est de cliquer sur le lien comme le ferait un humain, pas d'aller directement à l'URL sous-jacent. La requête XPath pour trouver `<ELEMENT>CONTENU</ELEMENT>` s'écrit `//ELEMENT[text()="CONTENU"]`.

2. Vérifiez la copie écran dans `tp6/testsWikipedia/reports/report.html`.

4.4. Menu déroulant

Maintenant, on va afficher la page sur `Robot Framework` en anglais en utilisant, comme le ferait un humain, le menu déroulant de la page. C'est un `<input>` que le script doit cliquer. Cela affiche un panneau avec les principales langues : un `<div>` contenant des `<li>` situés au début du document. Donc commencez par effectuer la manipulation à la main, en inspectant les éléments HTML concernés afin de repérer les types et identifiants des éléments.

1. Voici le squelette du test, à vous de le compléter :

```
Test Page "Robot Framework" en anglais
    Go to  https://fr.wikipedia.org/wiki/Robot_Framework
    Wait Until Location Is  https://fr.wikipedia.org/wiki/Robot_Framework
```

```
    Click Element  LE <input> DU MENU LANGUES
    Wait until element is visible  LE <li> DONT L'ATTRIBUT TITLE EST "English"
    Capture Page Screenshot
    # A VOUS DE JOUER...
    Location should be  A VOUS DE COMPLETER...
    Capture Page Screenshot
```

Remarquez le mot-clé `Wait until element is visible` pour vérifier la visibilité d'un élément.

2. Vérifiez les deux copies écran dans `tp6/testsWikipedia/reports/report.html`.

5. Test d'une application Web

Voici maintenant un mini-projet. C'est un formulaire de connexion directement tiré de https://www.w3schools.com/howto/howto_css_login_form.asp. La page d'accueil montre un bouton pour se connecter. Ça fait afficher un formulaire, on entre son nom et son mot de passe, on clique sur un bouton et on arrive dans une autre page. Le but est de tester entièrement toutes ces fonctionnalités avec `Robot Framework/Selenium`.

5.1. Préparatifs

1. Allez dans `tp6/testsPasswordForm`.

Il y a un logiciel PHP constitué d'un formulaire.

2. Lancez ce logiciel par `php -S 0.0.0.0:8100`. L'adresse IP autorise toute machine à ouvrir http://localhost:8100. C'est nécessaire pour Docker, mais il n'y a aucune sécurité. Laissez tourner `php`.
3. Ouvrez http://localhost:8100 dans le navigateur, appuyez sur le bouton `Login`, remplissez le formulaire, mais surtout **ne mettez pas** vos véritables identifiants !

Gardez PHP et le navigateur ouverts afin de pouvoir inspecter le formulaire.

5.2. Tests à compléter

Le but est de tester le logiciel. Il affiche un formulaire et des réponses qu'on va vérifier. Les tests sont dans `tests/TestPasswordForm.robot`.

1. Lancez `./run_robot.sh`. Affichez ensuite `reports/report.html` et gardez-le ouvert pour tous les lancements suivants. Il suffira de rafraîchir la page après chaque lancement.
2. Il y a plusieurs tests partiellement programmés dans `tests/TestPasswordForm.robot`. Complétez-les.

6. Rendu du TP

Vous devrez remettre votre travail à la fin de chaque séance. C'est ainsi que c'est noté, en contrôle continu.

Ouvrez un terminal bash :

```
cd /Docker/$USER
rm -fr tp6/*/reports
tar cfvz tp6.tgz tp6
```

Sur votre PC personnel, supprimez les dossiers `reports` avant de compresser `tp6`.

Puis déposez le fichier `tp6.tgz`, **ATTENTION, pas le tp6.zip fourni initialement**, dans la zone de dépôt du TP6 sur Moodle R4.02 Qualité de développement.

ATTENTION votre travail est personnel. Si vous copiez pour quelque raison que ce soit le travail d'un autre, vous serez tous les deux pénalisés par un zéro.

En cas de souci quelconque, envoyez un mail à `pierre.nerzic@univ-rennes1.fr` pour expliquer le problème.