École Supérieure Privée d'Ingénierie et de Technologies

Rapport de stage d'immersion en entreprise

Développement de tests automatisés

Présenté et soutenu par
Trabelsi Mohamed

Année universitaire
2023 - 2024

Dirigé par
M. Noomene Achref

Table des matières

Liste des figures

Remerciements

Je tiens à exprimer ma sincère gratitude à toutes les personnes qui ont contribué de près ou de loin à la réalisation de ce stage chez Swiver. Leur soutien, leurs conseils et leur encouragement ont été essentiels pour mener à bien ce projet de test automatisé.

Tout d'abord, je tiens à remercier chaleureusement la société Swiver pour m'avoir accordé cette opportunité de stage. Leur confiance en mes compétences et leur ouverture à l'innovation ont été des facteurs clés dans la réussite de ce projet. J'ai pu découvrir un environnement de travail dynamique et stimulant, propice à l'apprentissage et à l'épanouissement professionnel.

Je souhaite également exprimer ma gratitude à mon encadrant de stage, Achref, pour sa guidance et son expertise tout au long de ce projet. Ses conseils avisés, sa disponibilité et son soutien constant ont été précieux. Sa patience et sa volonté de partager ses connaissances ont grandement contribué à mon développement professionnel.

Un remerciement spécial s'adresse également à la direction de l'ESPRIT, l'établissement dans lequel j'ai suivi ma formation, pour leur encadrement et leur accompagnement tout au long de mon cursus universitaire. Leur engagement en faveur de l'excellence académique et de la préparation des étudiants au monde professionnel a été d'une importance capitale pour ma réussite.

Enfin, je tiens à remercier toute l'équipe de développement de Swiver pour leur accueil chaleureux, leur collaboration et leur esprit d'équipe. Leur expertise et leur enthousiasme ont été une source d'inspiration pour moi, et j'ai apprécié travailler à leurs côtés.

Introduction générale

La plupart des projets mis en place par les entreprises visent à atteindre un certain nombre d'objectifs et à s'adapter plus efficacement à l'environnement externe. Toutefois, le succès d'un projet ne repose pas uniquement sur la livraison d'une solution finale, mais également sur la garantie de sa qualité et de sa fiabilité. C'est là que le test des applications entre en jeu.

Le test des applications est essentiel pour les entreprises de tous les secteurs. En effet, les applications bloquées peuvent avoir des conséquences néfastes telles que des pertes financières, une baisse de la satisfaction des clients et des dommages à la réputation de l'entreprise. Ainsi, il est essentiel d'investir suffisamment de ressources et d'efforts dans la phase de test pour détecter et résoudre les problèmes avant de passer en production.

Tester une application implique une approche méthodique et rigoureuse. Il est nécessaire d'utiliser différentes techniques et méthodologies de test telles que les tests fonctionnels, les tests de performance, les tests de sécurité, etc. En outre, l'utilisation d'outils et de cadres de test spécialisés peut automatiser une partie du processus, ce qui contribue à accélérer les tests et à améliorer leur efficacité.

Dans le cadre de ce rapport de stage, nous aborderons le sujet du test automatisé des applications et son importance pour les entreprises. Nous présenterons également notre travail au sein de l'entreprise Swiver, où nous décrirons la méthodologie de travail adoptée, les outils utilisés et les réalisations obtenues. L'objectif ultime est de mettre en lumière l'impact positif du test automatisé sur la qualité des applications développées et sur la réussite des projets au sein des entreprises. En outre, nous soulignerons l'importance de la collaboration avec d'autres départements de l'entreprise, tels que le développement, pour garantir une mise en production réussie et une expérience utilisateur optimale.

Enfin, nous aborderons également les défis auxquels les entreprises peuvent être confrontées lorsqu'elles mettent en place un processus de test automatisé et les stratégies pour les surmonter.

Chapitre 1

Présentation du cadre du projet

1.1 Introduction

Dans cette section, nous allons présenter le cadre de projet dans lequel s'inscrit notre stage chez Swiver. Nous commencerons par une introduction à l'entreprise Swiver, puis nous aborderons l'étude de l'existant, une critique des méthodes actuelles, la solution proposée et les besoins fonctionnels et non fonctionnels identifiés.

1.2 Présentation de Swiver

Swiver est une entreprise spécialisée dans le développement de logiciels de gestion commerciale destinés aux TPE & PME, aux artisans professionnels et aux commerçants. Avec plus de 13 000 utilisateurs, elle propose une solution complète pour faciliter la gestion quotidienne des entreprises et optimiser leur performance.

Parmi les fonctionnalités clés de Swiver, vous pouvez créer et personnaliser vos factures en quelques clics, suivre facilement vos clients, gérer vos achats et dépenses, et automatiser vos entrées et sorties de stock. De plus, leur tableau de bord intuitif vous offre une vision globale de vos flux de trésorerie en temps réel, facilitant ainsi votre prise de décision.

1.3 Étude de l'existant

Les tests manuels impliquaient l'exécution répétitive de chaque scénario de test, ce qui prenait beaucoup de temps et de ressources. Les testeurs devaient suivre un ensemble d'étapes prédéfinies, vérifier les résultats obtenus et noter les problèmes rencontrés.

1.4 Critique de l'existant

Après avoir étudié en détail le processus de test existant, nous avons identifié plusieurs lacunes et limites significatives. Voici les principales critiques formulées :

— Consommation de temps et d'énergie : Le test manuel impliquait une exécution répétitive des scénarios de test, ce qui prenait beaucoup de temps et d'énergie. Les testeurs devaient effectuer manuellement chaque étape du scénario, ce qui était une tâche fastidieuse et répétitive.

— Erreur humaine : L'approche manuelle du test était sujette aux erreurs humaines. Les testeurs pouvaient commettre des erreurs lors de l'exécution des scénarios ou de la détection des erreurs, ce qui entraînait des résultats de test incohérents ou des erreurs non détectées.

1.5 Solution proposée

Après avoir identifié les limitations et les lacunes du processus de test existant, nous avons proposé une solution pour améliorer l'efficacité et la fiabilité des tests chez Swiver : l'adoption du test automatisé. Le test automatisé offre de nombreux avantages, notamment en termes de gain de temps, de réduction des erreurs humaines et d'amélioration de la couverture des tests.

Le test automatisé présente plusieurs avantages significatifs par rapport au test manuel :

— Gain de temps et d'effort

— Réduction des erreurs humaines

— Meilleure couverture des tests

— Rapidité et fiabilité des tests

1.6 Besoins fonctionnels

Il s'agit des fonctionnalités du système. C'est l'ensemble des actions que le système fournit pour satisfaire les besoins de l'utilisateur :

- Création de scénarios de test automatisé : Il est essentiel de pouvoir créer facilement et efficacement des scénarios de test automatisé pour chaque fonctionnalité de l'application. Cela permettra de garantir une couverture complète des tests.
- Exécution automatisée des tests : Les tests automatisés doivent être exécutés de manière automatique et systématique pour garantir leur cohérence et leur répétabilité. Une exécution automatisée permet également de gagner du temps et de libérer les testeurs pour des tâches plus critiques.
- Validation des résultats attendus : Les tests automatisés doivent être capables de valider les résultats attendus de manière fiable. Cela implique de vérifier que les fonctionnalités de l'application se comportent conformément aux spécifications et aux exigences définies.

1.7 Besoins non fonctionnels

- Performance : Les tests automatisés doivent être rapides et efficaces pour éviter les retards dans le processus de test et minimiser les temps d'exécution.
- Fiabilité : Les tests automatisés doivent être fiables et produire des résultats cohérents à chaque exécution. Cela garantit une détection précise des erreurs et une évaluation précise de la qualité de l'application.
- Maintenabilité : Les tests automatisés doivent être faciles à maintenir et à mettre à jour à mesure que l'application évolue. Il est important d'avoir des tests robustes qui restent pertinents et fonctionnels tout au long du cycle de vie de l'application.

Chapitre 2

La méthodologie de travail et les outils adoptés

2.1 Méthodologie Agile et Scrum

Pour mener à bien notre projet, nous avons choisi de suivre une approche agile, avec une mise en œuvre spécifique de la méthode Scrum. La méthodologie Agile nous a permis de travailler de manière itérative et collaborative, en nous adaptant aux changements et en assurant une communication transparente au sein de l'équipe.

2.2 Outils utilisés

Pour la réalisation de notre projet de test automatisé chez Swiver, nous avons utilisé plusieurs outils qui ont joué un rôle essentiel dans notre travail. Voici les principaux outils que nous avons adoptés :

2.2.1 Node.js

FIGURE 2.1 – Node.js

Node.js est une plateforme de développement basée sur JavaScript qui nous a permis de mettre en place l'infrastructure nécessaire pour le test automatisé. Grâce à Node.js, nous avons pu exécuter nos scripts de test, manipuler les données et interagir avec les bibliothèques de test.

2.2.2 React.js

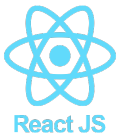

FIGURE 2.2 – React.js

React.js est une bibliothèque JavaScript largement utilisée pour le développement d'interfaces utilisateur. Nous avons utilisé React.js pour développer des composants de test réutilisables, simplifiant ainsi la création et l'organisation de nos scénarios de test.

2.2.3 Jest

FIGURE 2.3 – Jest

Jest est un framework de test JavaScript populaire qui nous a permis d'écrire et d'exécuter nos tests automatisés de manière efficace. Jest offre des fonctionnalités avancées pour l'écriture de tests, la gestion des assertions et la génération de rapports de test détaillés.

2.2.4 React Testing Library (RTL)

FIGURE 2.4 – React Testing Library

La bibliothèque React Testing Library est spécifiquement conçue pour tester les composants React. Elle nous a permis de simuler les interactions utilisateur et de vérifier les résultats attendus lors de l'exécution de nos tests automatisés.

L'utilisation de ces outils nous a offert une base solide pour la mise en place du test automatisé chez Swiver. Ils ont facilité le développement de scénarios de test, l'exécution des tests automatisés et l'analyse des résultats.

2.3 Environnement logiciel

2.3.1 Visual Studio Code

FIGURE 2.5 – Visual Studio Code

Visual Studio Code est un éditeur de code extensible développé par Microsoft pour Windows, Linux et macOS. Les fonctionnalités incluent la prise en charge du débogage, la mise en évidence de la syntaxe, la complétion intelligente du code, les snippets, la refactorisation du code et Git intégré. Les utilisateurs peuvent modifier le thème, les raccourcis clavier, les préférences et installer des extensions qui ajoutent des fonctionnalités supplémentaires.

Chapitre 3

Réalisation

3.1 Sprint 1 - Auto-formation sur React.js et étude des bibliothèques de test

Dans cette section, nous aborderons le contenu du premier sprint de trois semaines de notre projet chez Swiver. Ce sprint était dédié à notre auto-formation sur React.js, à la recherche sur les tests et à l'étude des bibliothèques Jest et React Testing Library (RTL).

3.1.1 Auto-formation sur React.js

Pour commencer ce sprint, j'ai entrepris une auto-formation sur React.js, qui est une bibliothèque JavaScript largement utilisée pour la construction d'interfaces utilisateur interactives. J'ai effectué des recherches approfondies sur les concepts fondamentaux de React.js, tels que les composants, le rendu virtuel, les propriétés (props) et les états (state). J'ai également étudié les principes clés tels que la réutilisabilité des composants et le concept de flux unidirectionnel des données.

Pour approfondir ma compréhension, j'ai réalisé des petits projets pratiques qui m'ont permis de mettre en pratique les concepts de React.js. J'ai créé des composants, géré les états, utilisé les propriétés pour passer des données entre les composants, et exploité les fonctionnalités de rendu conditionnel et de gestion des événements. Cette phase d'auto-formation m'a permis d'acquérir une solide compréhension des fondamentaux de React.js.

3.1.2 Recherche sur les tests et comparatif des bibliothèques

La deuxième partie de ce sprint était dédiée à la recherche sur les tests dans le contexte de développement d'applications React.js. J'ai exploré les différents types de tests, tels que les tests unitaires, les tests d'intégration et les tests d'interface utilisateur. J'ai étudié l'importance des tests automatisés et les avantages qu'ils offrent en termes de fiabilité, de maintenabilité et de réduction des erreurs.

Dans le cadre de ma recherche, j'ai également examiné les bibliothèques de test populaires, à savoir Jest et React Testing Library (RTL). J'ai comparé ces deux bibliothèques en termes de fonctionnalités, de facilité d'utilisation, de performance et de prise en charge de React.js. J'ai étudié la manière dont ces bibliothèques permettaient d'écrire des tests, de simuler les interactions utilisateur et de vérifier les résultats attendus.

Après avoir réalisé cette étude comparative, j'ai préparé une présentation pour l'équipe de développement chez Swiver. Cette présentation a permis de partager mes connaissances et mes recommandations sur l'utilisation de Jest et RTL pour le test automatisé des applications React.js.

3.2 Sprint 2 - Tests automatisés et mise en place sur la template Swiver

Dans ce chapitre, nous nous concentrons sur le contenu du deuxième sprint de trois semaines de notre projet chez Swiver. Ce sprint était dédié à la réalisation des tests automatisés et à l'intégration de ces tests sur la template du site web Swiver.

3.2.1 Réalisation des tests automatisés

Pour commencer ce sprint, j'ai utilisé les connaissances acquises lors du premier sprint pour développer des exemples de tests automatisés. En me basant sur les fonctionnalités clés de l'application Swiver, j'ai écrit des scénarios de test qui couvraient différents aspects.

J'ai utilisé les bibliothèques Jest et React Testing Library pour créer et exécuter ces tests automatisés. J'ai vérifié les résultats attendus en simulant les interactions utilisateur et en vérifiant les éléments de l'interface utilisateur à l'aide de sélecteurs spécifiques.

Chaque test automatisé a été soigneusement conçu pour couvrir les cas d'utilisation les plus importants et pour s'assurer que les fonctionnalités essentielles de l'application Swiver étaient correctement testées. J'ai également

pris en compte les différents scénarios d'erreur et les cas limites afin d'améliorer la robustesse des tests automatisés.

3.2.2 Intégration des tests automatisés sur la template Swiver

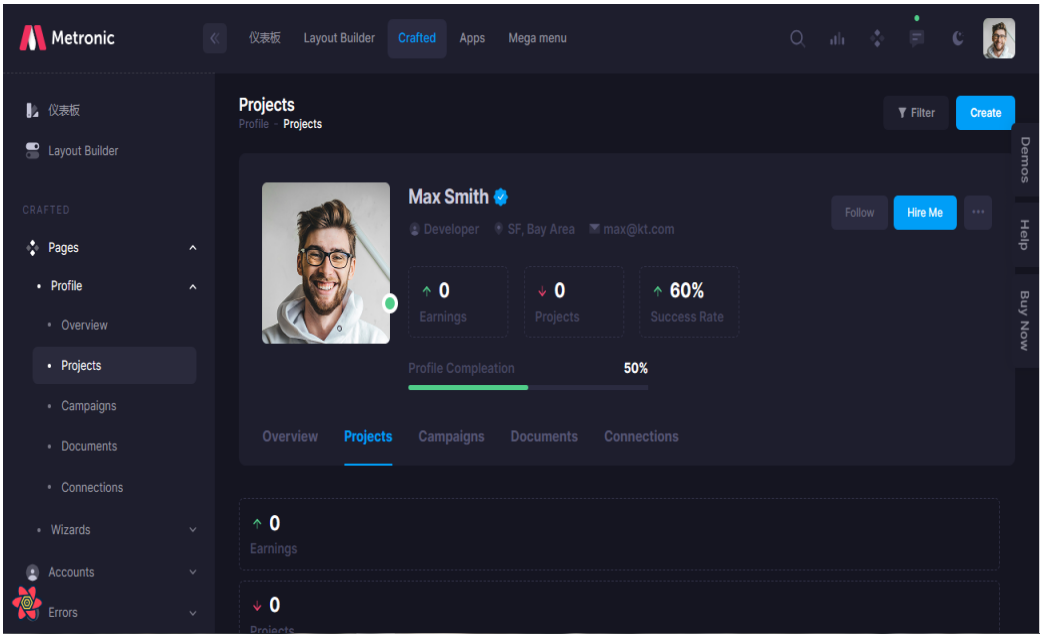

FIGURE 3.1 – Template Intégrée

Une fois les tests automatisés réalisés, j'ai procédé à leur intégration sur la template du site web Swiver. J'ai utilisé la structure existante de la template pour ajouter les tests automatisés aux différentes pages et fonctionnalités de l'application. Cela a permis d'assurer une couverture complète des tests sur la template Swiver et de garantir que les nouvelles fonctionnalités ajoutées seraient également testées de manière automatisée.

3.2.3 Rédaction d'un fichier Readme

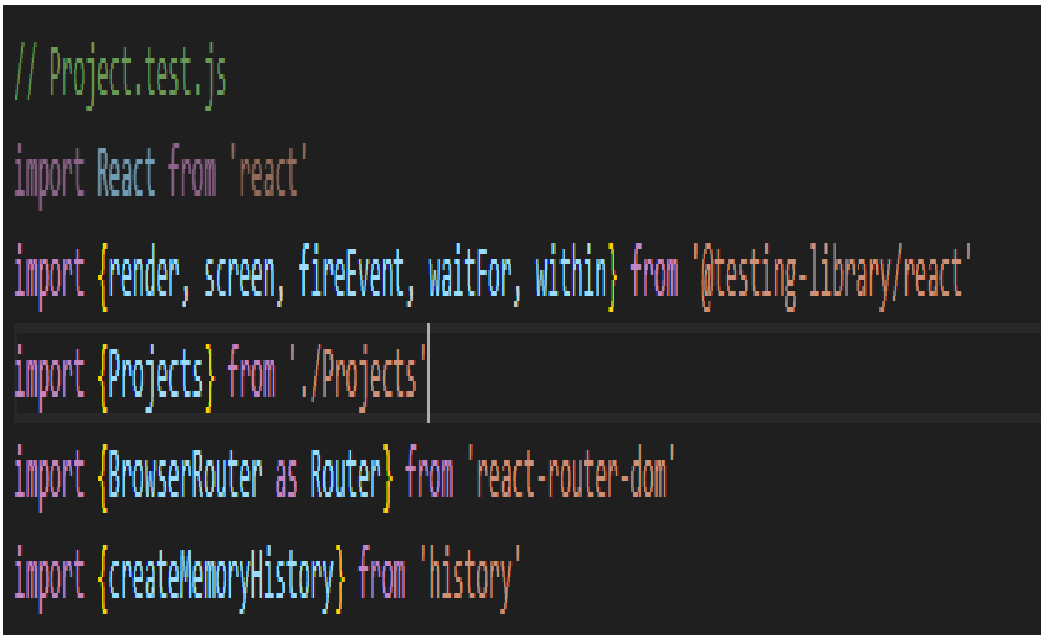

FIGURE 3.2 – Importation

En parallèle, j'ai rédigé un fichier README.md détaillé qui contenait une description complète de tous les tests automatisés réalisés. Ce document servait de guide pour l'équipe de développement et facilitait la compréhension et l'utilisation des tests automatisés. J'y ai inclus des informations sur les fonctionnalités couvertes, les scénarios testés, les résultats attendus et les dépendances requises.

Les imports dans le code fourni sont utilisés pour importer des modules et des fonctions spécifiques provenant de différentes bibliothèques et fichiers. Voici une explication de chaque importation :

render, screen, fireEvent, waitFor, within : Ces fonctions sont importées depuis la bibliothèque @testing-library/react. Elles sont couramment utilisées pour effectuer des tests sur les composants React. Par exemple, render est utilisé pour rendre un composant, screen est utilisé pour accéder aux éléments de l'interface utilisateur, fireEvent est utilisé pour déclencher des événements sur les éléments, waitFor est utilisé pour attendre que des conditions spécifiques soient remplies, et within est utilisé pour effectuer des sélections spécifiques à l'intérieur d'un élément parent.

Projects : Cet import provient du fichier './Projects' et il fait référence à un composant Projects spécifique. Ce composant est utilisé dans le test pour

simuler son rendu et effectuer des assertions sur son comportement.

BrowserRouter as Router : Cet import provient de la bibliothèque 'react-router-dom' et il permet d'utiliser le composant BrowserRouter en l'aliasant avec le nom Router. Le composant BrowserRouter est utilisé pour gérer les routes dans une application React.

createMemoryHistory : Cet import provient de la bibliothèque 'history' et il permet de créer une instance de history en mémoire. Cette instance de history est utilisée pour gérer l'historique des pages et les changements d'URL dans le test.

En utilisant ces imports, le code est en mesure de rendre le composant Projects avec un Router spécifique, d'interagir avec les éléments de l'interface utilisateur à l'aide de fonctions de la bibliothèque @testing-library/react, et de créer une instance de history en mémoire pour gérer les routes dans le test.

3.2.4 Exemple de réalisation d'un test

```
test('sorting projects by budget in descending order', () => {
  render(
    <Router>
      <Projects />
    </Router>
  )
  // Click on "New Project" button
  fireEvent.click(screen.getByTestId('create-project-form'))
  // Fill out and submit the first project form
  fireEvent.change(screen.getByLabelText(/Title/i), {target: {value: 'Project 1'}})
  fireEvent.change(screen.getByLabelText(/Description/i), {
    target: {value: 'Description of Project 1'},
  })
  fireEvent.change(screen.getByLabelText(/Due Date/i), {target: {value: '2023-07-15'}})
  fireEvent.change(screen.getByLabelText(/Budget/i), {target: {value: '1000'}})
  fireEvent.click(screen.getByTestId('create-project-button'))
  // Click on "New Project" button again
  fireEvent.click(screen.getByTestId('create-project-form'))
  // Fill out and submit the second project form
  fireEvent.change(screen.getByLabelText(/Title/i), {target: {value: 'Project 2'}})
  fireEvent.change(screen.getByLabelText(/Description/i), {
    target: {value: 'Description of Project 2'},
  })
  fireEvent.change(screen.getByLabelText(/Due Date/i), {target: {value: '2023-07-20'}})
  fireEvent.change(screen.getByLabelText(/Budget/i), {target: {value: '500'}})
  fireEvent.click(screen.getByTestId('create-project-button'))
  // Click on "New Project" button again
  fireEvent.click(screen.getByTestId('create-project-form'))
  // Fill out and submit the third project form
  fireEvent.change(screen.getByLabelText(/Title/i), {target: {value: 'Project 3'}})
  fireEvent.change(screen.getByLabelText(/Description/i), {
    target: {value: 'Description of Project 3'},
  })
  fireEvent.change(screen.getByLabelText(/Due Date/i), {target: {value: '2023-07-25'}})
  fireEvent.change(screen.getByLabelText(/Budget/i), {target: {value: '800'}})
  fireEvent.click(screen.getByTestId('create-project-button'))
  // Click on the "Tri décroissant" button
  const sortButton = screen.getByTestId('sort-button');
  fireEvent.click(sortButton);
  // Get the project cards
  const projectCards = screen.getAllByTestId('project-card');
  // Check if the projects are sorted correctly
  expect(projectCards).toHaveLength(3);
  expect(projectCards[0]).toHaveTextContent('Project 1');
  expect(projectCards[1]).toHaveTextContent('Project 3');
  expect(projectCards[2]).toHaveTextContent('Project 2');
});
```

FIGURE 3.3 – Exemple d'un code de test automatisé

Dans cette section, nous présenterons un exemple concret d'implémentation d'un test automatisé. L'exemple ci-dessous illustre un test qui vérifie le tri décroissant des projets en fonction du budget dans une application. Nous expliquerons chaque étape du test pour mieux comprendre son fonctionnement.

Tout d'abord, nous définissons un test avec la fonction test(), en lui donnant un nom descriptif.

Nous rendons le composant ¡Projects /¿ à l'aide de la fonction render(), en enveloppant le composant avec un ¡Router¿ pour gérer les routes dans l'application.

Nous effectuons une série d'actions pour simuler la création de trois projets avec différents budgets. Nous cliquons sur le bouton "New Project", remplissons les formulaires de projet avec les détails requis, puis soumettons les formulaires en cliquant sur le bouton de création du projet.

Ensuite, nous cliquons sur le bouton "Tri décroissant" pour déclencher le tri des projets en fonction du budget.

Nous récupérons les cartes de projet à l'aide de la fonction getAllByTestId().

Enfin, nous effectuons des assertions pour vérifier si les projets sont triés correctement. Dans cet exemple, nous vérifions si les projets sont affichés dans l'ordre "Project 1", "Project 3" et "Project 2".

Ce test illustre la manière dont les tests automatisés peuvent être écrits pour vérifier le bon fonctionnement des fonctionnalités clés de l'application. En exécutant ce test, nous pouvons nous assurer que le tri des projets par budget est effectué correctement.

N'hésitez pas à personnaliser cette explication en fonction des spécificités de votre propre exemple de test et à fournir davantage de détails ou d'explications supplémentaires si nécessaire.

3.2.5 Conclustion

Au terme de ce premier sprint, j'ai acquis une solide compréhension des fondamentaux de React.js grâce à une auto-formation intensive. J'ai également réalisé une recherche approfondie sur les tests dans le contexte de développement d'applications React.js, en mettant l'accent sur les bibliothèques Jest et React Testing Library.

Au terme de ce deuxième sprint, j'ai réussi à développer des tests automatisés exhaustifs pour les fonctionnalités clés de l'application Swiver. Ces tests ont été intégrés avec succès sur la template du site web Swiver, assurant une couverture complète des tests automatisés.

Le fichier [README.md](http ://readme.md/) que j'ai créé a servi de référence pour l'équipe de développement, fournissant des informations détaillées

sur les tests automatisés réalisés. Cela a facilité la compréhension des tests et a permis à l'équipe de les utiliser efficacement pour la validation des fonctionnalités et la détection des erreurs.

Conclusion générale

En conclusion, je tiens à exprimer ma gratitude envers toutes les personnes qui ont contribué à la réalisation de ce projet de test automatisé chez Swiver. Leur soutien, leur expertise et leur collaboration ont été essentiels pour mener ce projet à bien.

Je souhaite tout d'abord remercier l'équipe de développement chez Swiver pour leur accueil chaleureux, leur disponibilité et leur collaboration tout au long du projet. Leur expertise technique et leur engagement envers l'excellence ont été des facteurs clés dans la réussite de ce projet.

Un remerciement spécial s'adresse à mon encadrant de stage, qui m'a guidé et soutenu tout au long de ce projet. Sa disponibilité, ses conseils avisés et son expertise ont été d'une grande valeur pour moi. Je suis reconnaissant d'avoir eu l'opportunité de travailler avec lui et d'apprendre de ses connaissances.

Je tiens également à remercier l'équipe de test chez Swiver pour leur contribution à la réalisation des tests automatisés. Leur expérience et leur implication ont été précieuses pour identifier les scénarios de test pertinents et assurer la qualité des fonctionnalités testées.

Enfin, je souhaite exprimer ma gratitude envers la direction de Swiver pour avoir accordé leur confiance et leur soutien à ce projet. Leur engagement en faveur de l'innovation et de l'amélioration continue a créé un environnement propice à la réussite de ce projet.

En conclusion, le projet de test automatisé chez Swiver a été une expérience enrichissante et valorisante. Grâce à l'adoption du test automatisé et à la collaboration de toutes les parties prenantes, nous avons pu améliorer la qualité des applications et renforcer la confiance des utilisateurs. Je suis reconnaissant d'avoir participé à ce projet et j'espère que les bénéfices du test automatisé perdureront dans l'avenir.