

RAPPORT MARTIN CLAVER

1. INTRODUCTION

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

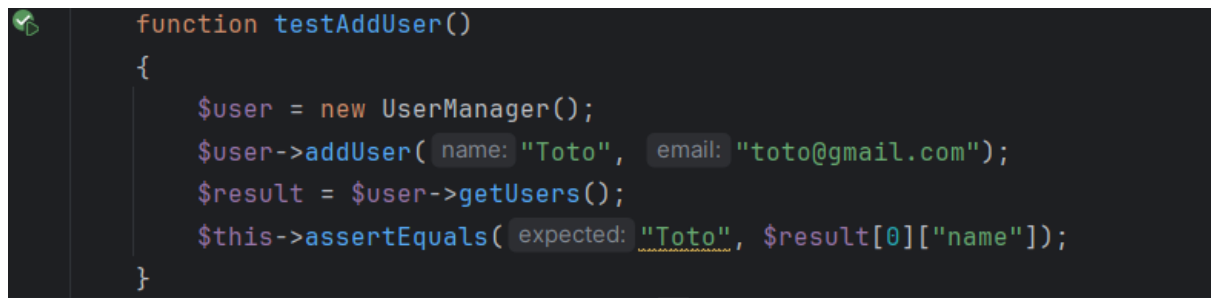
Avec la mise en place des tests fonctionnels, des tests end to end, tests de non régressions et tests de performance. On désignera ensuite les problèmes détectés.

On utilisera PHPUnit, Selenium, JMeter.

2. RÉSULTATS DES TESTS

1. Tests fonctionnels

Test ajout d'un user :

A screenshot of a code editor showing a PHP function named `testAddUser()`. The function is enclosed in curly braces. Inside, it creates a new `UserManager` object, calls `addUser` with the name "Toto" and email "toto@gmail.com", then calls `getUsers` to retrieve the list of users. Finally, it uses `assertEquals` to verify that the first user's name is "Toto".

```
function testAddUser()
{
    $user = new UserManager();
    $user->addUser( name: "Toto", email: "toto@gmail.com");
    $result = $user->getUsers();
    $this->assertEquals( expected: "Toto", $result[0]["name"]);
}
```

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 :

```
15 function testAddUserEmailException()
16 {
17     $user = new UserManager();
18     $this->expectException(InvalidArgumentException::class);
19     $this->expectExceptionMessage("Email invalide.");
20     $user->addUser( name: "Toto", email: "Toto");
21 }
22
```

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 :

```
function testUpdateUser()
{
    $user = new UserManager();
    $user->addUser( name: "Toto", email: "toto@gmail.com");
    $result = $user->getUsers();
    $resultId = $result[0]["id"];
    $user->updateUser($resultId, name: "Tata", email: "toto@gmail.com");
    $updatedResult = $user->getUsers();
    $this->assertEquals( expected: "Tata", $updatedResult[0]["name"]);
}
```

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 :

```
function testRemoveUser()
{
    $user = new UserManager();
    $user->addUser( name: "Toto", email: "toto@gmail.com");
    $result = $user->getUsers();
    $resultId = $result[0]["id"];
    $user->removeUser($resultId);
    $this->assertEmpty($user->getUsers());
}
```

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 :

```
function testGetUsers()
{
    $users = new UserManager();
    $users->addUser( name: "Toto", email: "toto@gmail.com");
    $users->addUser( name: "Tata", email: "tata@gmail.com");
    $result = $users->getUsers();
    $this->assertCount( expectedCount: 2, $result);
}
```

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 :

```
function testInvalidUpdateThrowsException()
{
    $user = new UserManager();
    $this->expectException(Exception::class);
    $this->expectExceptionMessage("Utilisateur introuvable.");
    $user->updateUser($user->getUser( id: 2)["id"], name: "Toto", email: "toto@gmail.com");
}
```

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

```
public function getUser(int $id): array {
    $stmt = $this->db->prepare( query: "SELECT * FROM users WHERE id = :id");
    $stmt->execute(['id' => $id]);
    $user = $stmt->fetch();
    if (!$user) throw new Exception( message: "Utilisateur introuvable.");
    return $user;
}
```

Test invalide à la suppression :

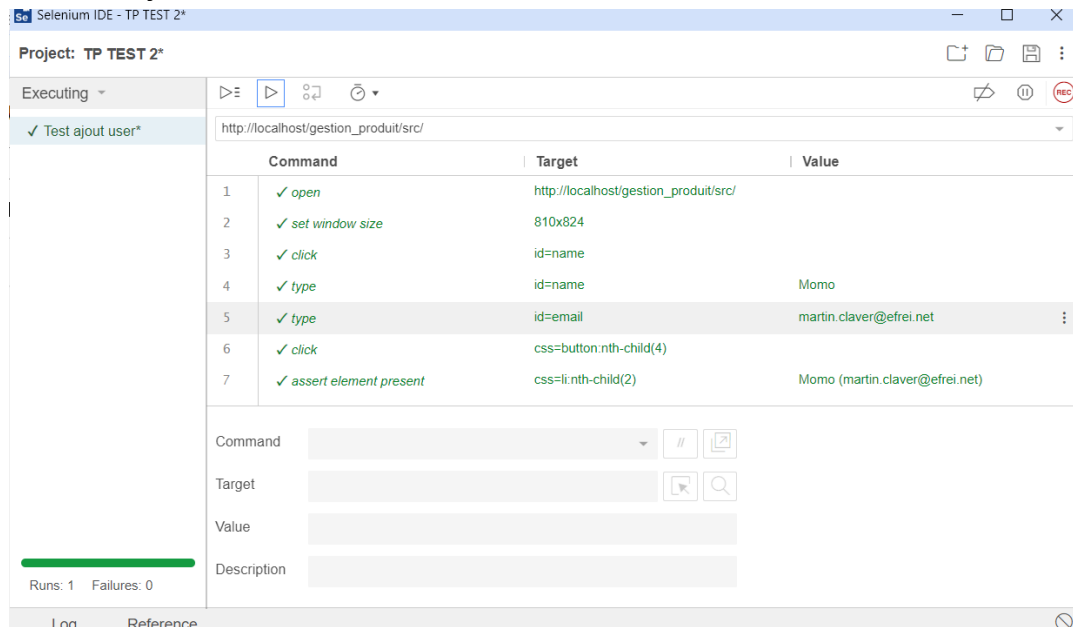
```
function testInvalidDeleteThrowsException()
{
    $user = new UserManager();
    $this->expectException(Exception::class);
    $this->expectExceptionMessage("Utilisateur introuvable.");
    $user->removeUser($user->getUser( id: 2)["id"]);
}
}
```

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

2. Tests End to End

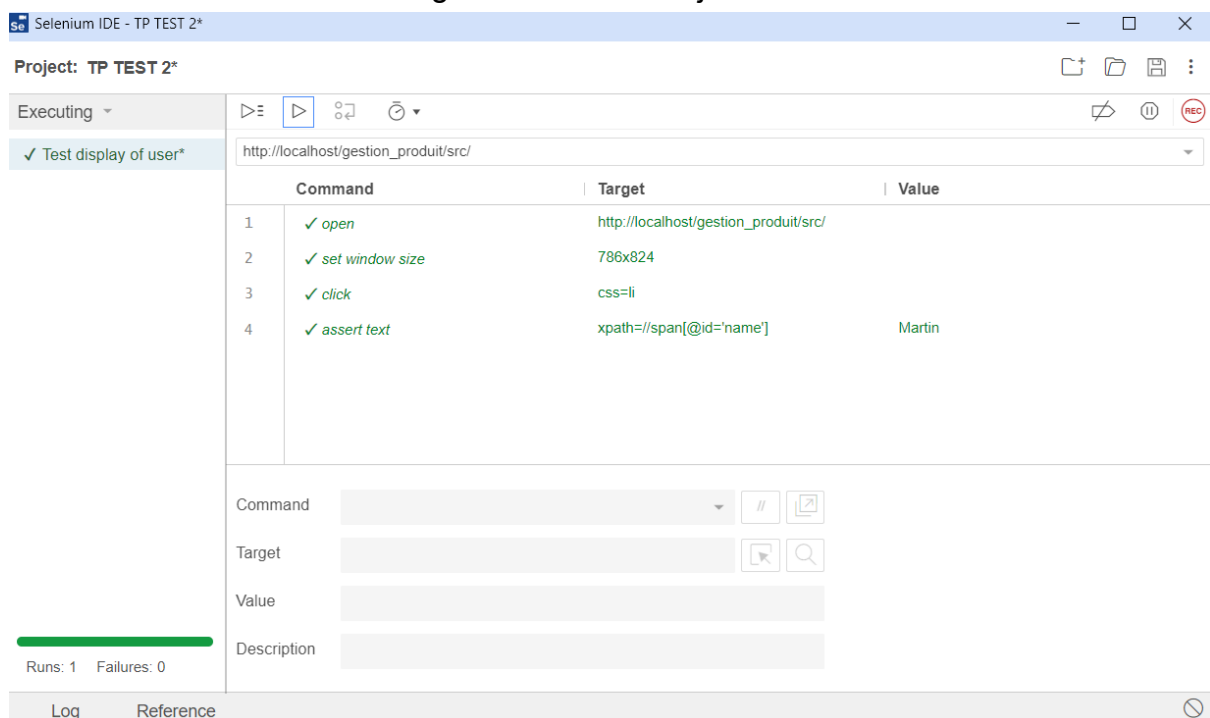
Tests Selenium.

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 :

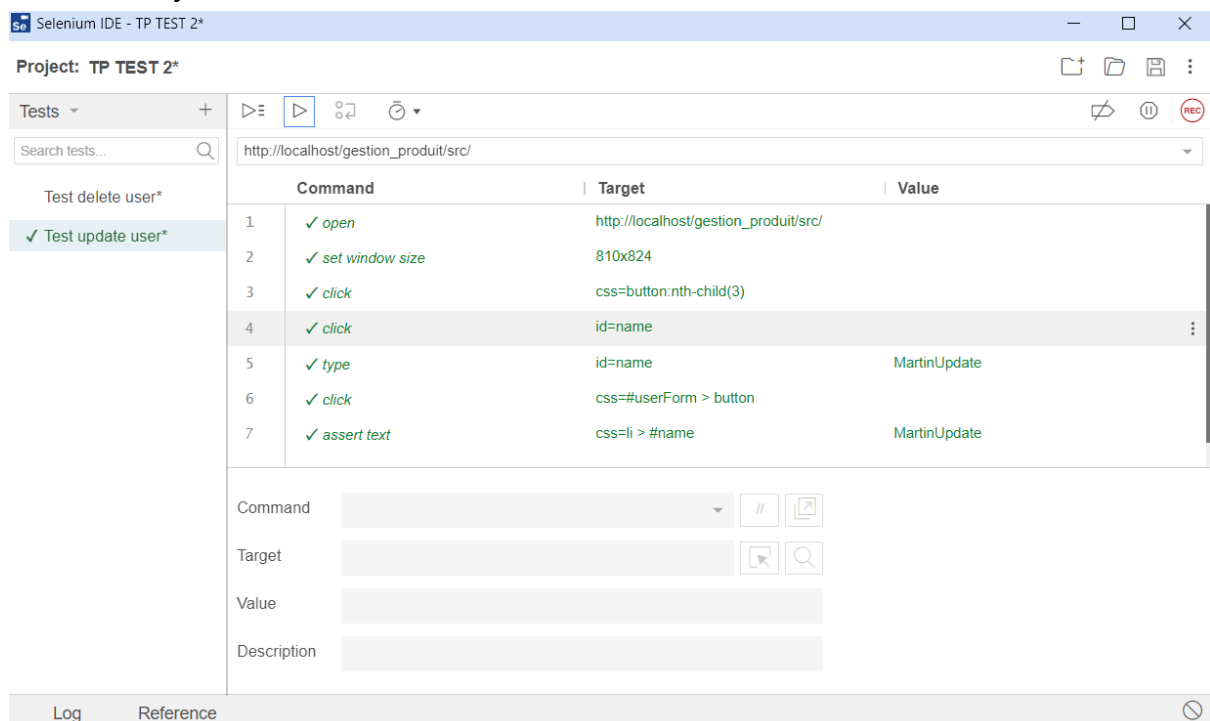
Gestion des utilisateurs

<input type="text" value="Nom"/>	<input type="text" value="Email"/>	<input type="button" value="Ajouter"/>
----------------------------------	------------------------------------	--

Liste des utilisateurs

Martin	(m.claver99@gmail.com)		
--------	------------------------	---	---

Test mise à jour du name de l'utilisateur :



Selenium IDE - TP TEST 2*

Project: TP TEST 2*

Search tests...

Test delete user*

✓ Test update user*

http://localhost/gestion_produit/src/

	Command	Target	Value
1	✓ open	http://localhost/gestion_produit/src/	
2	✓ set window size	810x824	
3	✓ click	css=button:nth-child(3)	
4	✓ click	id=name	
5	✓ type	id=name	MartinUpdate
6	✓ click	css=#userForm > button	
7	✓ assert text	css=li > #name	MartinUpdate

Command

Target

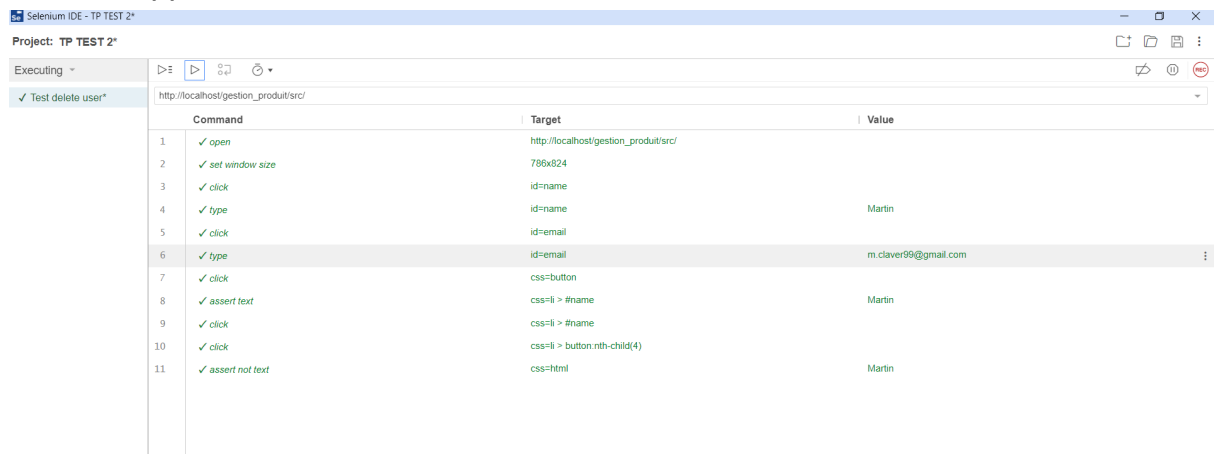
Value

Description

Log Reference

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

Test de suppression de l'utilisateur :



The screenshot shows the Selenium IDE interface. The project is 'TP TEST 2'. The test suite 'Test delete user*' is selected. The URL is 'http://localhost/gestion_produit/src/'. The test consists of 11 steps:

	Command	Target	Value
1	open	http://localhost/gestion_produit/src/	
2	set window size	786x824	
3	click	id=name	
4	type	id=name	Martin
5	click	id=email	
6	type	id=email	m.claver99@gmail.com
7	click	css=button	
8	assert text	css=li > #name	Martin
9	click	css=li > #name	
10	click	css=li > button:nth-child(4)	
11	assert not text	css=html	Martin

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`.

```

1 describe('User Management E2E Test', () => {
2   beforeEach(() => {
3     cy.visit('http://localhost/gestion_produit/src/')
4     cy.get('h1').should('contain', 'Gestion des utilisateurs')
5   })
6
7   it('Ajoute un utilisateur, le modifie, puis le supprime', () => {
8     const userName = 'John Doe';
9     const userEmail = 'john.doe@example.com';
10    const updatedUserName = 'Jane Doe';
11
12    // Ajouter un utilisateur
13    cy.get('input[placeholder="Nom"]').type(userName);
14    cy.get('#email').type(userEmail);
15    cy.get('#userForm').submit();
16
17    // Vérifier que l'utilisateur est bien affiché
18    cy.get('#userList li').should('contain', userName).and('contain', userEmail);
19
20    // Modifier l'utilisateur
21    cy.get('#userList li button').contains('✎').click();
22    cy.get('#name').clear().type(updatedUserName);
23    cy.get('#userForm').submit();
24
25    // Vérifier que le nom de l'utilisateur est bien modifié
26    cy.get('#userList li').should('contain', updatedUserName).and('contain', userEmail);
27
28    // Supprimer l'utilisateur
29    cy.get('#userList li button').contains('✖').click();
30
31    // Vérifier que l'utilisateur est bien supprimé
32    cy.get('#userList li').should('not.exist');
33  });
34 });
35

```

```

✓ User Management E2E Test
  ✓ Ajoute un utilisateur, le modifie, puis le supprime
    ✓ BEFORE EACH
      1  visit http://localhost/gestion_produit/src/
      2  get h1
      3  - assert expected <h1> to contain Gestion des
        utilisateurs

```

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

```
▼ TEST BODY
1  get  input[placeholder="Nom"]
2  -type  John Doe

(fetch) ● GET 200 /gestion_produit/src/api.php
3  get  #email
4  -type  john.doe@example.com
5  get  #userForm
6  -submit

(fetch) ● POST 200
/gestion_produit/src/api.php
(fetch) ● GET 200 /gestion_produit/src/api.php
7  get  #userList li
8  -assert  expected <li> to contain John Doe
9  -assert  expected <li> to contain
john.doe@example.com
```

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


```

10  get #userList li button
11  -contains ✎
12  -click
13  get #name
14  -clear
15  -type Jane Doe
16  get #userForm
17  -submit
    (fetch) ● PUT 200 /gestion_produit/src/api.php
    (fetch) ● GET 200 /gestion_produit/src/api.php
18  get #userList li
19  -assert expected <li> to contain Jane Doe
20  -assert expected <li> to contain
    john.doe@example.com

```

Le champ name est bien modifié.

```

21  get #userList li button
22  -contains ✖
23  -click
    (fetch) ● DELETE 200
    /gestion_produit/src/api.php?id=31
    (fetch) ● GET 200 /gestion_produit/src/api.php
24  get #userList li
25  -assert expected #userList li not to exist
    in the DOM

```

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

3. 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.

<input type="checkbox"/>	1	id	int(11)	Non	Aucun(e)	AUTO_INCREMENT	Modifier	Supprimer	Plus
<input type="checkbox"/>	2	name	varchar(100) utf8mb4_general_ci	Non	Aucun(e)		Modifier	Supprimer	Plus
<input type="checkbox"/>	3	email	varchar(150) utf8mb4_general_ci	Non	Aucun(e)		Modifier	Supprimer	Plus
<input type="checkbox"/>	4	role	varchar(100) utf8mb4_general_ci	Non	Aucun(e)		Modifier	Supprimer	Plus

```
public function addUser(string $name, string $email, string $role): void {
    if (!filter_var($email, filter: FILTER_VALIDATE_EMAIL)) {
        throw new InvalidArgumentException( message: "Email invalide.");
    }

    $stmt = $this->db->prepare( query: "INSERT INTO users (name, email, role) VALUES (:name, :email, :role)");
    $stmt->execute(['name' => $name, 'email' => $email, 'role' => $role]);
}
```

```
public function updateUser(int $id, string $name, string $email, string $role): void {
    $stmt = $this->db->prepare( query: "UPDATE users SET name = :name, email = :email, role = :role WHERE id = :id");
    $stmt->execute(['id' => $id, 'name' => $name, 'email' => $email, 'role' => $role]);
}
```

On relance nos tests précédents.

Test Results 834 ms

Assertion failure

testAddUser 651 ms

testAddUserE 87 ms

testUpdateUs 17 ms

testRemoveUs 3 ms

testGetUsers 3 ms

Tests failed: 5, passed: 2 of 7 tests – 834 ms

u:\xampp\htdocs\gestion_produit\src\userManager.php:24

D:\xampp\htdocs\gestion_produit\tests\testUserManager.php:20

.

ERRORS!

Tests: 7, Assertions: 5, Errors: 4, Failures: 1.

Process finished with exit code 2

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.

```
function testUpdateUser()
{
    $user = new UserManager();
    $user->addUser( name: "Toto", email: "toto@gmail.com", role: "Papa");
    $result = $user->getUsers();
    $resultId = $result[0]["id"];
    $user->updateUser($resultId, name: "Tata", email: "toto@gmail.com", role: "Papa");
    $updatedResult = $user->getUsers();
    $this->assertEquals( expected: "Tata", $updatedResult[0]["name"]);
}
```

Les tests fonctionnels passent !



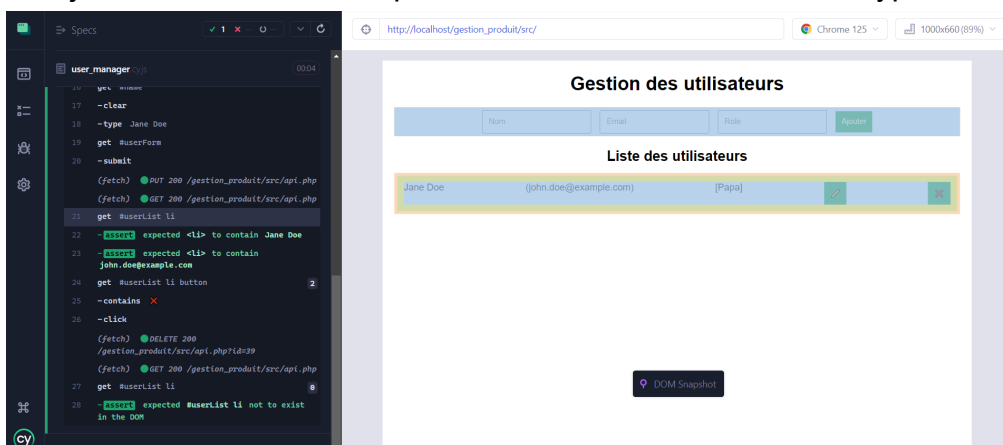
On pourrait imaginer l'amélioration d'implémenter un test end to end qui vérifie que le rôle agit de la même manière que le champ name ou email.

```
it('Ajoute un utilisateur, le modifie, puis le supprime', config().void => {
    const userName :string = 'John Doe';
    const userEmail :string = 'john.doe@example.com';
    const role :string = 'Papa';
    const updatedUserName :string = 'Jane Doe';

    // Ajouter un utilisateur
    cy.get(selector: 'input[placeholder="Nom"]').type(userName);
    cy.get(selector: '#email').type(userEmail);
    cy.get(selector: '#role').type(role);
    cy.get(selector: '#userForm').submit();

    // Vérifier que l'utilisateur est bien affiché
    cy.get(selector: '#userList li').should(chainer: 'contain', userName).and(chainer: 'contain', userEmail).and(chainer: 'contain', role);
});
```

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



4. Tests de performance

Requête HTTP envoyée :

Groupe d'unités de début

Nom :

Commentaires :

Action à suivre après une erreur d'échantillon

☒ Continuer ☐ Démarrer l'itération suivante du Thread ☐ Arrêter l'unité ☐ Arrêter le test ☐ Arrêter le test immédiatement

Propriétés du groupe d'unités

Nombre d'unités (utilisateurs) :

Durée de montée en charge (en secondes) :

Nombre d'itérations : ☐ Infini

Requête HTTP

Nom :

Commentaires :

Basique Avancée

Serveur web

Protocole (http) : Nom ou adresse IP : Port :

Requête HTTP

GET Encodage contenu :

☐ Rediriger automat. ☒ Suivre les redirect. ☒ Connexion persist. ☐ Multiparts/form-data ☐ Entêtes compat. navigateur

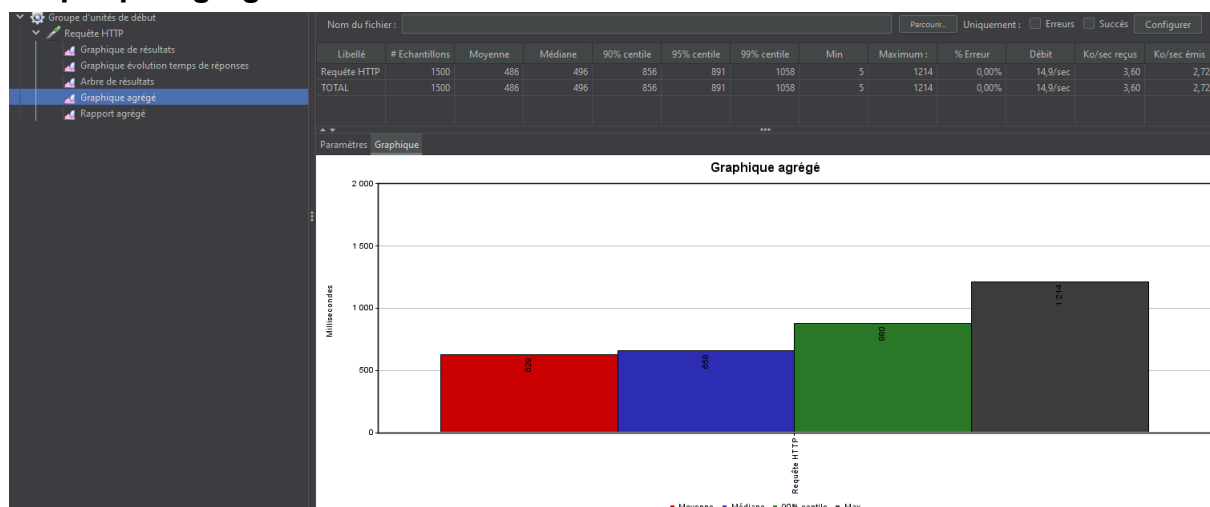
Paramètres Corps de la requête Téléchargement de fichiers

Envoyer les paramètres avec la requête :

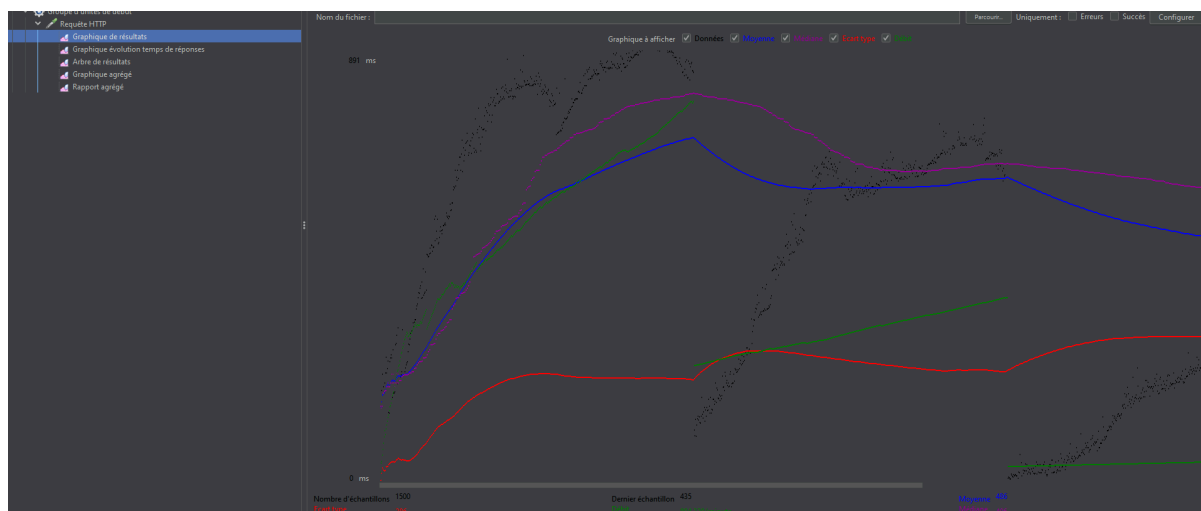
Nom :	Valeur :	URL Encoder	Content-Type	Inclure égal ?
name	Toto	<input type="checkbox"/>	text/plain	<input checked="" type="checkbox"/>
email	name\$[_counter(FALSE)]@example.com	<input type="checkbox"/>	text/plain	<input checked="" type="checkbox"/>
role	Papa	<input type="checkbox"/>	text/plain	<input checked="" type="checkbox"/>

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 :

Rapport agrégé

Nom :

Rapport agrégé

Commentaires :

Ecrire les résultats dans un fichier ou lire les résultats depuis un fichier CSV / JTL

Nom du fichier :

Parcourir...

Uniquement : ☐ Erreurs ☐ Succès

Configurer

Libellé	# Echantillons	Moyenne	Médiane	90% centile	95% centile	99% centile	Min	Maximum :	% Erreur	Débit	Ko/sec reçus	Ko/sec émis
Requête HTTP	1500	486	496	856	891	1058	5	1214	0,00%	14,9/sec	3,60	2,72
TOTAL	1500	486	496	856	891	1058	5	1214	0,00%	14,9/sec	3,60	2,72

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.