

Roombâ

LE CONTEXTE

Roombâ est une startup qui a été créée il y a un an et qui repose principalement sur la proposition de location de salle événementielle pour des particuliers ou des entreprises. L'objectif de cette entreprise repose notamment à faciliter la location de salles qui peuvent parfois être oubliées ou inconnues.



L'entreprise est composée de deux entrepreneurs, monsieur Pechberty et monsieur Foujols qui dirigent le projet et le recrutement.

Par ailleurs, ils souhaitent également proposer ce type de service afin d'accélérer la transaction des salles entre la mairie et les locataires.

Leur entreprise se situe à Bercy et plus précisément à Station F où ils sont installés depuis un peu plus d'un an et où ils peuvent travailler sur leur projet.

Puisque leur entreprise repose sur la possibilité de louer des salles événementielles à travers la mairie, ils entretiennent un site web qui dispose de nombreuses fonctionnalités pouvant être à risque si les données ne sont pas protégées correctement.

Il était donc important de faire un site web qui protège toutes les données sensibles en limitant le plus d'erreurs et en testant leur site pour voir si des failles pouvaient exister.

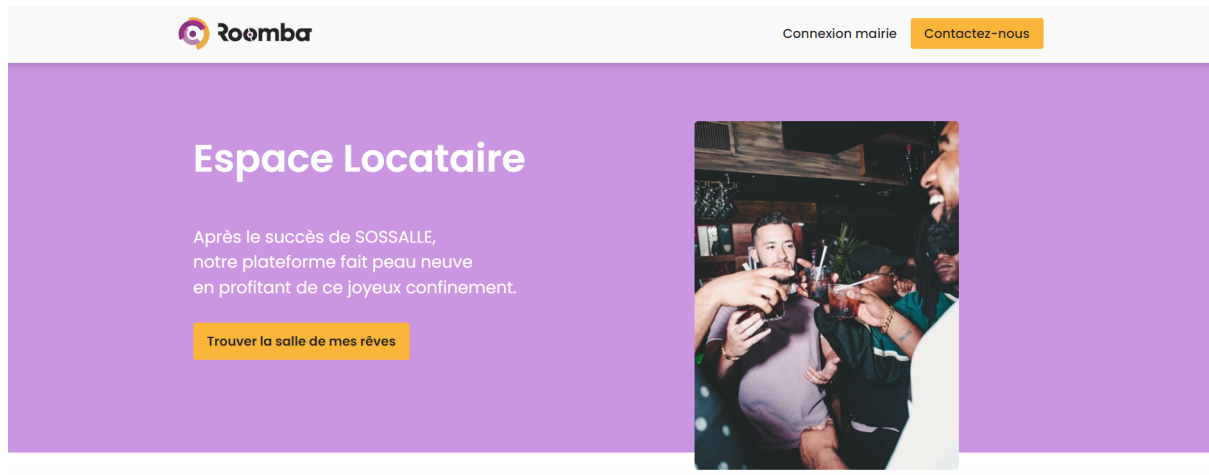
Mon projet durant ces 5 semaines de stage s'est donc divisé en deux étapes:

- Tout d'abord j'ai appris et utilisé le langage Cypress pour les test end2end
- Ensuite, j'ai appris le langage javascript et la librairie jest pour effectuer les test unitaires du site web

EXPRESSION DES BESOINS

Lorsque je suis arrivé dans l'entreprise, le site de Roombâ était déjà bien avancé et bien développé. Comme expliqué précédemment, le site web de Roombâ repose sur l'affichage des salles événementielles pouvant être louées pour les particuliers et les entreprises.

Comme l'entreprise est encore jeune, l'objectif ici était encore de tester le site web afin de voir s'il existait des bugs ou des failles pouvant impliquer une difficulté à utiliser le site correctement ou à vouloir voler des données sensibles.



Le site est divisé en deux espaces différents, l'espace client et l'espace mairie qui représente l'entreprise Roombâ ou l'on peut se connecter.

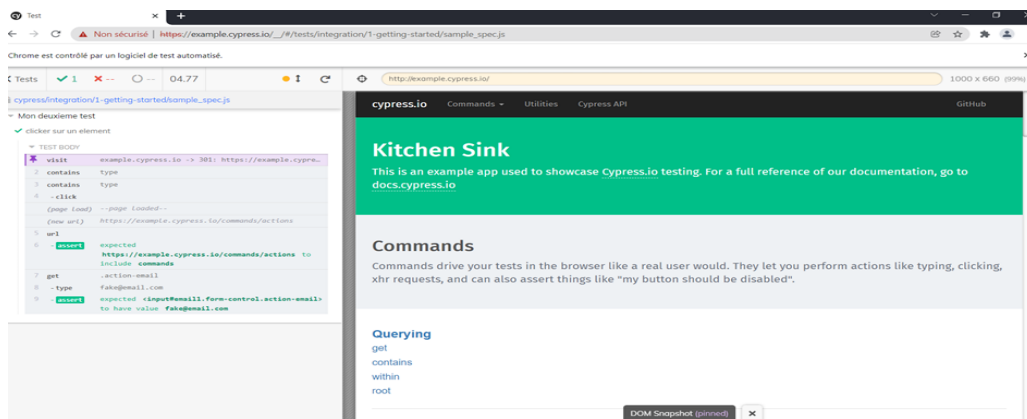
Durant mon stage, deux tâches m'ont donc été demandées:

- des tests unitaires avec Jest
- et des tests end2end avec Cypress.

Ma première tâche reposait sur l'utilisation du front end Cypress qui est une nouvelle application utilisée par les entreprises pour tester leur site à travers la simulation client.

Son utilisation permet alors de voir si le site est bien fonctionnel en testant l'intégralité des fonctionnalités disponible sur le site en tant que client et même administrateur.

Son utilisation s'effectue directement à travers l'application, une fois installée, et permet de tester un fichier ou l'on écrit des tests sur un site web donné.



Au cours de ma 2ème tâche, j'ai utilisé le framework Jest et ses tests unitaires.

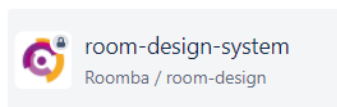
Le framework Jest permet tout d'abord de tester l'application et de simuler des actions afin de voir si le résultat est bien celui attendu.

En effet, les tests unitaires permettent de voir si les applicatifs des sites sont fonctionnels correctement et qu'aucune erreur est existante pouvant empêcher une action de bien se dérouler.

On peut retrouver par exemple le test d'une action de checkbox ou encore de changement de couleur en cas de clic.

Contrairement à Cypress, Jest s'effectue directement sur un éditeur de texte avec un projet importé permettant ensuite de tester le site action par action et en programmant des fonctions.

Repositories



ANALYSE

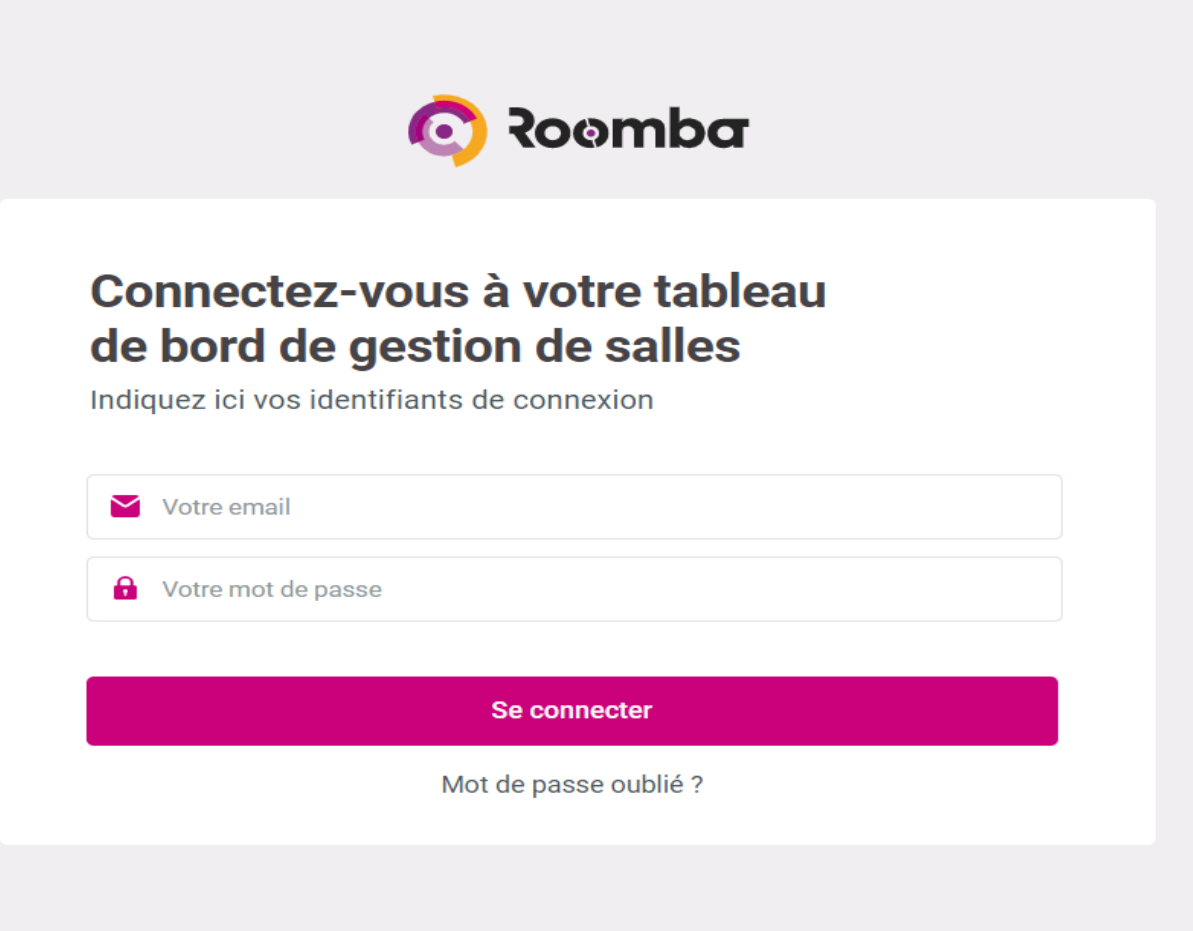
Ici l'objectif du stage reposait donc sur des tests à la fois end2end et unitaires afin de tester le site de Roombâ au niveau de sa fonctionnalité.

Comme dit précédemment, un site peut toujours faire face à une tentative de vol d'informations ou de bug contraignant son utilisation. Afin de protéger les utilisateurs et toutes leurs données sensibles, il est important de faire des tests de son site web pour que ce dernier puisse être fonctionnel.

Dans un premier temps, le site web de Roombâ n'était pas totalement testé dans son intégralité, me permettant ainsi de faire des tests pour trouver certaines erreurs ou bugs. Ces tests permettraient par la suite d'assurer la sûreté du site web au niveau client et au niveau administrateur.

Du côté administrateur, il fallait surtout tester la connexion et les erreurs pouvant survenir en cas de mauvaise connexion. Comme pour de nombreux sites, chaque administrateur avait son identifiant, c'est-à-dire son mail de travail (ici Roombâ) et son mot de passe qu'il avait choisis préalablement.

Comme on peut le voir sur l'image ci-dessous, l'administrateur doit remplir son mail et son mot de passe afin de se connecter.



The image shows a login page for 'Roombâ'. At the top, there is a logo consisting of a stylized 'c' in purple and orange, followed by the word 'Roombâ' in a bold, black, sans-serif font. Below the logo, the main heading reads 'Connectez-vous à votre tableau de bord de gestion de salles' in a bold, black font. Underneath this heading is a subtitle: 'Indiquez ici vos identifiants de connexion'. There are two input fields: the first is for the email, with a purple envelope icon and the placeholder text 'Votre email'; the second is for the password, with a purple lock icon and the placeholder text 'Votre mot de passe'. Below these fields is a large, solid purple button with the white text 'Se connecter'. At the bottom of the form area, there is a link that says 'Mot de passe oublié ?' in a smaller, grey font.

Par ailleurs, si l'administrateur avait oublié son mot de passe il pouvait faire une demande à partir de "Mot de passe oublié ?" qui lui donnait la possibilité de créer un mot de passe en lui envoyant un lien sur son adresse mail professionnel.
Sinon, l'administrateur avait accès aux parties du site qui lui étaient autorisées.

L'ensemble de toutes ces étapes étaient donc important à tester pour voir si aucune erreur ne survenaient durant l'intégralité des actions.
Grâce à Cypress il était alors possible de faire des simulation client ou administrateur pour voir si tout se déroulait correctement.

Dans un deuxième temps, j'ai utilisé le framework Jest pour effectuer des tests unitaires. Comme avec Cypress, l'objectif ici était également de tester le site web mais davantage au niveau de ses fonctionnalités.
En effet, le site web pouvait également présenter des failles au niveau des actions comme un clic sur une checkbox ou un radio button qu'il fallait tester.

L'objectif ici était donc de faire des tests unitaires qui simulaient des actions sur des buttons, qui vérifiaient que les checkbox étaient bien cochées lors d'un clic ou qu'un texte apparaissait bien lors d'un mouseOver.

Les différentes fonctions utilisées pour faire des tests unitaires permettaient ainsi à la fois de comprendre comment fonctionne chaque action et quel résultat devrait apparaître.
A travers l'éditeur de texte visual studio code, j'ai importé le projet test de Roomba puis j'ai commencé à effectuer mes tests avec les fonctions.

```
render(<Radios name="radios" defaultChecked</Radios>)</Radios>
const radios = document.querySelector("input");
expect(radios).toBeChecked();
```

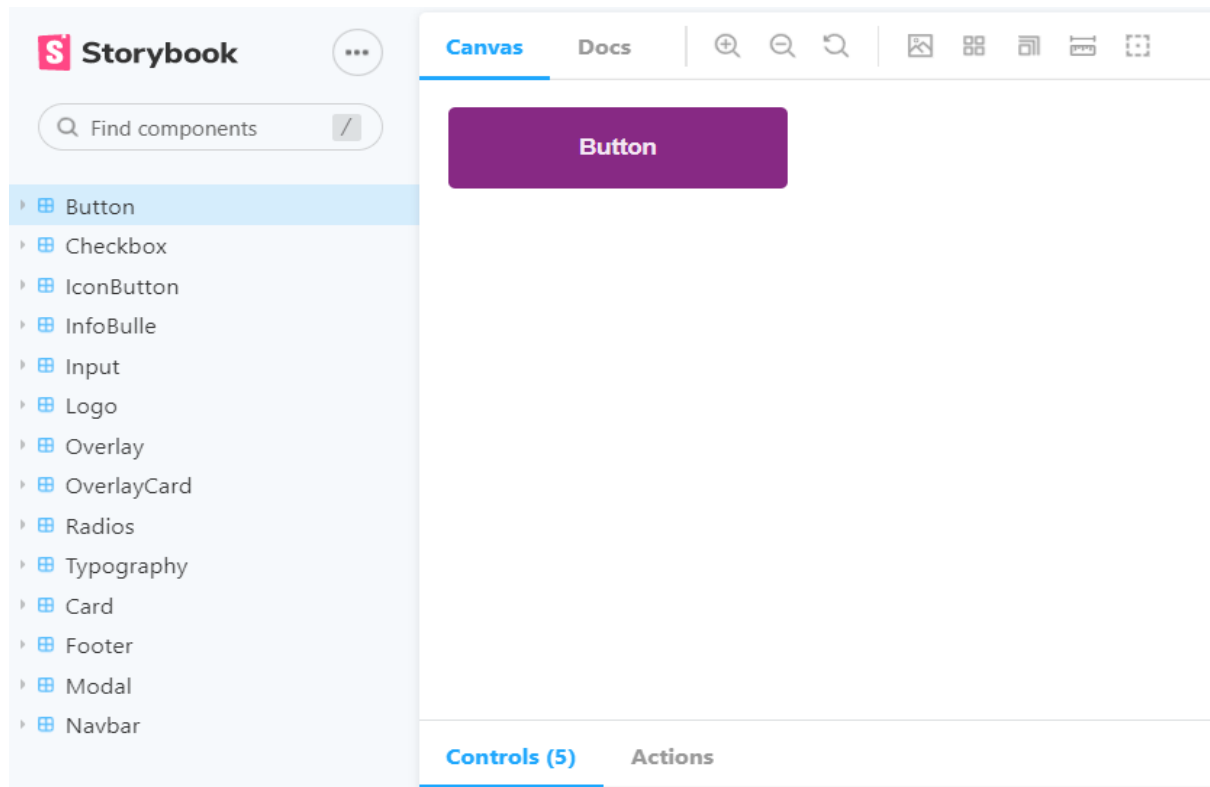
L'intégralité des tests fonctionnent sur une base qui est de créer une constante pour chaque action différente et de voir si l'action effectuée bien le résultat attendu quel que soit la valeur de la constante.

A l'intérieur de la fonction, on définit également une action comme un clic ou un passage de souris ou l'on attend donc un résultat différent comme un changement de couleur ou une ouverture d'une page après le clic.

```
const clickCallback = jest.fn();
render(<Button.Delete onClick={clickCallback}>Supprimer</Button.Delete>)
const button = document.querySelector("button");
fireEvent.click(button);
expect(clickCallback.mock.calls.length).toEqual(1);
```

Le projet se divise en plusieurs catégories comme la page des composants où l'on retrouve l'intégralité de tous les inputs sur lesquels on effectue des tests. Il y a aussi une partie pour les librairies qui permettent de simuler différentes actions dans les composants en appelant des méthodes.

Pour finir, l'attente du projet était d'effectuer un maximum de fonctions de tests pour l'ensemble de tous ces composants pouvant cacher des bugs. Je pouvais alors suivre la progression de mon travail sur un storybook répertoriant tous les input à tester, la progression de mes tests sur chacun d'eux(pour voir si les tests sont fonctionnels ou non) et ceux qu'ils me restent à tester.



All files

64.04% Statements 57/89 **30%** Branches 6/20 **41.46%** Functions 17/41 **64.04%** Lines 57/89

RÉALISATION

Cypress

Pour la réalisation de mon projet, j'ai dû installer Cypress afin d'effectuer la simulation et les tests end2end.

Durant les 2 premières semaines je me suis documenté sur le langage de Cypress et de son utilisation.

En effet, les tests end2end ont pour objectif de rendre l'application fonctionnelle à travers la vision client. Après les tests d'intégration qui permettent de tester principalement l'interface de l'application et les tests unitaires qui permettent de tester les fonctions d'une application ou d'un site web, les tests end2end sont centralisés sur le test du site web dans sa globalité.

Durant les débuts de mon stage je me suis donc initié au langage javascript à travers Cypress. Effectivement, Cypress est une nouvelle génération de front end testing permettant ainsi de tester son application web. Son arrivée sur le marché offre la possibilité aux grandes entreprises de pouvoir tester leur nouvelles application à travers une simulation client.

Par ailleurs, l'interface de Cypress nous donne la possibilité de contrôler totalement le test des applications grâce à différentes fonctions comme wait(requête).

De plus, cette application est d'autant plus intéressante car elle a la particularité de posséder le langage Javascript ce qui la rend plus accessible et plus maniable.

Plusieurs points du site web étaient importants à tester pour mener correctement la tâche.

Afin de comprendre le concept de Cypress j'ai donc d'abord effectué des tests d'analyse sur d'autres sites comme <https://example.cypress.io/>.

A travers ce site j'ai pu apprendre les différentes manières d'utiliser Cypress sur un site avec des actions requises comme un mail ou un mot de passe à fournir.

Ensuite, avec le peu de temps qui me restait avant de passer au framework Jest, j'ai effectué quelques tests sur la connexion à Roomba pour voir si un message d'erreur apparaissait bien en cas de mauvais mail ou mot de passe.

The screenshot shows the Cypress test runner interface on the left and the Roomba login page on the right. The test runner displays a test suite named 'test connexion' with a single test 'test 1'. The test body includes a 'visit' command to 'https://mairie.roomba.app/' and a series of 'get' and 'click' commands to interact with the login form. The Roomba login page features the Roomba logo, a title 'Connectez-vous à votre tableau de bord de gestion de salles', a subtitle 'Indiquez ici vos identifiants de connexion', and two input fields for 'email' and 'password'. A red error message is displayed below the inputs: 'Aucun compte ne possède cette adresse email ou alors les informations ne correspondent pas !'. A 'Se connecter' button is at the bottom, along with a link for 'Mot de passe oublié ?'.

Pour expliquer, on peut voir sur l'image ci-dessus qu'elle se divise en deux. Du côté gauche représente l'ensemble des toutes les actions écrites sur visual studio code détachées les unes des autres permettant de comprendre le détail de chaque étape.

En effet, comme l'image le montre ci-dessous, plusieurs actions sont effectuées sur la page. Elles sont ainsi structurées et classées par ordre d'appel avec un détail du résultat de l'action sur le côté droit de l'image(cit-dessus).

Comme on peut le voir, la simulation d'une connexion avec un mail ("email") et un mot de passe ("motdepasse") faux entraînent un message d'erreur et un refus de connexion.

```
describe('test connexion', function(){  
  beforeEach(() =>{  
    cy.visit('https://mairie.roomba.app/')  
  })  
  it('test 1',function(){  
    cy.get("input[type='text']").type('email')  
    cy.get('input[type="password"]').type('motdepasse')  
    cy.get("button").first().click()  
  })  
})
```

Comme on peut le voir sur la fonction du dessus, 'Describe' permet de créer un nouveau fichier Cypress qui rendra un rendu plus structuré.

On peut voir juste en dessous la fonction 'beforeEach' qui est une fonction qui s'appelle automatiquement en première à chaque nouveau test('it' sur Cypress).

Finalement dans les dernières lignes, on voit la syntaxe de cypress qui s'écrit en commençant toujours par cy. puis par l'action qu'on souhaite réaliser comme simulation.

Jest

Pour la réalisation de mon deuxième projet, j'ai dû importer le projet Roombâ afin d'effectuer la simulation et les tests unitaires.

Après ma réalisation avec Cypress sur la page de connexion à Roombâ, j'ai programmé des fonctions pour faire des tests unitaires.

J'ai d'abord commencé par approfondir mes connaissances en javascript qui était indispensable pour pouvoir programmer sous le framework jest.

Effectivement Cypress demande également des connaissances en javascript cependant moins poussé, contrairement à jest qui demande de plus grandes connaissances.

J'ai ensuite écrit des fonctions pour tester chaque composants différents pouvant créer un bug.

Pour expliquer, j'ai créé plusieurs fonctions pour chacun de ces composants afin de comprendre comment ils fonctionnaient en les analysant. Chaque composant agit de manière différente et entraîne un résultat, qui peut être différent de celui attendu.

Il faut ainsi structurer chacune des fonctions d'un composant en testant chacune de ses actions pouvant être réalisées et voir si le résultat est bien celui attendu.

```
test("devrait executer la fonction au click", () => {  
  const clickCallback = jest.fn();  
  render(<Checkbox onChange={clickCallback}></Checkbox>)  
  const checkbox = document.querySelector("input");  
  fireEvent.click(checkbox);  
  expect(clickCallback.mock.calls.length).toEqual(1);  
});
```

Par exemple, il est important pour une checkbox de vérifier si le clic va correctement fonctionner sur le site web. Ici on teste donc si la fonction va bien enregistrer un clic effectué lorsque l'on va le simuler à travers l'événement "fireEvent.click".

Logiquement par la suite on s'attends donc à un résultat positif qui nous retourne 1 puisque un clic a été effectué. Si la fonction nous retourne une erreur, alors cela veut dire que la fonction est fausse ou qu'un bug est peut être présent sur le composant.

Checkbox	<div></div>	100%	2/2
----------	-------------	------	-----

Si aucun bug ne survient et que le test passe, on peut voir dans le storybook que la progression est au maximum et que l'entièreté des fonctions sont correctes.

Dans l'exemple ci-dessus on voit bien que l'utilisateur fait un test en créant une constante en lui assignant une valeur `jest` qui correspond à un `fireEvent`.

Ensuite on crée la checkbox sur laquelle on assigne l'événement `onChange` avec la valeur de la constante.

On peut voir qu'on crée une deuxième constante qui renvoie tout simplement la checkbox du site web récupéré par un `"input"`. Puis on finit par ajouter un événement clic sur la checkbox avec comme résultat attendu `1`, qui correspond au nombre de clic effectué et donc attendu par la fonction `"toEqual"`.

CONCLUSION

Lors de mon premier jour de stage, Monsieur Foujols nous a expliqué le fonctionnement de son entreprise, la date de sa création et son objectif. Il existe plusieurs employés à l'intérieur de son entreprise avec des missions différentes pour chacun.

En tant que stagiaire, il m'a été confié comme tâche d'effectuer un maximum de test, que ce soit end2end ou unitaires, afin de réduire le plus de bug pouvant exister.

Grâce à une petite formation antérieure en javascript de sa part, l'utilisation de Cypress et Jest a pu être plus accessible et maniable.

Mes premières semaines à l'apprentissage et à la pratique de Cypress ont été plus faciles que celles avec Jest car l'application est plus facile à comprendre et demande moins de connaissances.

Malgré tout, j'ai bien apprécié l'utilisation du framework Jest qui m'a permis de bien comprendre son fonctionnement et par la même occasion le langage javascript.

Durant la totalité de mon stage, certaines étapes ont été plus dures que d'autres comme le passage à la pratique de Cypress qui n'était pas facile au début et difficile à comprendre. De plus, tout le programme de Jest, de l'apprentissage à la pratique, était également un passage difficile car la connaissance en javascript requise était importante.

Cependant, la pratique m'a permis de comprendre Jest et ainsi de proposer différents tests fonctionnels et intéressants.

L'entreprise Roombâ pourra donc profiter d'une partie de ces tests pour contrôler qu'aucun bug ne surviennent ainsi que pour terminer les tests manquants.