



# COMPTE-RENDU DE STAGE

GALLATO Peyo

BTS SIO stage première année

## Sommaire

1.	Contexte de l'entreprise.....	2
2.	Sujet du stage .....	2
3.	Tâches effectuées pendant le stage et outils utilisés.....	3
a.	Création d'un robot de test .....	3
b.	Application du nouveau socle Playwright sur une première application CNP : PUMA .....	4
c.	Application du nouveau socle Playwright sur une seconde application CNP : @DELE .....	4
d.	Application du nouveau socle Playwright sur une troisième application CNP : Mercure .....	4
e.	Documentation Playwright.....	6
f.	Modification du script d'installation pour y ajouter Playwright.....	7
g.	Renforcement des tests unitaires des socles Playwright et Selenium .....	7
h.	Ajout de la librairie Appium .....	8
4.	Les difficultés rencontrées et les solutions apportées.....	8
a.	Pour la création du robot de test du formulaire de souscription auto .....	8
b.	Pour l'application du socle Playwright sur l'application PUMA .....	9
c.	Pour l'application du socle Playwright sur l'application @DELE.....	9
d.	Pour la création d'un nouveau robot sur l'application Mercure .....	9
e.	Ajout de la librairie Appium .....	10
f.	Pour le travail avec l'équipe.....	11
5.	Bilan .....	11

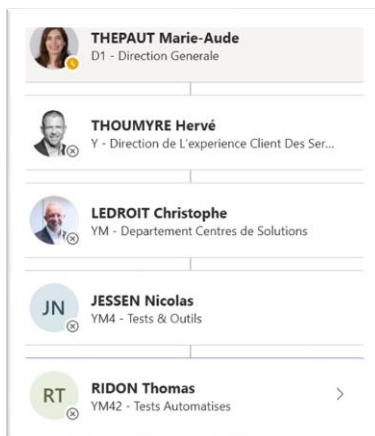
## REMERCIEMENTS

Je tiens à remercier la CNP et tout particulièrement Monsieur Thomas RIDON pour m'avoir fait confiance et accueilli dans son équipe. Ses remarques et ses conseils m'ont beaucoup aidé tout au long du stage et m'ont permis d'acquérir de nouvelles compétences qui me seront utiles pour mes études et ma future carrière.

## 1. Contexte de l'entreprise

CNP Assurances est un acteur de référence de l'assurance de personnes et de biens, membre du grand pôle financier public français. Le groupe est présent dans 19 pays, en Europe et en Amérique latine. L'entreprise est une filiale du Groupe La Banque Postale et compte près de 7000 collaborateurs dans le monde. Les solutions de CNP Assurances couvrent le champ complet de l'assurance de personnes et de biens (assurance emprunteur, assurance vie, la santé et la prévoyance, la retraite et les assurances de biens ou IARD (Incendie, Accidents et Risques Divers)).

L'équipe dans laquelle je me trouve est l'équipe *Tests automatisés*. Cette équipe fait partie du service *Tests et Outils* qui est dans le département Centre de Solutions sous la direction de l'expérience client, des services numériques et de la donnée (DECSND).



1 Organigramme équipe Tests automatisés

## 2. Sujet du stage

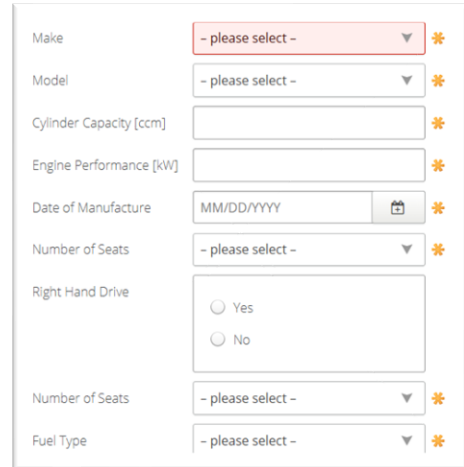
Le stage consiste à mettre en place une librairie Playwright dans le socle automatisé écrit actuellement en Robot Framework afin de remplacer l'actuelle librairie Selenium. La CNP a souhaité passer à cette nouvelle librairie car certains robots ne fonctionnaient pas totalement avec la librairie Selenium notamment quand les applications à tester contenaient des pop-ups. De ce fait, les applications ne pouvaient pas être testées en mode automatisé ce qui obligeait à faire des tests manuels. Ces tests manuels pouvaient prendre du temps. Il faut noter que la nouvelle librairie mise en place ne va pas totalement remplacer l'ancienne. Elle va simplement permettre d'avoir une deuxième option si nécessaire.

Une fois la mise en place effectuée, un second objectif (si le temps le permet) sera de renforcer les tests unitaires existants pour la librairie Selenium et les adapter si nécessaire à la librairie Playwright en vue de mettre en place un pipeline de déploiement à chaque push du socle automatisé sur les machines d'exécutions en cas de tests concluants.

### 3. Tâches effectuées pendant le stage et outils utilisés

#### a. Création d'un robot de test

La première tâche que j'ai faite pendant mon stage a été de créer un robot permettant d'automatiser un formulaire web de souscription à une assurance voiture. Ce formulaire contenait différents types d'input à renseigner (liste déroulante, case à cocher, bouton radio, champs textes et des boutons d'actions). Avant de pouvoir démarrer j'ai passé une demi-journée à installer les différents outils dont j'aurais besoin (Git Bash, Visual Studio Code).

Un extrait d'un formulaire web de souscription à une assurance voiture. Le formulaire contient plusieurs champs : 'Make' et 'Model' sont des listes déroulantes avec '- please select -' ; 'Cylinder Capacity [ccm]' et 'Engine Performance [kW]' sont des champs de texte ; 'Date of Manufacture' est un champ de date avec un calendrier ; 'Number of Seats' est une liste déroulante ; 'Right Hand Drive' est un groupe de boutons radio avec 'Yes' et 'No' ; 'Fuel Type' est une liste déroulante. Des icônes d'étoile sont présentes à droite de certains champs.

2 extrait du formulaire de souscription

Le robot a été créé à l'aide du langage RobotFramework. La première version que j'ai faite utilisait la librairie Selenium. Le robot ouvre le navigateur sur la page du formulaire puis renseigne les champs présents. Ensuite j'ai fait une seconde version du robot avec la librairie Playwright en remplaçant les fonctions Selenium par des fonctions Playwright. L'objectif du stage étant de remplacer la librairie Selenium par la librairie Playwright, cela m'a permis de voir les différences entre les deux et de comprendre le fonctionnement de chacune des librairies.

Une des différences entre les deux librairies est que Selenium ouvre la même version du navigateur qu'un utilisateur lambda (navigateur déjà installé sur le poste de travail). Alors que Playwright ouvre une instance modifiée du navigateur qui lui permet d'interagir plus facilement avec les éléments d'une page ou du navigateur.

La seconde version du robot avec la librairie Playwright était un peu plus simple parce qu'elle avait une meilleure gestion des timeouts pour le contexte particulier de ce robot.

Les robots déjà en place utilisent un socle technologique Selenium que j'ai pu récupérer avec l'outil GitLab. Ce socle contient différentes fonctions utilisées par les robots existants. J'ai donc fait une 3<sup>ème</sup> version du robot pour utiliser les fonctions du socle Selenium puis j'ai dupliqué le socle Selenium pour le transformer en socle Playwright. J'ai d'abord remplacé les fonctions utilisées par mon robot en Playwright. Puis, une fois mon robot fonctionnel, je me suis attaqué aux autres fonctions que je n'utilisais pas encore. Pour être sûr que les fonctions que je remplaçais

```
623 Cliquer Sur Element
624 [Arguments]    ${locator}
625 ...            ${visible}
626 ...            ${javascript}
627 ...            ${attente}
628 ...            ${timeout}
629 ...            ${error}
630
631 selenium2.Confirmer Page Prete
632 TRY
633     SeleniumLibrary.Wait Until Page Contains Element    ${locator}    ${timeout}    ${error}
634     SeleniumLibrary.Wait Until Element Is Enabled    ${locator}    ${timeout}    ${error}
635     IF    ${visible}
636     ... SeleniumLibrary.Wait Until Element Is Visible    ${locator}    ${timeout}    ${error}
637 EXCEPT    ${except_stale_element}    ${except_unknown_error}    type=start
638     SeleniumLibrary.Wait Until Page Contains Element    ${locator}    ${timeout}    ${error}
639     SeleniumLibrary.Wait Until Element Is Enabled    ${locator}    ${timeout}    ${error}
640     IF    ${visible}
641     ... SeleniumLibrary.Wait Until Element Is Visible    ${locator}    ${timeout}    ${error}
642 END
643
644 TRY
645     IF    ${javascript}
646     IF    $attente Sleep    ${attente}
647     ${element}= SeleniumLibrary.Get WebElement    ${locator}
648     SeleniumLibrary.Execute Javascript    arguments[0].click();    ARGUMENTS    ${element}
649     ELSE
650     IF    $attente Sleep    ${attente}
651     TRY
652     SeleniumLibrary.Click Element    ${locator}
653     EXCEPT    ${except_stale_element}    ${except_unknown_error}    type=start
654     SeleniumLibrary.Click Element    ${locator}
655     END
656     END
657 EXCEPT AS    ${exception}
658     # Permet de renvoyer l'erreur attendue
659     IF    $error != $None
660     FAIL    ${error}
661     END
662     FAIL    ${exception}
```

3 - Extrait du socle Selenium dans Visual Studio Code

étaient fonctionnelles, j'ai utilisé une suite de test unitaires permettant de tester la plupart des fonctions. J'ai dû adapter certains des tests pour les faire fonctionner avec Playwright notamment ceux qui vérifiaient un message d'erreur.

#### b. Application du nouveau socle Playwright sur une première application CNP : PUMA

Pendant ma seconde semaine de stage, maintenant que j'avais un socle Playwright, mon tuteur m'a demandé de l'appliquer à une application utilisée par CNP Assurances. L'application s'appelle PUMA, elle est mise à disposition des conseillers de la caisse d'épargne pour le suivi des produits d'assurances vie, prévoyance et retraites. Cette application m'a amené à lire et comprendre l'algorithme d'un robot déjà existant pour pouvoir le modifier et le rendre compatible avec mon nouveau socle. Elle m'a permis de continuer à valider ou non le nouveau socle que j'avais mis en place. Cela m'a pris un peu plus d'une semaine.

#### c. Application du nouveau socle Playwright sur une seconde application CNP : @DELE

Après PUMA, j'ai travaillé sur un autre robot pour l'application @DELE. Ce robot est une des raisons de l'existence de mon stage car lors de l'authentification, une pop-up apparait et cette pop-up ne peut pas être gérée par Selenium. On m'a donc donné accès à l'application pour faire des tests avant que je ne commence à travailler sur l'adaptation du robot.

En démarrant les tests avec Playwright, j'ai découvert que Playwright n'affichait jamais les pop-ups. En effet, avec Playwright, il faut gérer les pop-ups avant leur apparition. C'est de cette manière que Playwright peut faire une action sur la pop-up. Par contre, si aucune action n'est prévue, Playwright n'affiche simplement pas la pop-up. Dans le cas d'@DELE, la pop-up devait simplement être ignorée, donc je n'ai rien eu à modifier pour régler ce problème. Par contre le robot n'avait pas été lancé depuis plus de 6 mois donc il était devenu obsolète sur certains points. J'ai donc dû enlever un bout du code et modifier quelques fonctions pour le rendre de nouveau fonctionnel.

Au final, j'ai travaillé 3 jours sur cette application pour adapter le robot qui m'a aussi permis de valider le bon fonctionnement du socle Playwright.

#### d. Application du nouveau socle Playwright sur une troisième application CNP : Mercure

Après avoir terminé la modification du robot pour @DELE, mon maître de stage m'a confié la mission de développer un nouveau robot pour l'application Mercure. Le robot était une demande du projet pour automatiser une partie de leurs tests. Le robot consiste à tester la fonctionnalité de mise en place d'un Versement Régulier. Pour créer ce robot, j'ai eu comme entrant des fichiers PDF contenant des captures d'écran de l'application fournies par le projet.

A la CNP, les robots font tous appel au même fichier. Ce fichier fait ensuite appel au Socle (Selenium ou Playwright en fonction de l'exécution). Par exemple, si le robot doit exécuter une fonction, il appelle le fichier web2.robot. Ce fichier permet de faire le lien entre le robot et le Socle. Il permet aussi d'avoir des paramètres de base et de ne pas avoir à saisir tous les arguments de la fonction.



```

web2.robot 1 X playwright.robot 4 ●
tnr_socle_automatisation > resources > web2.robot > ...
Load in Interactive Console
446 Effacer Dans Champ
447 [Documentation] Efface du texte dans un champ.
448
449 [Arguments] ${locator}
450 ... ${timeout}=${None}
451 ... ${erreur}=${None}
452
453 [Tags] Socle
454
455 Run Keyword ${web_library}.Effacer Dans Champ locator=${locator}
456 ... timeout=${timeout}
457 ... erreur=${erreur}
458
  
```

4 - Extrait du fichier web2.robot 'Effacer Dans Champ'

`${web_library}` est défini au lancement du robot. C'est ce qui définit quel socle doit être appelé (soit selenium soit playwright).

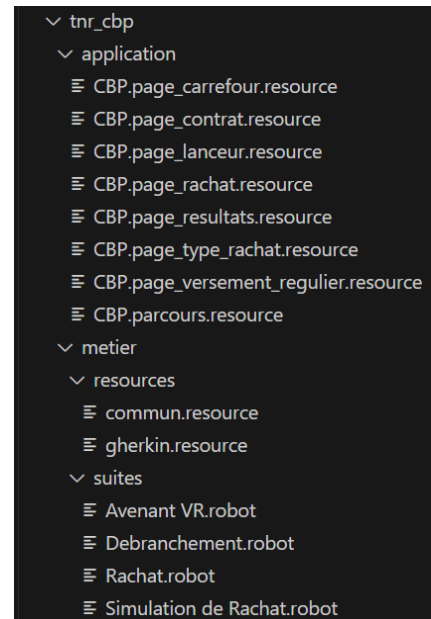
```

web2.robot 1 playwright.robot 4 ●
tnr_socle_automatisation > resources > playwright.robot > ...
Load in Interactive Console
538 Effacer Dans Champ
539 [Arguments] ${locator}
540 ... ${timeout}=${None}
541 ... ${erreur}=${None}
542
543 playwright.Confirmer Page Prete
544
545 Browser.Wait For Elements State ${locator}
546 ... state=attached
547 ... timeout=${timeout}
548 ... message=${erreur}
549
550 Browser.Wait For Elements State ${locator}
551 ... state=enabled
552 ... timeout=${timeout}
553 ... message=${erreur}
554 TRY
555 | Browser.Clear Text ${locator}
556 | EXCEPT AS ${exception}
557 | # Permet de renvoyer l'erreur attendue
558 | IF $erreur != $None
559 | | FAIL ${erreur}
560 | END
561 | FAIL ${exception}
562 END
  
```

5 - Extrait de playwright.robot 'Effacer Dans Champ'

Les robots ont tous la même architecture de fichier :

Le fichier qui est appelé pour lancer le robot est dans le dossier *suites* en fonction du robot que l'on veut appeler. Ce fichier appelle ensuite le fichier *commun.resource*. Il contient des fonctions communes à tous les robots de l'application comme les fonctions à lancer avant et après le test ou les fonctions de gestion d'erreur pendant l'exécution. On a ensuite le fichier *gherkin.resource*. C'est ce fichier qui définit le chemin que le robot emprunte. Il appelle le fichier *CBP.parcours.resource*. Ce fichier contient la totalité des parcours exécutés par tous les robots. Il appelle les autres fichiers du même dossier en fonction de la page où l'on est rendu. Cette architecture permet de retrouver plus facilement l'emplacement d'une fonction lorsqu'il y a besoin de faire une modification ou s'il faut ajouter une fonction pour une page dans une application.



6 - architecture des robots de l'application Mercure

Pour créer le robot, je suis d'abord 'parti de rien' en créant des fonctions de base avant de me rendre compte que je pouvais les récupérer d'autres robots existant pour cette application notamment les fonctions de la page lanceur qui sont communes à tous les robots. Une fois que je m'en suis rendu compte, j'ai pu aller plus vite car je pouvais réutiliser des fonctions qui existaient déjà. Une fois toutes les fonctions de bases récupérées puis adaptées à mon robot, j'ai dû développer toutes les parties spécifiques au Versement Régulier.

J'ai également dû créer le fichier Excel nécessaire au robot. En effet les robots sont alimentés en Jeux de données via un fichier Excel que les testeurs modifient en fonction du cas de test qu'ils veulent exécuter. J'ai donc créé le fichier et je l'ai renseigné avec les données de tests qui étaient visibles dans les PDF fournis par le projet. Pour ce fichier, je me suis appuyé sur un fichier Excel qui avait déjà été créé par le projet et qui contenait tout ce que le projet souhaitait modifier dans l'exécution du robot. Ce fichier m'a donc aidé à faire mon fichier Excel. Cependant, le fichier initial ne contenait pas toutes les informations nécessaires pour bien faire fonctionner mon robot. J'ai donc eu à regarder dans les fichiers Excel des autres robots existants pour reprendre certaines informations utiles au bon fonctionnement de mon robot.

Pour créer ce robot, j'ai travaillé 2 semaines. Une fois le robot terminé et fonctionnel, j'ai fait une démo aux test managers qui utiliseront ce robot pour leur montrer le fonctionnement, comment remplir le fichier Excel et les petits bonus que j'avais pu ajouter grâce à Playwright et qui n'étaient pas disponibles avec Selenium. Durant cette démo, je me suis rendu compte qu'il y avait un cas que je n'avais pas encore traité et je l'ai donc rajouté par la suite.

#### e. Documentation Playwright

En prévision de la reprise de mes travaux, j'ai créé une page Confluence qui explique comment installer Playwright et qui détaille les différences entre le socle Selenium et le socle Playwright. Confluence est utilisé à la CNP pour centraliser toutes les informations et la documentation.

Dans les deux socles, les fonctions ont les mêmes noms. Cependant il peut exister des différences et la page confluence explique ces différences, qui peuvent être de deux ordres :

1. Certaines fonctions renvoient une réponse identique mais leurs algorithmes sont différents
2. Certaines fonctions renvoient une réponse différente et il faut se méfier lors de l'appel. Ces fonctions sont souvent moins utilisées par les robots car elles ont des actions plutôt précises qui ne sont pas modifiables.

## Playwright

Créée par RIDON Thomas, dernière modification par GALLATO Peyo le 12 juin 2025

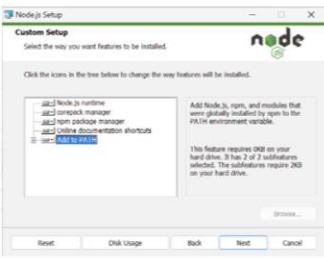
### Installation

Pour utiliser la librairie Playwright, il faut installer Node.js.

Pour cela, aller sur le site officiel de [Node.js](#) et télécharger l'installateur Windows.

Lors de l'installation, sélectionner "Add to PATH" et choisir "Next" à chaque fois.

Voir la page [Mise en route](#) pour l'installation complète.



### Différences notables avec Selenium

#### Mots Clés

- **Déterminer Répertoire Captures Ecran** => défini une variable de test qui est réutilisée dans Capturer Ecran pour enregistrer les captures d'écran
- **Capturer Ecran** => permet de capturer la page complète sans modifier le zoom du navigateur. Disponible seulement pour Playwright. A chaque capture, augmente une variable de test qui compte le numéro de la capture. Modifie le numéro pour qu'il soit sous le format 001. Renomme le fichier pour qu'il est le même format de nom que Selenium. /!\ Il arrive que la capture échoue (Timeout). Ce n'est pas une erreur bloquante mais un Warning sera logger dans la console. /!\
- **Sélectionner Frame** => Ne marche qu'avec un Locator contenant un id.  
Exemple : `//*[@id="frameExample"]`
- **Saisir Dans Champ** => Pour saisir un texte en JavaScript, il faut nécessairement utiliser un xpath.
- **Rechercher Une Éléments** => Playwright utilise l'identifiant de l'élément pour chercher l'élément. Prendre chaque élément ayant le même identifiant et vérifier si la liste est vide.

7 - Extrait de la documentation confluence

#### f. Modification du script d'installation pour y ajouter Playwright

Lors de mon arrivée, j'ai lancé un script d'installation qui m'a permis d'avoir le même environnement que toutes les autres machines, afin d'éviter toute difficulté lors du lancement des robots. Ce script installait toutes les librairies nécessaires, à l'heure actuelle, au bon fonctionnement des robots. Il a donc fallu le modifier pour y intégrer l'installation de la librairie Playwright. Cette dernière repose sur des dépendances liées à Node.js, qui doit donc être préalablement installé. L'extrait de script vérifie si Node.js est présent. S'il l'est, alors le script procède à l'installation des dépendances nécessaires à Playwright.

```

40 where node >nul 2>nul
41 if %errorlevel% == 0 (
42     echo Installation des dependances playwright
43     rfbrowser init 2>nul
44     if %errorlevel% neq 0 (
45         echo rfbrowser n'est pas trouve, tentative avec Python...
46         python -m Browser.entry init
47     )
48 else (
49     echo Node.js n'est pas installe
50 )

```

8 - Extrait du script d'installation

#### g. Renforcement des tests unitaires des socles Playwright et Selenium

La dernière mission comprise dans le sujet du stage consiste à renforcer les tests unitaires des socles Playwright et Selenium. Au départ, il y avait une suite de 88 tests unitaires uniquement pour Selenium. Au début du stage, je les avais dupliqués dans le but de vérifier mon socle Playwright. Après la création du robot pour Mercure, j'ai ajouté de nouveaux tests pour le socle Playwright. Je suis passé de 88 à 180 tests unitaires. Mon but dans ces nouveaux tests était de tester chaque fonctionnalité de toutes les



fonctions pour m'assurer du bon fonctionnement et aussi pour permettre de détecter des potentielles régressions par la suite. L'ajout de ces tests m'a permis de détecter plusieurs bugs dans des fonctions qui étaient rarement utilisées. J'ai ensuite dupliqué les tests que j'avais rédigés pour Playwright afin de les ajouter aux tests unitaires de la librairie Selenium. Ces ajouts m'ont aussi permis de me rendre compte de certaines erreurs présentes dans le socle Selenium. Ces erreurs étaient pour la plupart des mauvaises gestions de message d'erreurs. Ce n'étaient donc pas des erreurs critiques ce qui explique qu'elles n'avaient pas été détectées jusque-là.

#### h. Ajout de la librairie Appium

A ce stade du stage, j'avais fini le sujet principal du stage. Mon tuteur m'a donc proposé de créer un socle Appium, comme je l'avais fait pour le socle Playwright. Appium est une librairie qui permet de faire des tests WEB mais sur mobile. Cette librairie se base sur la librairie Selenium. Elle a donc des similitudes avec ce qui existait déjà pour Selenium. Un socle Appium existait déjà mais il était inutilisé et globalement hors d'usage donc mon tuteur m'a demandé de repartir de zéro pour refaire un socle propre et compatible avec les autres tests. J'ai implémenté tous les mots clés qui était disponibles directement avec la librairie mais je n'ai pas eu le temps d'aller plus loin. Le socle n'est donc pas totalement terminé. Pour l'instant, il n'y a pas de test sur mobile à la CNP mais cela pourrait arriver assez rapidement. J'ai donc préparé le terrain pour ensuite faciliter la finalisation du socle. Ainsi, mon maître de stage pourra passer moins de temps sur ce socle. De plus, cela a permis de d'ores et déjà découvrir plusieurs difficultés liées à cette librairie.

### 4. Les difficultés rencontrées et les solutions apportées

#### a. Pour la création du robot de test du formulaire de souscription auto

Lors de la création de ma première version de robot avec la librairie Selenium, plusieurs tentatives ont été nécessaires pour compléter le formulaire correctement. Je n'ai pas eu de difficulté particulière si ce n'est un temps de chargement qui engendrait un Timeout. Pour résoudre ce problème, j'ai un peu cherché dans la documentation et j'ai pu me débloquent.

J'ai également rencontré quelques problèmes dans la réécriture du socle avec la librairie Playwright. Tout d'abord, les fonctions du socle Playwright doivent reproduire le même comportement que les fonctions du socle Selenium mais ce n'est pas possible pour la totalité des fonctions à cause de la différence de fonctionnement des librairies. Certaines fonctions n'ont donc pas le même comportement entre les deux socles. Un exemple de ce fonctionnement différent est sur les pop-ups. Avec Selenium, les pop-ups sont gérées après leur apparition tandis qu'avec Playwright, les pop-ups doivent être gérées avant leur apparition. C'est un problème car le code des robots existants gère les pop-ups après l'apparition et ne sera donc pas compatibles avec les deux librairies. De plus, certains tests unitaires vérifiaient des erreurs renvoyées par les fonctions mais les deux librairies ne renvoient pas du tout les mêmes erreurs donc j'ai dû modifier les messages d'erreurs attendus. Enfin, le code Selenium contenait beaucoup de paramètres spécifiques à Selenium qui ne peuvent pas être reproduits avec Playwright. Certains des paramètres n'étaient simplement pas nécessaires mais d'autres (comme celui pour choisir le temps qu'une page peut mettre à charger) n'existent pas avec Playwright ce qui va par la suite être gênant pour certains robots.

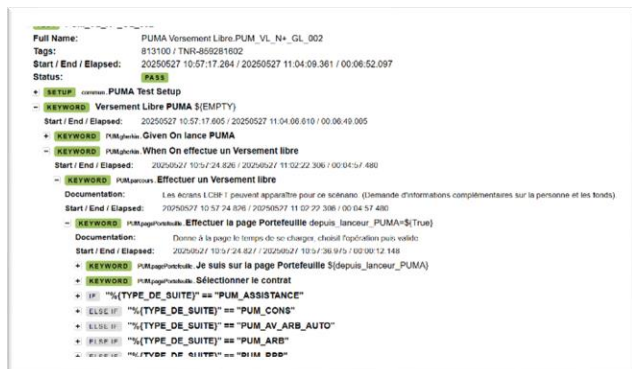
## b. Pour l'application du socle Playwright sur l'application PUMA

L'application PUMA est difficile à automatiser car elle a un comportement erratique.

Par exemple, lorsqu'on sélectionne un élément sur une liste, l'élément est bien sélectionné mais la page est parfois, en même temps, rafraîchie. Le robot n'a pas la capacité de détecter le rafraîchissement et poursuit son action alors que ce n'est pas possible. Pour résoudre ce problème, j'ai dû ajouter des attentes dans le code du robot pour que les actions souhaitées puissent réellement être effectuées. Cela nuit donc à l'efficacité du robot. Le robot existant déjà, ces types de comportements étaient globalement déjà gérés. J'ai tout de même eu certains moments où le robot allait trop vite pour le site web qui n'arrivait pas à suivre. Playwright étant plus rapide que Selenium, ces problèmes n'étaient jamais arrivés avant. J'ai donc dû allonger les temporisations.

De même, certaines pages peuvent mettre plus d'une minute à charger ce qui met en échec le robot.

J'ai eu également un problème à un moment dans mon code qui amenait un bouton à devenir non cliquable et mettait mon code en échec. Pour trouver l'origine de ce problème je me suis un peu cassé les dents sur l'analyse des logs d'exécution du robot car l'anomalie se cachait sous une ligne qui était considérée comme OK.



```
Full Name: PUMA Verserment Libre.PUM_VL_N+.GL_002
Tags: 813100 / TNR-858281802
Start / End / Elapsed: 20250527 10:57:17.284 / 20250527 11:04:09.361 / 00:06:52.097
Status: PASS
+ [STEP] comment: PUMA Test Setup
- [KEYWORD] Verserment Libre PUMA $(EMPTY)
Start / End / Elapsed: 20250527 10:57:17.605 / 20250527 11:04:06.610 / 00:06:49.005
+ [KEYWORD] PUMajobite: Given On lance PUMA
- [KEYWORD] PUMajobite: When On effectue un Verserment libre
Start / End / Elapsed: 20250527 10:57:24.826 / 20250527 11:02:22.306 / 00:04:57.480
- [KEYWORD] PUMajobite: Effectuer un Verserment libre
Documentation: Les actions LCB 1 peuvent apparaître pour ce scénario. (Demande d'informations complémentaires sur la personne et les fonds)
Start / End / Elapsed: 20250527 10:57:24.826 / 20250527 11:02:22.306 / 00:04:57.480
- [KEYWORD] PUMajobite: Effectuer la page Portefeuille depuis_lanceur_PUMA=${True}
Documentation: Demande à la page le temps de se charger, check l'application pour vérifié
Start / End / Elapsed: 20250527 10:57:24.827 / 20250527 10:57:36.975 / 00:00:12.148
+ [KEYWORD] PUMajobite: Je suis sur la page Portefeuille $(depuis_lanceur_PUMA)
+ [KEYWORD] PUMajobite: Sélectionner le contrat
+ [IF] "${(TYPE_DE_SUITE)}" == "PUM_ASSISTANCE"
+ [ELSEIF] "${(TYPE_DE_SUITE)}" == "PUM_CONS"
+ [ELSEIF] "${(TYPE_DE_SUITE)}" == "PUM_AV_ARB_AUTO"
+ [ELSEIF] "${(TYPE_DE_SUITE)}" == "PUM_ARB"
+ [ELSEIF] "${(TYPE_DE_SUITE)}" == "PUM_XXX"
+ [ELSEIF] "${(TYPE_DE_SUITE)}" == "PUM_YYY"
```

9 exemple de log OK de RobotFramework

Cette application m'a donc forcé à trouver des moyens de contournement pour effectuer certaines actions.

Une autre difficulté visible à travers l'application PUMA est que le robot doit rester fonctionnel pour être utilisé à la fois avec la librairie Selenium et la librairie Playwright. En fonction de la librairie choisie, le code exécuté n'est pas totalement le même mais doit donner le même résultat. Le comportement du robot n'a pas changé mais quelques adaptations ont été nécessaires pour le traitement de certains inputs. Certaines actions sont possibles avec Selenium mais pas avec Playwright et inversement pour d'autres actions. Toutefois le robot doit fonctionner exactement de la même façon qu'on le lance avec l'une ou l'autre des librairies.

## c. Pour l'application du socle Playwright sur l'application @DELE

En commençant à travailler sur @DELE, je n'arrivais pas à faire afficher la pop-up d'authentification. Or, la demande était justement de faire disparaître la pop-up avec Playwright. Comme je faisais tourner le robot en local, je pensais que le problème venait de la configuration de mon poste qui passait toute la partie de l'authentification. Il a fallu que je fasse le test en condition réelle pour me rendre compte que la pop-up n'apparaissait pas du tout avec Playwright et que donc le problème s'était réglé de lui-même.

## d. Pour la création d'un nouveau robot sur l'application Mercure

Pour créer le robot sur l'application Mercure, j'ai rencontré plusieurs difficultés.

Tout d'abord, les entrants fournis par le projet sont des captures d'écran de l'application enregistrées dans un fichier PDF. Ces captures simulent des parcours en fonction des données d'entrée. En

développant le robot, je me suis aperçu que les affichages changent en fonction des données d'entrée. J'ai donc dû faire des adaptations dans le robot en fonction de ce que je découvrais et il y a un risque que des cas non visibles dans les PDF ne fonctionnent pas puisque je ne les connais pas.

Ensuite, le robot a dû être développé sur les environnements de développement et de recette alors qu'ils sont normalement développés sur l'environnement de préproduction. Cela s'explique car le projet était important et il fallait anticiper la création du robot. Par contre, ces environnements sont instables et sont dépendants d'API de La Banque Postale. Ces API sont régulièrement hors service et j'ai donc été souvent bloqué pour tester le robot. Comme ces API sont gérées en dehors de la CNP, leur remise en état prenait beaucoup de temps. Il m'est arrivé plusieurs fois d'être bloqué toute la journée. A cause de ces problèmes d'environnement, je n'ai pu tester mon robot que le jeudi 19/06 soit le jour prévu pour la démo et deux jours avant la fin de mon stage ce qui laisse peu de temps pour régler les problèmes découverts.

De plus, ces problèmes d'environnement et le fait que je ne connaisse pas l'application font que lorsque je rencontrais une erreur, je ne savais pas forcément si elle venait de l'application, du robot ou de l'environnement. Lorsque j'ai commencé à travailler sur le robot et que j'avais une erreur, je demandais à un membre du projet pour comprendre ce que signifiait l'erreur. Je me suis vite rendu compte que l'erreur venait tout le temps de l'environnement.

Le robot nécessite aussi d'automatiser le téléchargement de PDF. Jusqu'à présent, je n'avais pas rencontré ce cas de figure donc le socle n'était pas compatible avec le téléchargement de fichier. En effet, pour un PDF, Playwright ne télécharge pas le fichier automatiquement mais ouvre le viewer de PDF Chrome pour l'afficher directement dans le navigateur. Ce n'était pas le comportement attendu. J'ai donc dû adapter le socle Playwright en lançant le navigateur en mode Headless. De cette manière, le viewer de PDF ne s'active pas et le téléchargement peut se faire. Cependant, cela pose une difficulté supplémentaire car on ne peut plus voir ce que fait le robot en temps réel et cela rend le debug plus compliqué. En outre, le fichier téléchargé prend un nom totalement aléatoire et sans extension. Pour ce robot, le nom du fichier téléchargé n'est pas important mais ça pourrait poser soucis sur d'autres robots qui nécessiteraient de récupérer le bon nom de fichier. Par contre, le problème d'extension devait être corrigé. J'ai donc complété le robot pour qu'il rajoute l'extension après le téléchargement. Le fait que le fichier n'ait pas d'extension complique grandement le code car il faut chercher tous les fichiers du répertoire et vérifier qu'ils ne contiennent pas d'extension. Cela rend le code potentiellement moins performant.

Enfin, les problèmes d'environnement m'ont empêché de tester le robot de bout en bout pendant la quasi-totalité du temps où j'étais sur le projet. Pour valider le téléchargement du PDF qui est l'avant-dernière étape du robot, j'ai dû créer un mini site WEB qui simule le téléchargement d'un PDF.

Pour terminer, lors de la démo du robot, un bug a été détecté car je n'avais pas pu tester tous les cas à cause des problèmes d'environnement, ce qui fait que j'ai dû refaire des modifications dans le robot par la suite. Heureusement, ces modifications n'étaient pas très importantes et je n'ai pas eu de mal à faire ces corrections.

#### e. Ajout de la librairie Appium

L'implémentation de la librairie Appium a causé beaucoup de difficultés. A la CNP, nous ne disposons pas de téléphones pour tester directement nos robots. Il faut donc faire appel à un service externe qui met à disposition de nombreux appareils sur lesquels nous pouvons tester. Le fait de ne pas le faire en local a posé plusieurs problèmes. En effet, pour tester sur des téléphones à distance, il faut passer par

un proxy pour tenir compte des restrictions réseau de la CNP. Le souci est que ces contraintes ne sont documentées nulle part car c'est un cas particulier lié à l'entreprise. Régler ce problème a pris plusieurs journées. De plus, pour lancer l'application sur le téléphone à distance, il faut renseigner des informations précises pour déterminer quel téléphone doit exécuter les tests.

Un autre souci vient du fait que la librairie Appium est moins bien documentée que Selenium ou Playwright. Par moment, il faut en quelque sorte deviner ce qu'il faut faire notamment sur les mots clés plus techniques comme ceux permettant d'ouvrir l'application. Les paramètres nécessaires ne sont pas tous donnés sur la documentation officielle et il faut aller les trouver sur différents forums. De même, cette librairie n'est pas souvent mise à jour et est beaucoup moins fournie que les deux autres présentées avant. Il faut donc trouver des solutions pour simuler les mots clés qui n'existent pas en faisant une série plus complexe d'actions.

#### f. Pour le travail avec l'équipe

Les équipes de la CNP peuvent être en télétravail 3 jours par semaine. Etant sur site à temps plein, j'ai pu rencontrer des difficultés lorsque mon tuteur n'était pas présent. Dans ces cas, nous utilisons Teams pour échanger. C'était plus compliqué pour me débloquer de cette façon mais on a toujours réussi à trouver des solutions à mes problèmes et il restait quand même très joignable.

## 5. Bilan

Je suis très satisfait de mon stage car je vais pouvoir réutiliser ce que j'ai appris l'année prochaine et tout au long de ma carrière. Par exemple, le stage m'a permis d'apprendre à automatiser des tests WEB sur PC et mobile et à utiliser de nouveaux outils que je ne connaissais pas avant. J'ai aussi appris à me servir de la documentation en ligne pour pouvoir travailler sur ces outils en autonomie. Mon stage m'a permis de créer entièrement un robot de test et de travailler pour la première fois sur du code existant : pendant la première année de BTS, je n'ai travaillé que sur mon code alors que pendant le stage, j'ai dû comprendre comment le code fait par une autre personne fonctionnait.

Le stage me permet aussi de voir que j'ai encore des réflexes à prendre : par exemple, il m'est arrivé de travailler plusieurs heures sur de l'écriture de code avant de me rendre compte qu'il avait déjà été écrit à un autre endroit. J'aurais dû regarder un peu plus ce qui existait déjà avant de me lancer. Un autre point à améliorer serait de lire la documentation plus attentivement avant de demander de l'aide. Il m'est arrivé quelque fois de demander à mon maître de stage une information que j'aurais pu trouver seul en lisant la documentation deux minutes de plus.

A l'issue de mon stage, mon maître de stage va reprendre mes travaux en s'appuyant sur le socle Playwright et la documentation associée. Par exemple, si les testeurs se rendent compte qu'il manque un cas de test qui n'a pas été fait pour le robot, mon maître de stage devra l'ajouter au code. Le socle Appium reste également encore à finir mais la base est là pour commencer à faire des tests sur mobile. Par contre, le socle Playwright est normalement suffisant pour permettre la bascule des autres robots du périmètre de l'équipe grâce aux tests unitaires. Le socle serait à adapter uniquement si un futur robot comprenait un nouveau cas d'usage qui n'existe pas encore.