RAPPORT MARTIN CLAVER

1. INTRODUCTION

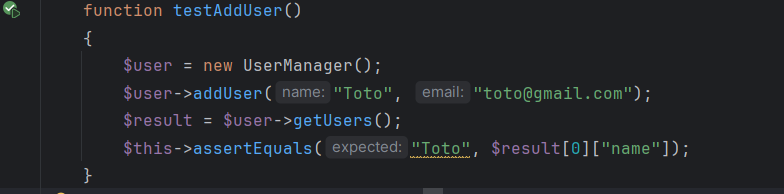
L’objectif est de tester l’application de gestion des users de différentes manières.

Avec la mise en place des tests fonctionnels, des tests end to end, tests de non régressions et tests de performance. On désignera ensuite les problèmes détectés.

On utilisera PHPUnit, Selenium, JMeter.

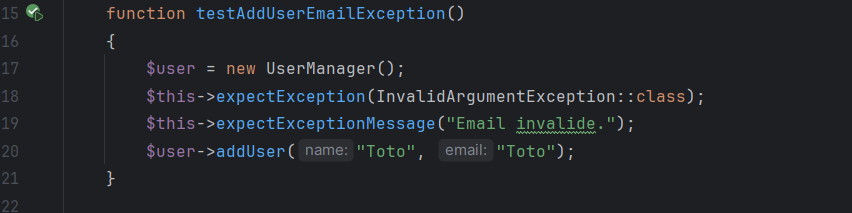
1. RÉSULTATS DES TESTS
2. **Tests fonctionnels**

Test ajout d’un user :



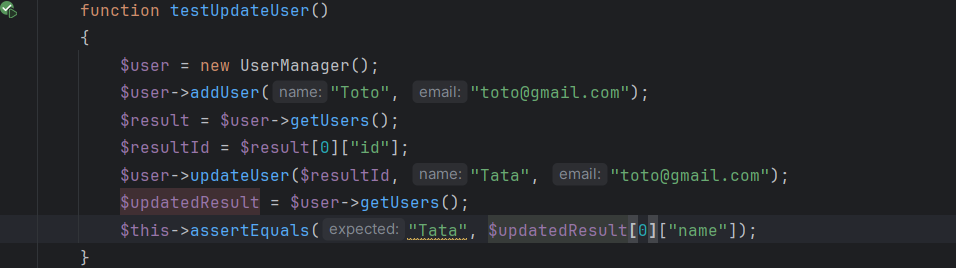
On ajoute un user avec **addUser** et on récupère la base de donnée avec **getUsers**. On vérifie l’égalité avec **assertEquals**.

Test exception ajout de mail :



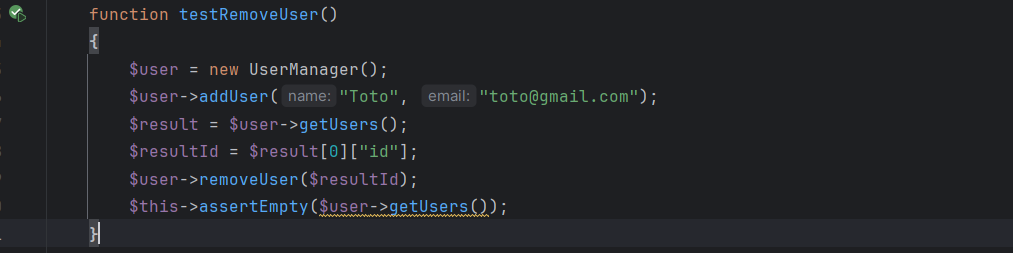
On ajoute un mail qui n’est pas valide avec **addUser** et on vérifie que l’exception et son message sont bien envoyés avec **expectException** et **expectExceptionMessage**.

Test update user :



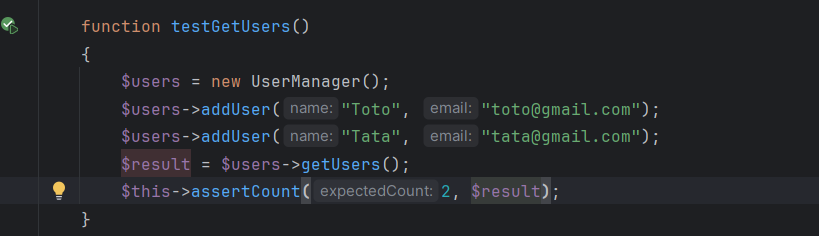
On ajoute un user avec **addUser**, puis on récupère les données et son id avec **getUsers**, on update l’user avec les données récupérées avec **updateUser** et on récupère les nouvelles données avec **getUsers**, on vérifie que les données sont bien modifiées avec **assertEquals**.

Test supprimer un user :



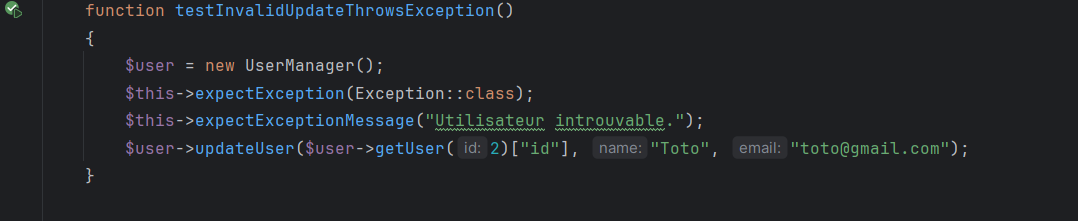
On ajoute un user qu’on supprime en ayant récupéré son id avec **addUser, getUsers** et **removeUser**. Puis on vérifie que les bases de données sont bien vides avec **assertEmpty**.

Test récupérer les users :

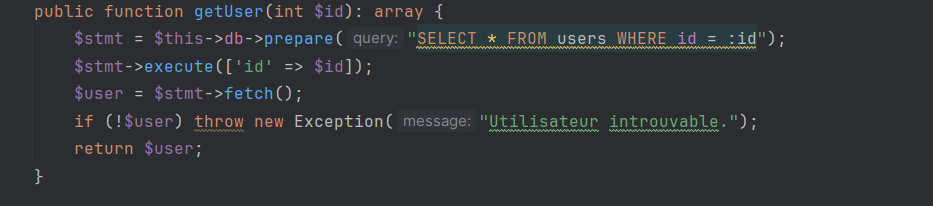


On crée deux users et on les récupère avec **addUser** et **getUsers**, puis on compte le nombre de users dans la bdd avec **assertCount**.

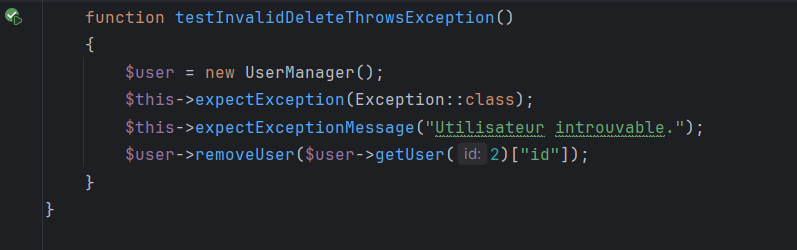
Test update invalide renvoie une Exception :



Quand **getUser** est invalide il renvoie une Exception, **updateUser** dépend de la même condition, l’utilisateur doit exister en base de données.



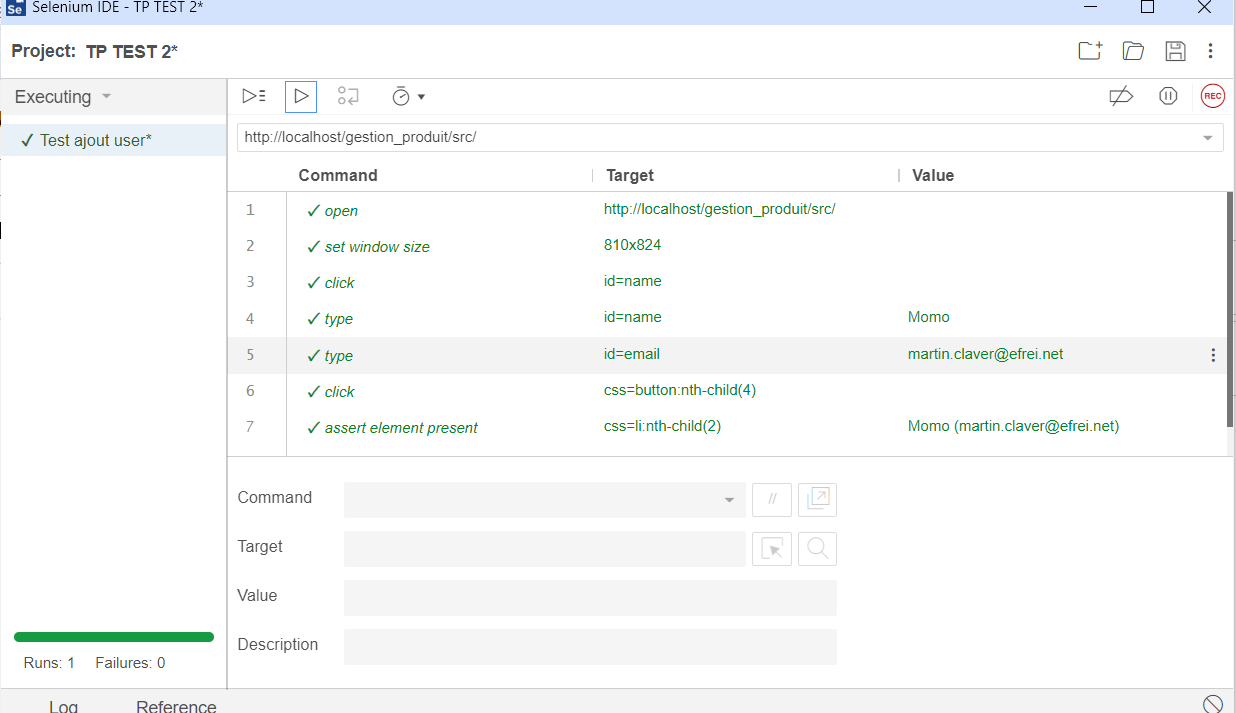
Test invalide à la suppression :



Même logique que pour l’update, si le user n’existe pas, on attend une erreur.

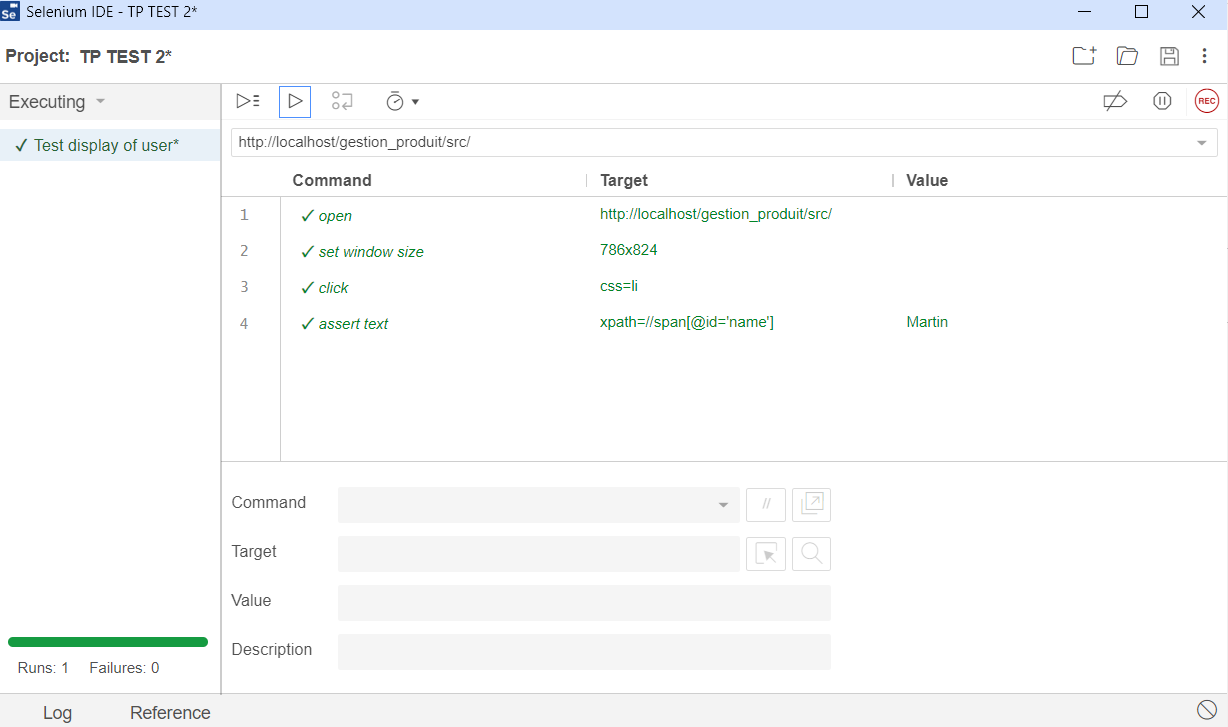
1. **Tests End to End**

Tests Selenium.

Test de l’ajout d’un user :

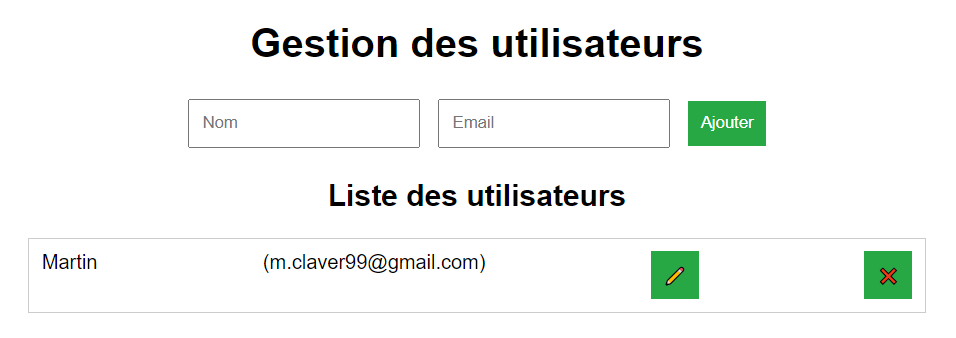
Un élément est bien ajouté après le clic on vérifie avec **assert element present**.

Test de vérification de l’affichage de l’utilisateur ajouté :

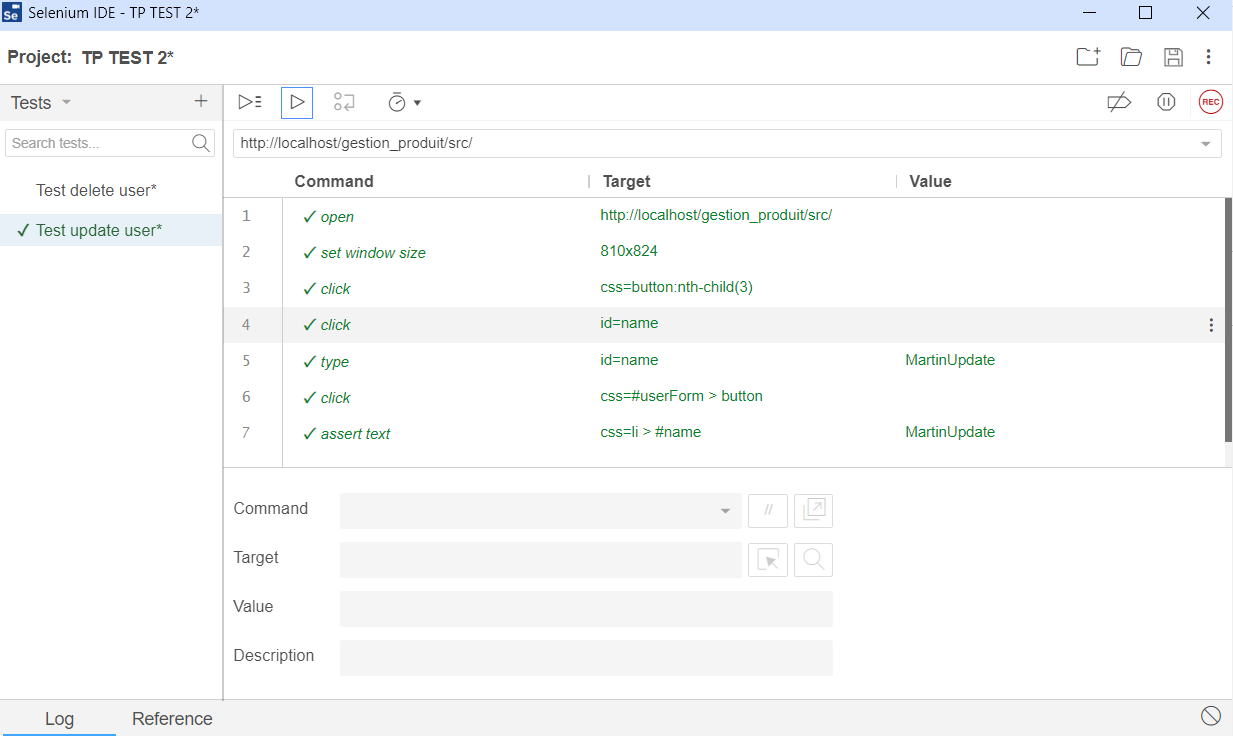


On vérifie que l’élément est correct dans la liste avec **assert text** et la bonne value.

Voici l’interface sur laquelle ont été effectués les tests :

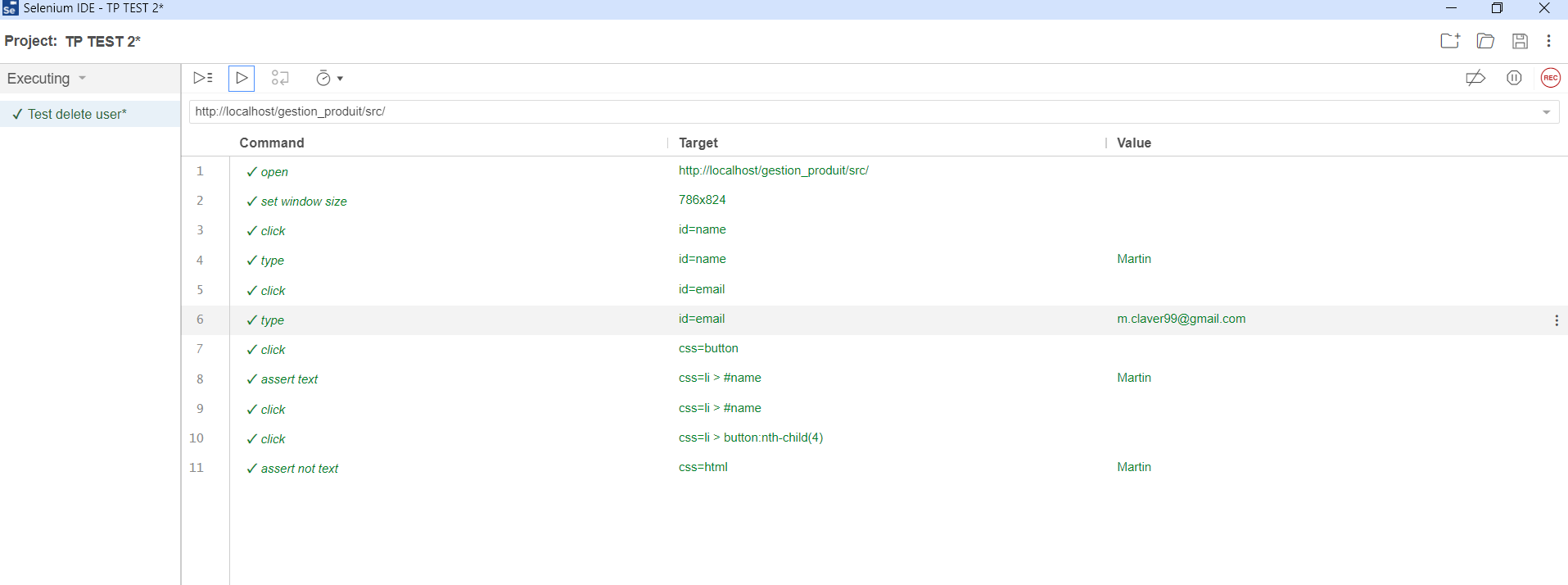


Test mise à jour du name de l’utilisateur :



Un user était déjà ajouté on change son name en MartinUpdate et on vérifie avec **assert text**.

Test de suppression de l’utilisateur :



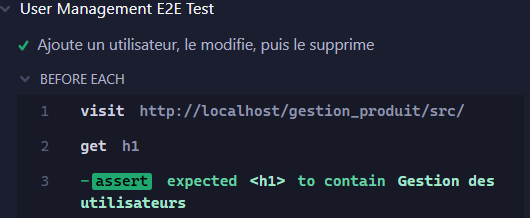
On ajoute un utilisateur, on vérifie qu’il est bien ajouté avec **assert text** et sa **value**, puis on le supprime et on vérifie qu’il est bien supprimé avec **assert not text** et la même **value**.

Tests Cypress.

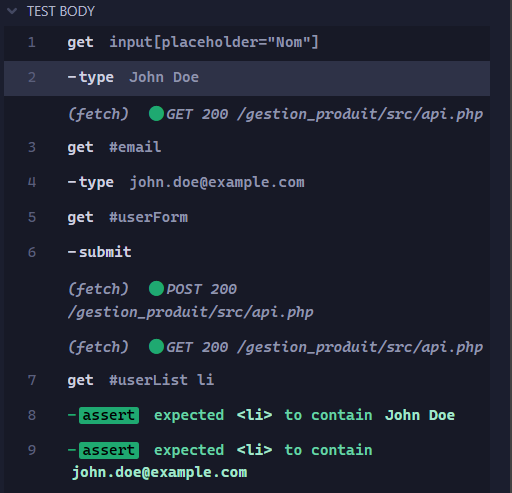
On va réaliser tous les tests ensemble. D’abord l’ajout d’un utilisateur en renseignant les champs voulus. Puis on vérifie que l’utilisateur est ajouté. On modifie le nom. On vérifie qu’il est bien modifié. Puis on le supprime et on vérifie.

Pour vérifier ou ajouter les éléments on va les sélectionner avec un .get et leur id ou leur balise html et son placeholder, ils vont alors être cherché dans la page. Puis, on peut ajouter une **action** .type, .submit, .clear, .click ou une **vérification** .should .contains.

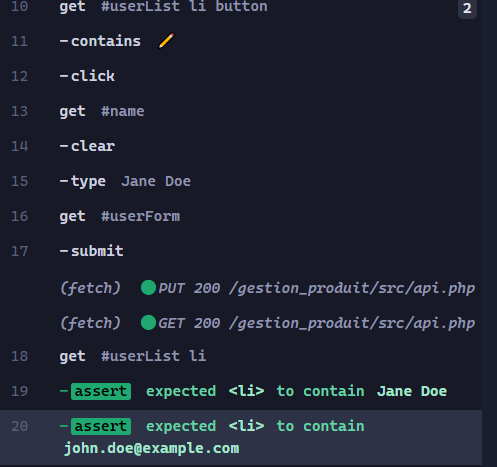




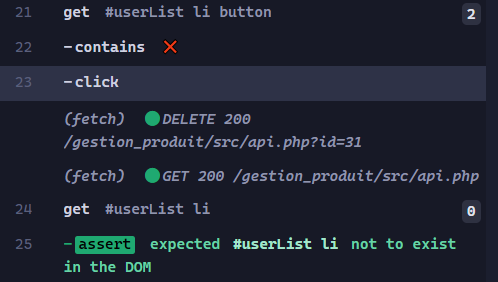
Avant chaque test, on vérifie qu’on est bien sur la bonne page.



On a bien réussi à ajouter un utilisateur et il est bien visible.



Le champ name est bien modifié.

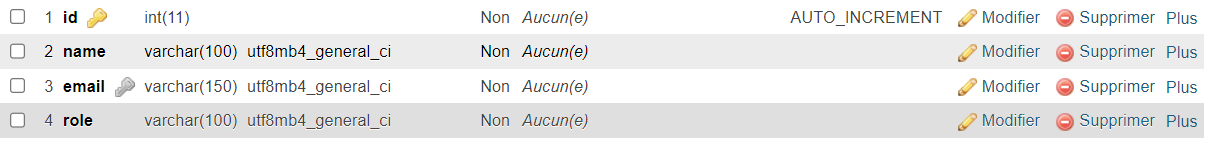


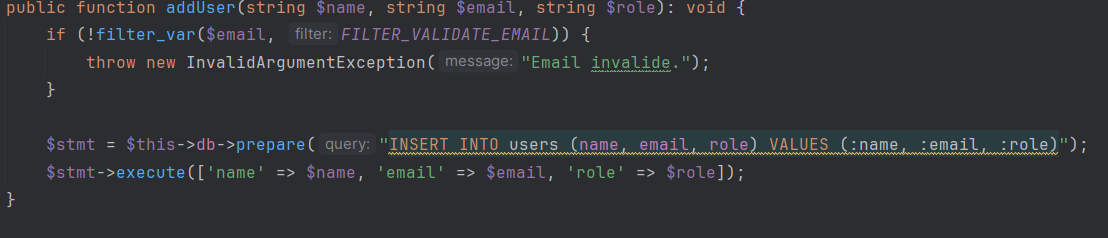
La liste est bien supprimée et elle n’existe plus.

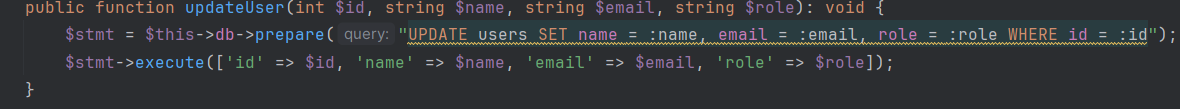
1. **Tests de non régression**

Ajout du champ rôle.

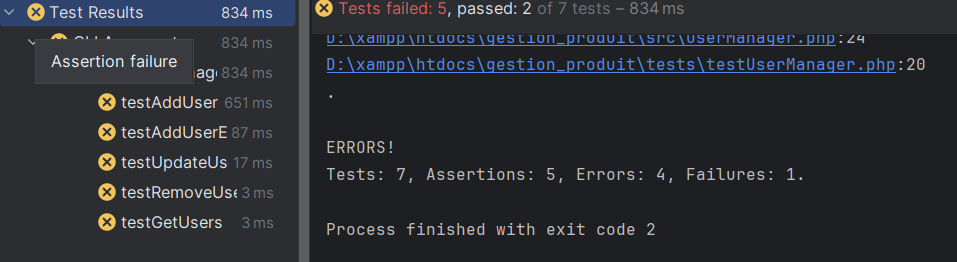
Pour tester la non régression de notre application on ajoute un nouveau champ dans la base de donnée, le champ rôle.







**On relance nos tests précédents.**



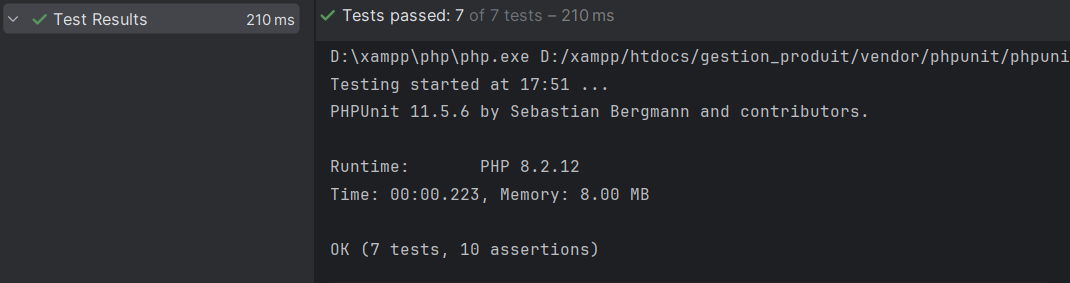
**Les tests ne passent pas, pourquoi ?**

Les tests ne passent pas essentiellement parce que notre modification entraîne des régressions.Des fonctions que nous utilisons dans nos tests sont modifiées, notamment **addUser et updateUser**, il faut donc modifier les tests également pour qu’ils correspondent au code.

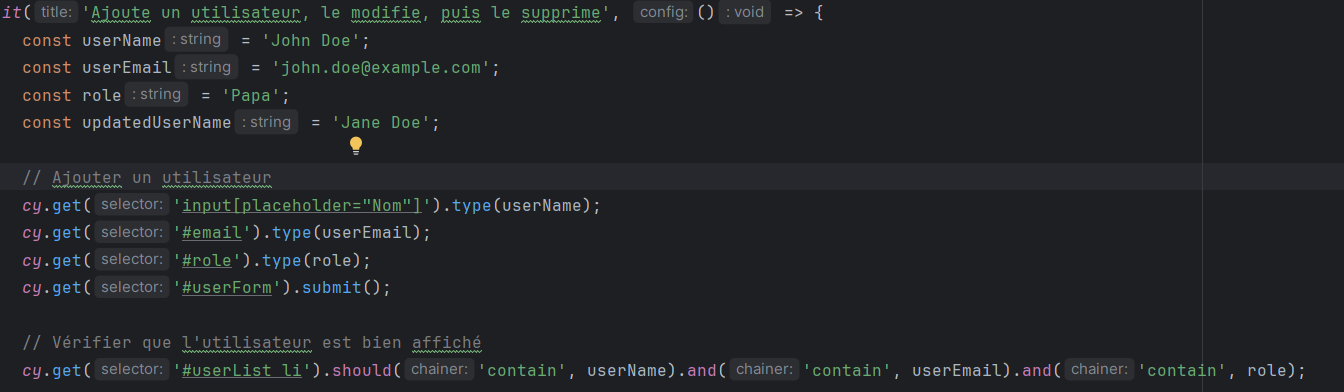
On effectue les changements dans ces fonctions en ajoutant le rôle partout où il est nécessaire.



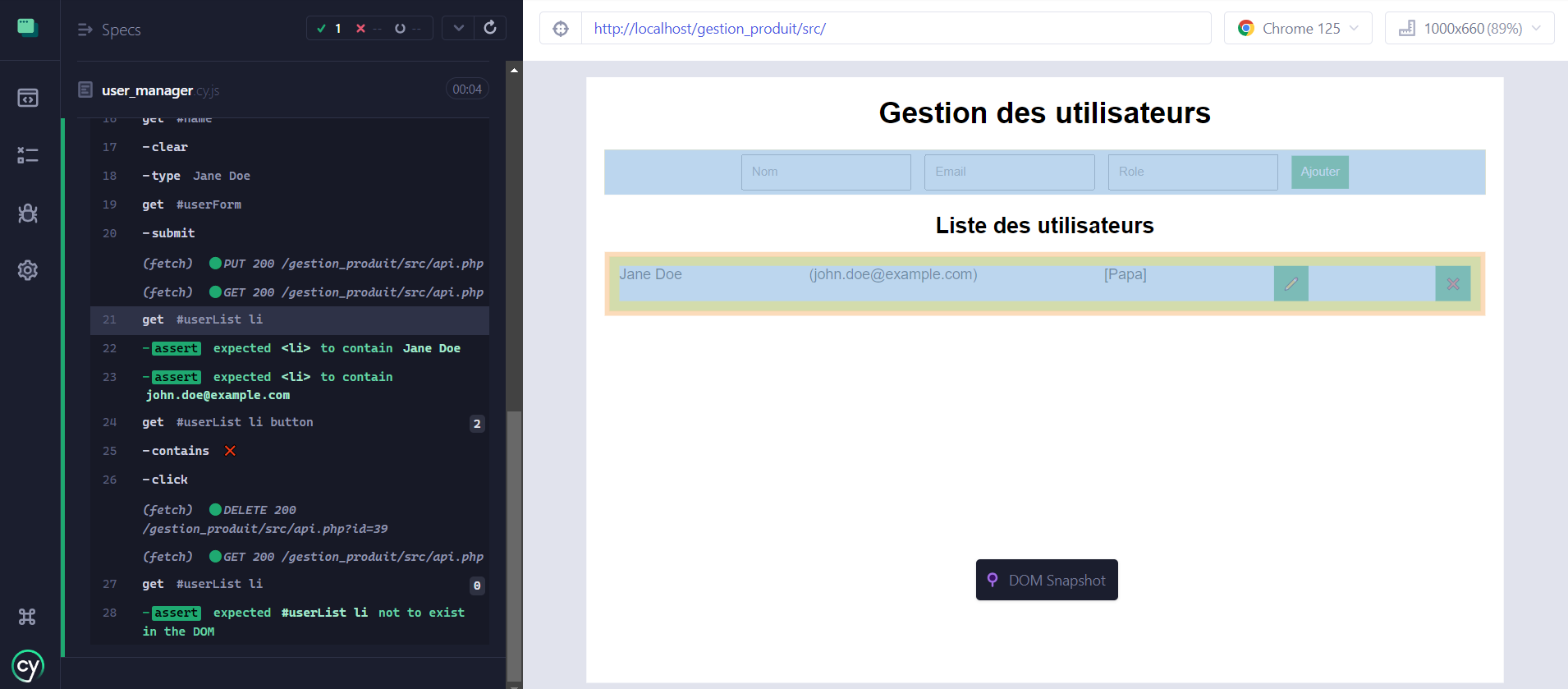
Les tests fonctionnels passent !



On pourrait imaginer l’amélioration d’implémenter un test end to end qui vérifie que le rôle agit de la même manière que le champ name ou email.

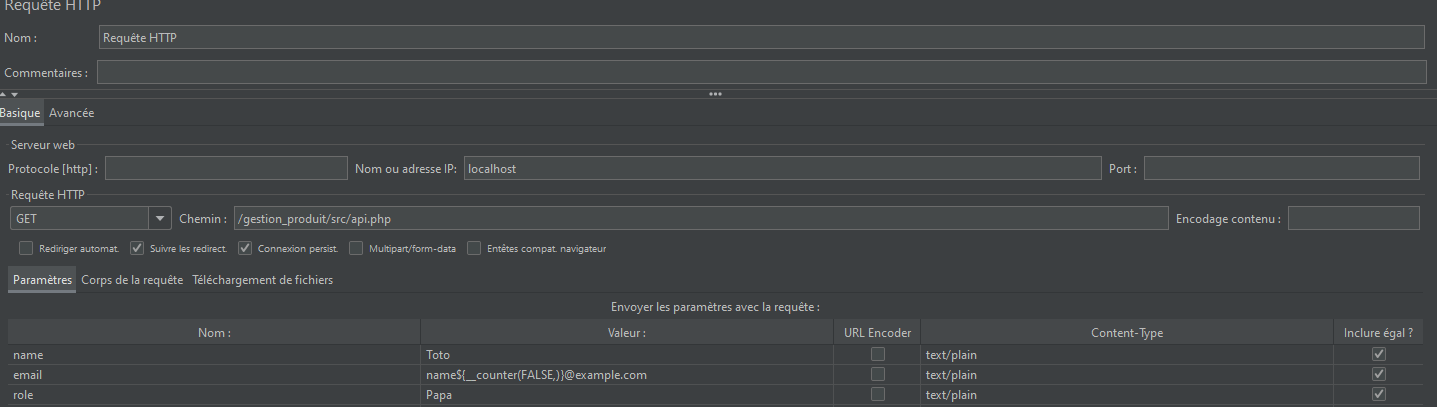
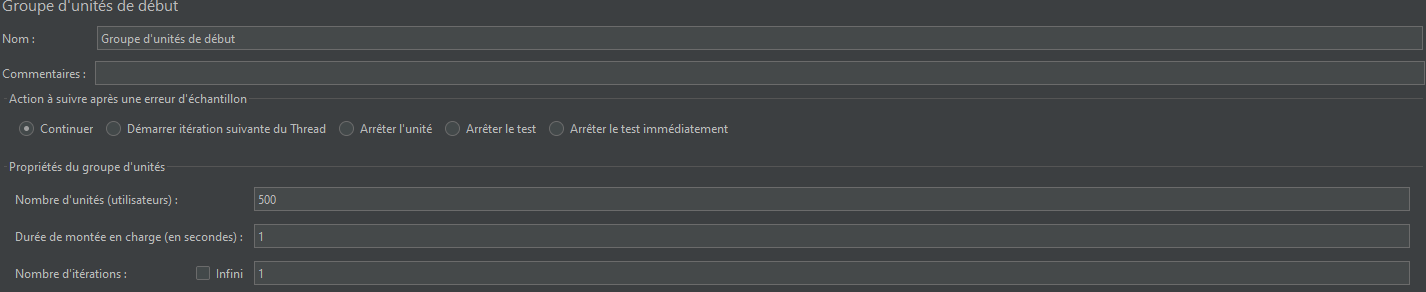


On ajoute le nouveau champ rôle dans le test end to end de cypress. Il passe.



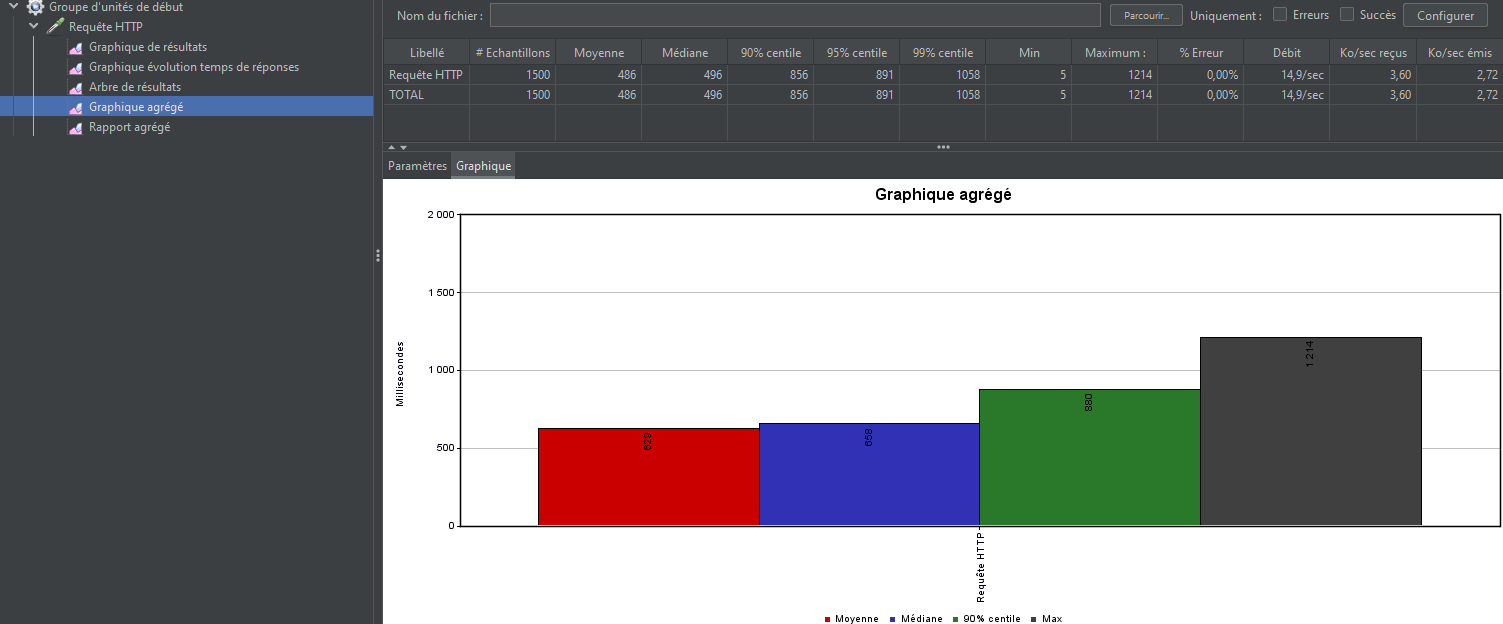
1. **Tests de performance**

**Requête HTTP envoyée :**

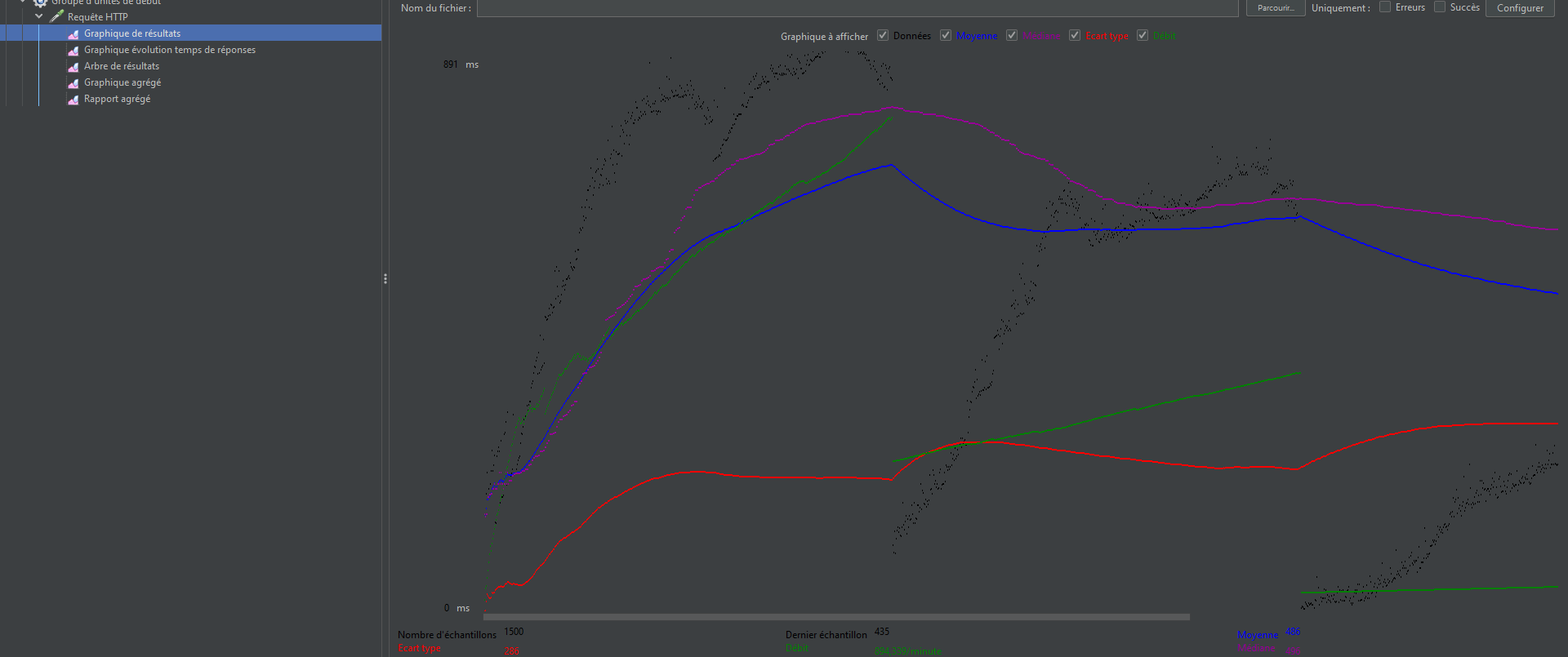
****

On envoie 500 requêtes get, une par utilisateur, dans les différentes valeurs disponibles dans notre api.php afin de les tester.

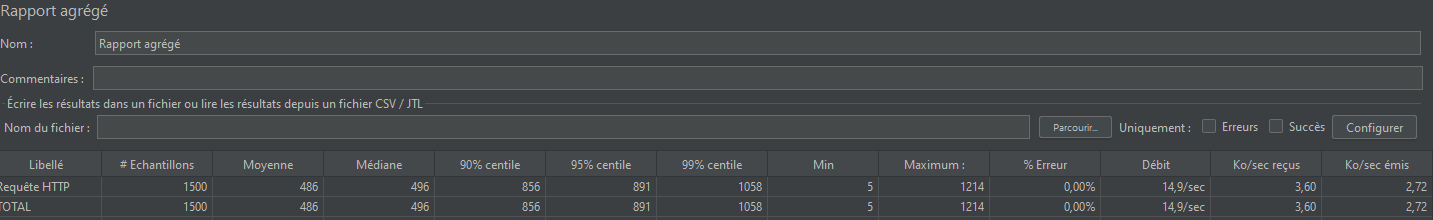
**Graphique agrégé :**



**Graphique de résultat :**



**Rapport agrégé :**



**Analyse des données :**

Dans l’analyse de nos requêtes, on se rend compte que le temps moyen des requêtes est plutôt acceptable mais indéniablement plutôt haut avec **486 ms**, également **496 ms**  en médiane signifie qu’il y a autant de requêtes avec plus ou moins de temps que ça. On considère qu’une requête de **300 ms** à **500 ms** délai enregistre un délai perceptible. Le minimum de délai est de **5 ms**, ce qui est très bas, le maximum lui est de **1214 ms**. C'est un délai clairement perceptible par un utilisateur, néanmoins il n'est pas extrêmement élevé.

On peut conclure de cette analyse que les résultats sont acceptables, compte tenu du nombre de requêtes réparties en peu de temps. Néanmoins, l’application n’est pas considérable comme rapide. L’utilisateur pourra ressentir un léger délai, surtout si les interactions sont fréquentes et que le trafic est élevé.

3. CONCLUSION

Pour conclure, l’application est fonctionnelle, facilement testable et plutôt stable. L’application est facile à tester en end to end et facile à prendre en main. Tous les boutons ou champs sont fonctionnels et les ajouts/modifications/suppressions en base de données fonctionnent.

Néanmoins, elle est sujette à des régressions si on ajoute des modifications conséquentes comme un nouveau champ de formulaire ou en base de données. Les tests ne passeront plus et devront être modifiés. De plus, un fort trafic pourrait mettre en péril l’application. Des optimisations sont sûrement possibles.