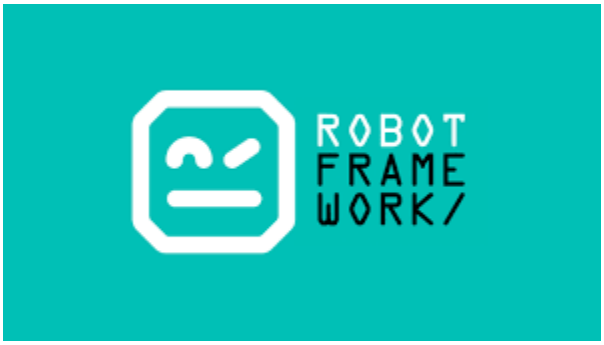

ROBOT
FRAME
WORK/


ZEPHYR

« On a nos tests manuels qui sont suivis dans notre outil de gestion des tests, qui est Zephyr. Et puis, on a des tests automatisés qui tournent tous les jours (ou en tout cas à des fréquences différentes), avec Robot Framework… Les résultats de ces derniers sont visibles dans l'outil de CI/CD, et ce n'est pas pratique… Ce serait bien d'avoir tous nos résultats de test au même endroit, pour prendre des décisions plus facilement ! ».

Cette phrase, on l'entend à peu près tous les mois, voire toutes les semaines. Il suffirait d'y changer « Zephyr » en « XRay » ou en « Squash TM », et « Robot Framework » en « Selenium », « Playwright », « Katalon Studio » ou « Cypress » pour que n'importe quel testeur sur le marché l'ait déjà prononcée !

Dans cet article, nous allons voir comment centraliser dans Zephyr (un gestionnaire de test édité par Smartbear, et qui se plugue parfaitement au JIRA d'Atlassian) les résultats des tests automatisés de Robot Framework dans Zephyr (où on trouve déjà les résultats des tests manuels…).

Pour la petite histoire, nous sommes aussi parvenus à récupérer les résultats de tests Postman dans Zephyr. Nous pouvons en fait y récupérer les résultats de beaucoup d'outils, tant qu'on a (entre autres) un xml au format xunit comme résultat, ou encore un format Cucumber…

1. L'API Zephyr Scale (Cloud)

Il n'y aura évidemment pas de magie pour récupérer dans Zephyr les résultats des tests automatisés Robot Framework. Cela va passer par l'API Zephyr Scale, dont la documentation est ici :

https://support.smartbear.com/zephyr-scale-cloud/api-docs/

Rapidement, on peut voir qu'il est possible, via cette API, de manipuler les différents types de tickets que l'on peut trouver nativement dans Zephyr :

Test Cases,

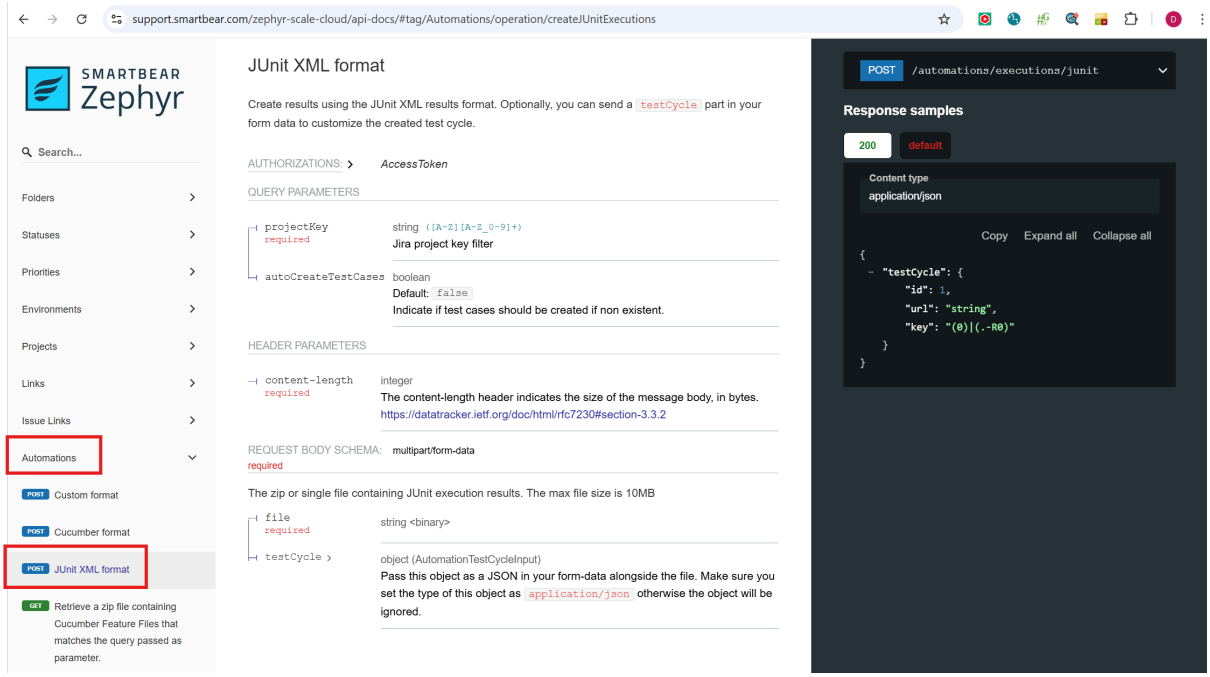
Test Cycles (c'est un ensemble d'exécutions d'une suite de tests, ou encore de N cas de test qui se sont exécutés ensemble).

Test Plans (contient l'ensemble des Tests Cycles, qu'ils soient manuels ou automatisés),

On peut aussi manipuler des données de Test Executions, de statuts, de priorités, d'environnements, et également d'automatisations.

On peut vite voir dans cette documentation qu'on peut utiliser des méthodes GET, POST, PUT sur les principaux artefacts : bref, on va pouvoir faire des choses intéressantes, surtout si on est à l'aise avec l'utilisation des APIs, dans un outil comme Postman, Bruno ou équivalent…

Ci-dessous la copie d'écran de la documentation de l'API Zephyr, sur un POST d'un JUnit XML Format, au niveau des Automatisations :



Ce endpoint permet ainsi de faire un POST d'un fichier résultat (xml) en précisant l'ID du projet JIRA concerné (projectKey) et également en précisant si les cas de test qui ne sont pas trouvés dans Zephyr devront être automatiquement créés ou pas (autoCreateTestCases qui peut valoir True ou False).

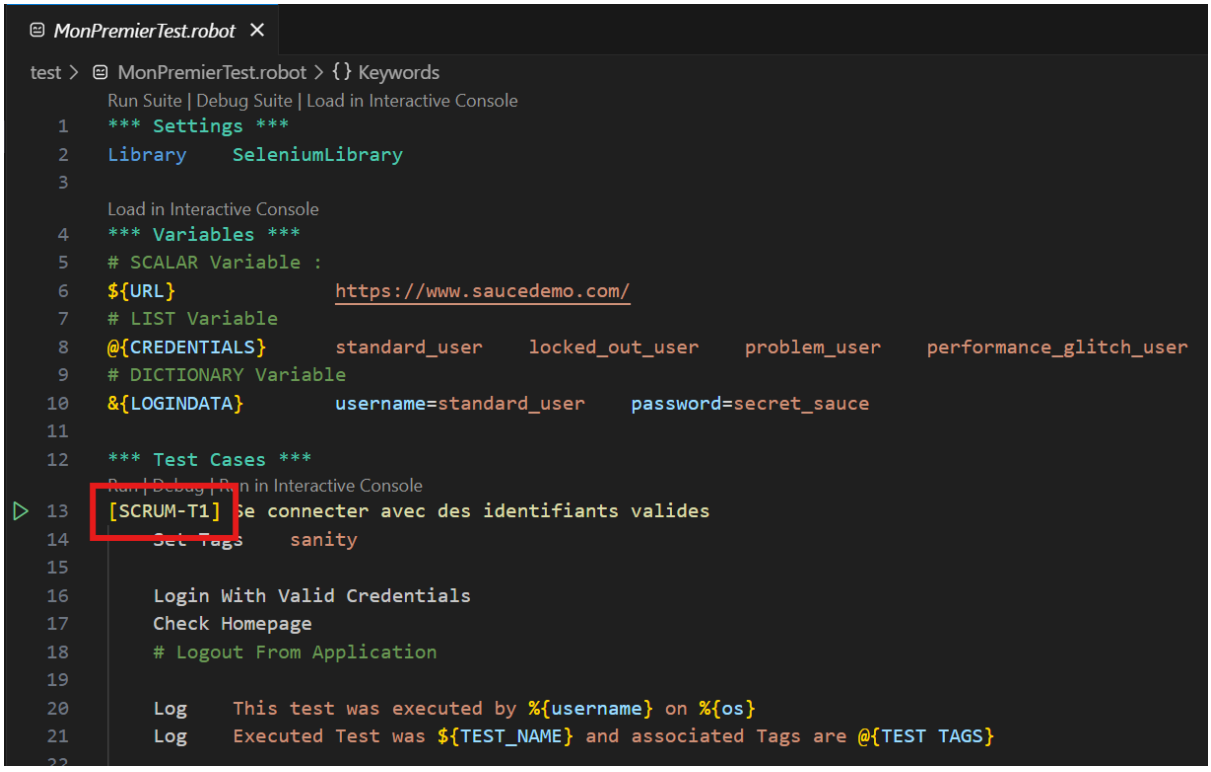
2. Pré-requis entre Zephyr & Robot Framework

La future liaison entre un résultat de test Robot Framework et le cas de test présent dans Zephyr nécessite UN ET UN SEUL pré-requis : l'Identifiant du cas de test Zephyr doit être présent dans le TITRE du cas de test Robot Framework.

Exemple avec ce cas de test qui porte l'ID « SCRUM-T1 ».

Dans Zephyr :

Dans le script de test automatisé Robot Framework (implémentation en Python sous VS Code) :



Ce pré-requis est le seul à respecter pour que la récupération du résultat de test automatisé fonctionne : qu'importe si les titres soient différents, ou si votre cas de test Zephyr est écrit en mode test manuel. Il va falloir de la rigueur pour facilement distinguer dans votre référentiel de tests les tests manuels et les tests automatisés.

3. Étape 1 : Exécuter la suite de test Robot Framework

Cette étape est essentielle pour obtenir en sortie des résultats de test. L'exécution peut être faite sur un poste en local, comme depuis une CI/CD (Jenkins, GitLab CI, GitHub Actions…).

En local, depuis un terminal en lignes de commandes :

robot .\test\MaSuiteDeTest.robot

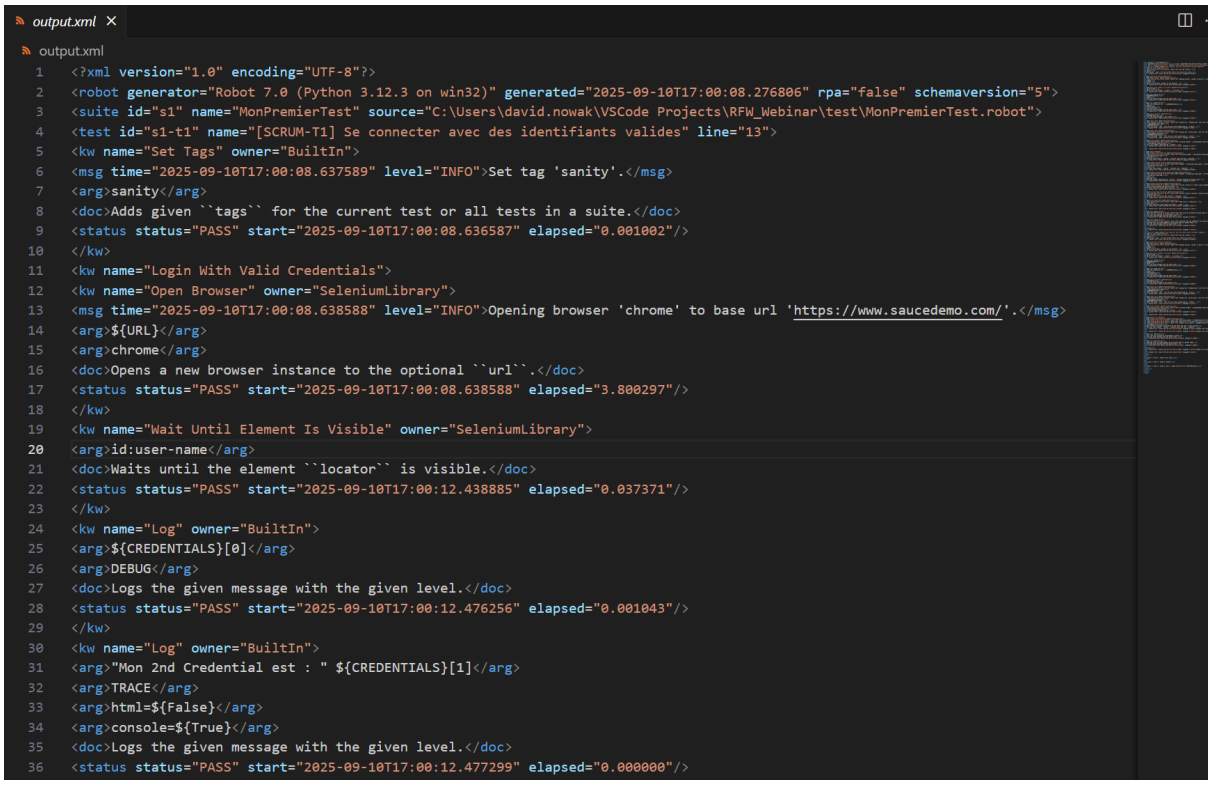
Avec Robot Framework, nativement, 3 fichiers sont créés suite à l'exécution d'une suite de tests :

log.html ➔ Détail des étapes OK ou KO pour l'ensemble des tests de la suite de test.

output.xml ➔ Le fichier qui nous intéresse ici ! On y retrouve toutes les infos du fichier log.html.

report.html ➔ Rapport global d'exécution avec principales métriques

Voici un extrait du fichier output.xml obtenu au moment de la rédaction de cet article :



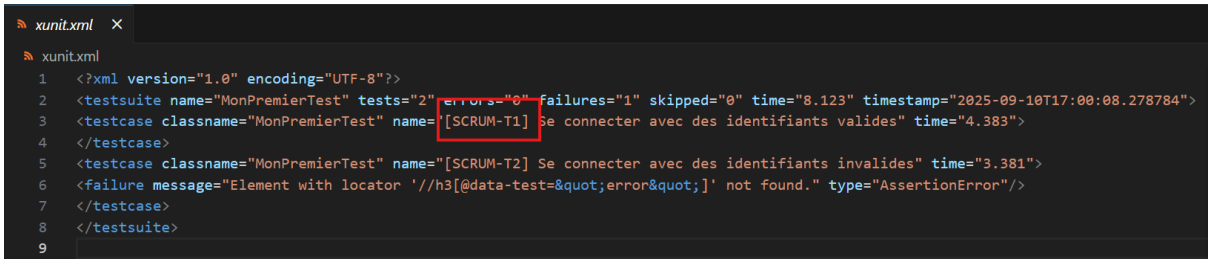
4. Étape 2 : Transformer output.xml en xunit.xml

Le malheur avec Zephyr (à date), c'est que l'API n'accepte pas le format natif de output.xml (ce n'est pas le cas d'autres APIs d'autres outils de gestion de test). Pour Zephyr, il va falloir transformer ce fichier output.xml pour en obtenir un autre, ici xunit.xml : ce nouveau fichier respecte le format JUnit, compatible avec l'API Zephyr.

En ligne de commandes :

rebot --xunit xunit.xml output.xml

Cette commande prend output.xml (format Robot Framework) en entrée, et créée xunit.xml (format JUnit) en sortie. Le fichier, bien plus épuré, ressemble à ceci :



On retrouve bien SCRUM-T1, l'identifiant du test Zephyr, dans ce fichier.

5. Étape 3 : Envoyer les résultats (xunit.xml) dans Zephyr par API

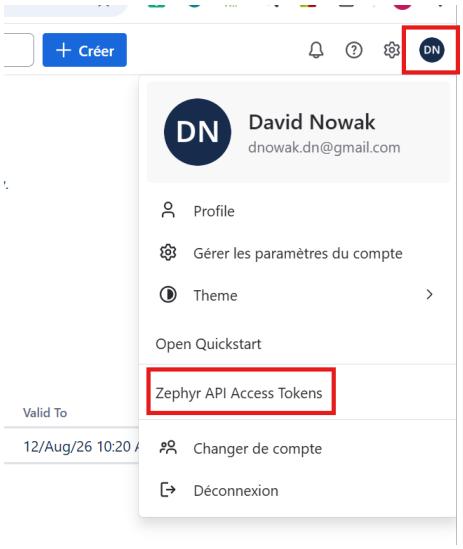
Cette étape est ici réalisée avec Postman, cela pourrait se faire en fin d'exécution en CI/CD, dans une étape dédiée à faire les mêmes appels.

On fait un appel POST sur l'URL suivante :

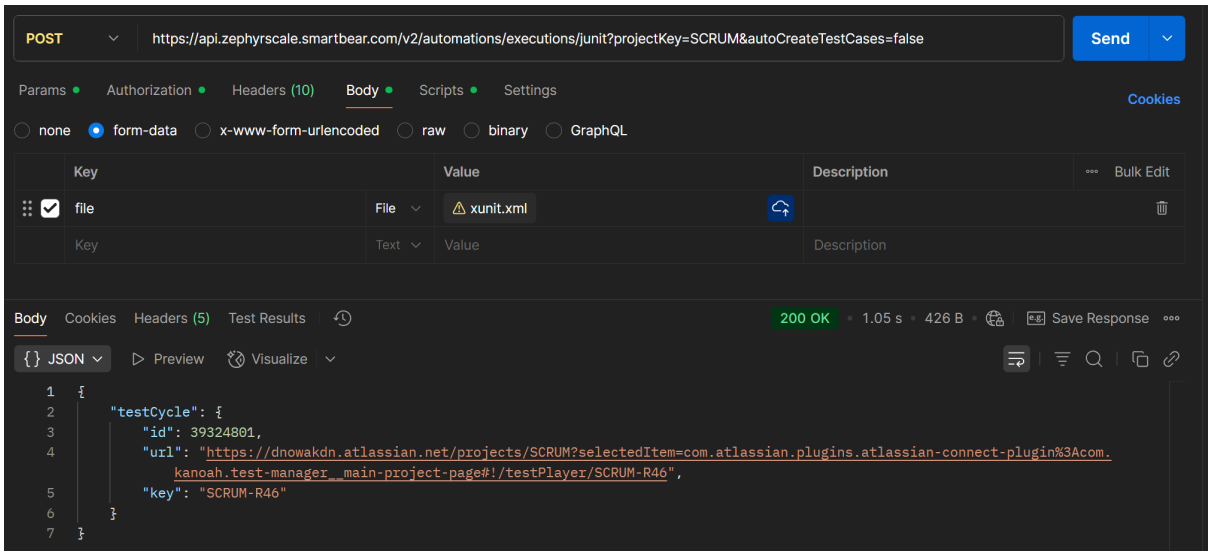
https://api.zephyrscale.smartbear.com/v2/automations/executions/junit?projectKey=SCRUM&autoCreateTestCases=false

Paramètres : « SCRUM » comme projectKey, et false sur autoCreateTestCases

Authorization : à noter que l'autorisation est de type « Bearer Token » : le token est fourni par votre administrateur Zephyr dans votre organisation. Sinon, dans Jira, cliquer sur l'utilisateur (en haut à droite) puis sur « Zephyr API Access Tokens » :

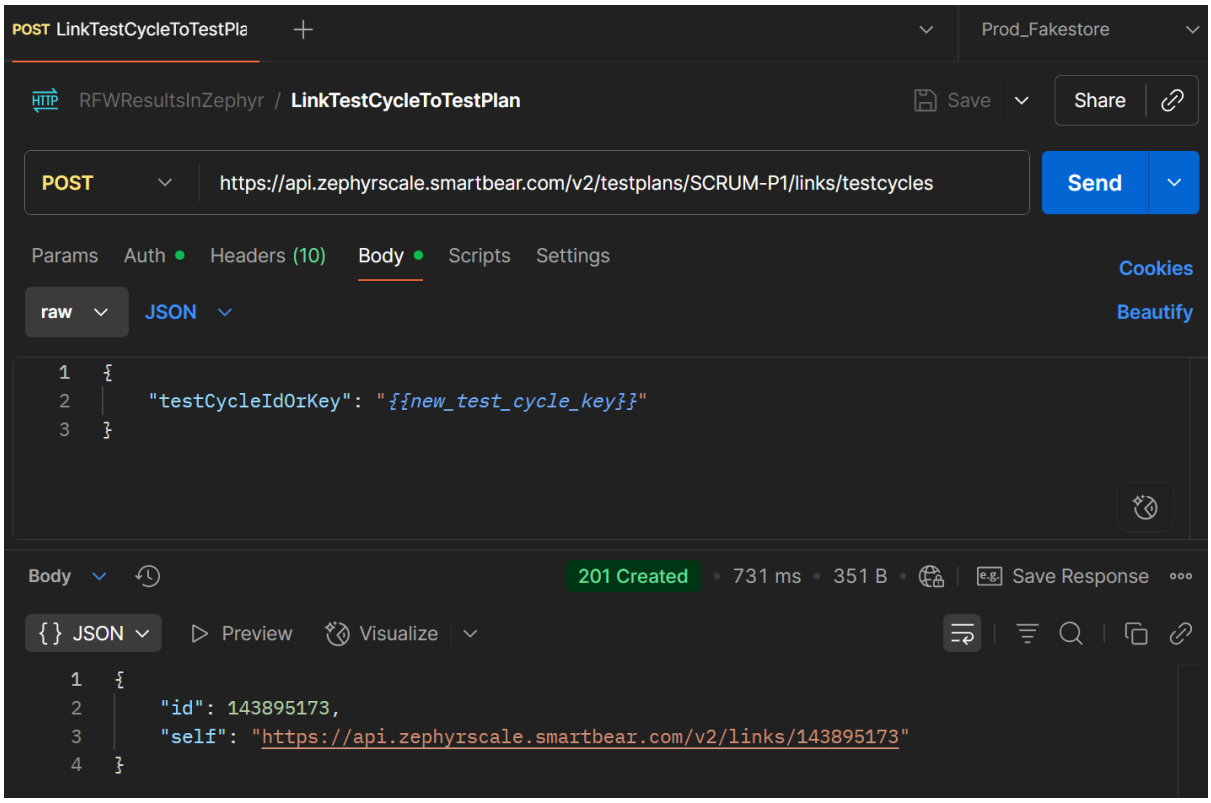


Body : Sélectionner form-data, et renseigner la clé « file », de type « File ». Uploader le fichier xunit.xml, et exécutez la requête :



La récupération du résultat du test case « SCRUM-T1 » dans Zephyr est OK à ce stade.

Notez qu'un Test Cycle (ici avec l'ID « SCRUM-R46 » est également créé : il contient les résultats d'exécutions de l'ensemble des cas de test qui ont été exécutés avec Robot Framework. Il est ainsi possible de récupérer cet identifiant, et d'utiliser une AUTRE API Zephyr pour rattacher automatiquement ce TEST CYCLE nouvellement créé à un TEST PLAN. Ici, nous sommes passés par un SET et un GET d'une variable de collection avec Postman :

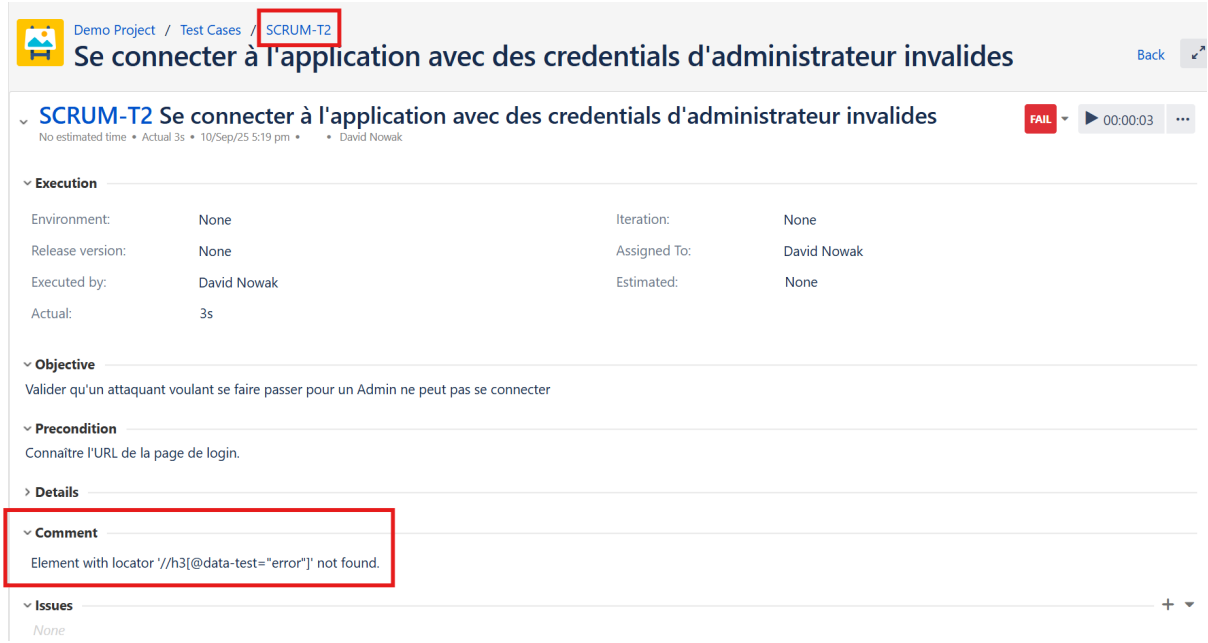


Enfin, vérifions dans Zephyr, au niveau de l'historique des exécutions du cas de tests « SCRUM-T1 » :

La nouvelle exécution « Passed » est bien présente, en étant considérée (tout à droite) comme un résultat de test automatisé. SCRUM-R46 est le TEST CYCLE qui vient d'être créé. SCRUM-E102 est le résultat de ce test UNIQUE.

Le cas de test « SCRUM-T2 » a aussi été impacté, puisque son exécution était FAILED, comme le montre la copie d'écran suivante avec le même TEST CYCLE SCRUM-R46 :

En cliquant sur le détail de cette exécution, on peut voir l'erreur qui a été rencontrée par le script de test automatisé, au niveau du COMMENT :



6. Conclusion

La technique est ici au service de l'organisation. Pour récupérer les résultats d'une suite de tests automatisés, il y a plusieurs possibilités.

D'abord, un QA technico-fonctionnel peut décider de ne pas envoyer ses résultats dans Zephyr, par exemple lorsqu'il est en train de développer de nouveaux scripts qui ne sont pas encore totalement fonctionnels.

Ensuite, un QA technico-fonctionnel peut décider d'envoyer ses résultats dans Zephyr, même s'il les lance localement. Pour cela, il lui suffira d'utiliser un outil comme Postman, Bruno ou autre, comme ce qui a été fait tout au long de cet article.

Enfin, il est possible pour une organisation d'industrialiser cette récupération, avec les appels de l'API Zephyr qui se feront dans l'outil d'orchestration de la CI/CD (Jenkins, GitLab CI, GitHub Actions…) : après l'étape d'exécution des tests Robotframework, il suffira d'appeler les APIs via le workflow yml.