R4.02 | Qualité de développement (JMM)

Vous testerez votre application et en particulier vous montrerez que dans un développement itératif, vous avez effectué méthodiquement des tests qui garantissent la qualité et la non-régression à différentes étapes.

Vous rendrez un rapport de test (entre 5 et 10 pages, hors annexes, avec entête et nommage explicite SAE4-R402-2024-#NOMS#.pdf), présentant les tests d'au moins 2 itérations, en explicitant la couverture de test selon les trois approches étudiées au R4.02. Il sera intéressant de montrer que des tests ont permis de détecter des fautes et ont aidé à les corriger (ce qui a été vu en R2.03 peut être utilisé bien sûr et mis en avant dans vos rapports)

Deadline : 10 avril 2024 - 19h00
A rendre sur Madoc R4.02, un pdf avec un lien vers votre dépôt gitlab (dont m'aurez mis **maintener**) et des instructions pour qu'on puisse nous-même lancer vos tests.

Présentation de l’application Insight Galaxy

Introduction

Ce document est le compte-rendu de la partie Qualité de Développement de la SAE du 4e semestre. Il porte sur la création et la documentation des divers tests de notre projet.

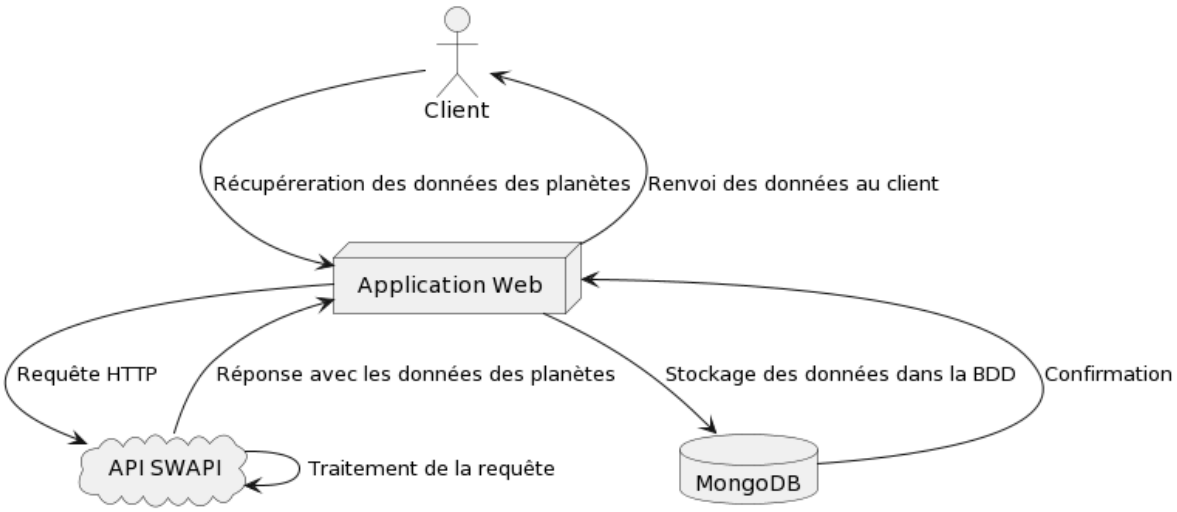
Notre SAE se porte sur un site web qui affiche les planètes de l’univers de star wars et qui permet de rajouter les planètes de notre choix.
En effet, en plus de contenir une soixantaine de planètes officielles de l’univers Star Wars, les utilisateurs ont la possibilité de créer de nouvelles planètes, de voter pour les planètes proposées par la communauté.

A chaque fin de semaine, le dimanche soir, la planète ajoutée par les utilisateurs la plus votée sera alors ajoutée à notre application.

Un vote unique par ordinateur est autorisé pour chaque planète proposée, car nous stockons l’adresse ip de chaque personnes votant pour une planète.

Bien évidemment à la fin de la semaine, chaque planète qui n’a pas été votée se voit supprimée de notre base de données. Les votes et adresses ip stockés se trouvent alors tous supprimés.

Structure de notre projet



Notre projet est subdivisé en 3 parties :

- Une API du monde de Star Wars, SWAPI, qui nous fournit les informations nécessaires à notre site web
- Le DAO de notre projet, qui sert à communiquer avec SWAPI, et qui gère une base de données visant à stocker les différents ajouts que notre site web permet (ajout de planète)
- Notre application Web React, qui gère l'affichage des informations de l'API, et qui permet tant à l'utilisateur de voter et de rajouter une planète, qu'à l'administrateur de supprimer une planète.

Développement itératif

Dans le cadre de notre projet, notre équipe composée de cinq membres a pu mettre en place un **développement itératif**, définissant les tâches du jour en fonction des besoins et des priorités du moment.

Même si nous n'avons pas strictement respecté les principes de "sprints" de la méthode agile, nous nous en sommes inspiré en n'hésitant pas à nous concentrer pendant une courte durée (quelques jours généralement) sur une tâche particulière, qu'elle soit liée à un intervalle de temps court avant un rendu (rendu de l'application Android) ou lié à une volonté d'éviter les "goulots d'étranglement" (mise en place de la structure du projet et de la séparation du site et du DAO pour permettre à de multiples personnes de travailler en même temps).

Bien sûr, nous avions des **livraisons incrémentales**, respectant l'esprit de la méthode agile, en modifiant et améliorant les différents éléments de notre SAE par pallier, ainsi, nous faisons attention à ne mettre sur le git que des éléments sans bug et fonctionnels, mais surtout, d'un point de vue plus large, nous nous focalisions, lors d'une journée, que sur l'objectif de la journée, dans le but d'implémenter des fonctionnalités entières et fonctionnelles, plutôt que divers petits objectifs non-respectés.

Par exemple, nous avons bien pris soin de passer beaucoup de temps à désigner notre application, avant de l'implémenter, permettant de faire un travail *par pallier*, et de rajouter des fonctionnalités possédant un design presque définitif.

De même, la création de l'application Android, ou d'un premier site web React fonctionnel ont fait l'objet d'itérations par eux-mêmes dans le cadre de "sprints".

Finalement, notre groupe respecte les principes de flexibilité, et de communication continue, en ayant une équipe polyvalente qui a travaillé sur tous les aspects de notre projet à un moment ou à un autre, et en n'hésitant pas à réassigner des membres de notre équipe à une tâche plutôt qu'une autre pour faire aux besoins actuelles.

C'est aussi une forme de ***révision continue*** de notre travail.

Tests

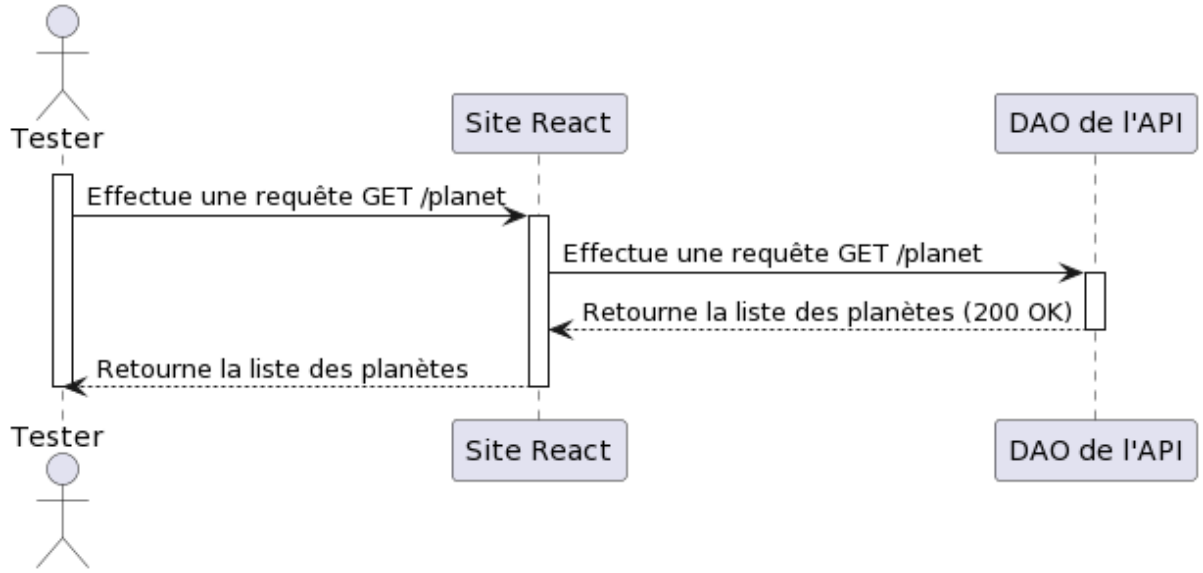
Dans le contexte de notre application, nous aborderons les différents aspects des tests appliqués à notre application, nous permettant de détecter et de corriger les éventuelles erreurs.

Nous avons opté pour une approche systématique combinant à la fois les méthodes **fonctionnelles** et **structurelles** dans nos tests.
À travers des **tests unitaires** et **des tests d'intégration**, la méthode fonctionnelle se concentre sur la vérification du comportement attendu de chaque fonctionnalité de notre application.
Les tests structurels se concentrent à examiner la structure et le flux de contrôle de notre projet. Cela permet aux utilisateurs de naviguer sur notre site avec fluidité et sans erreur.

Couverture de test

Nous visons une couverture complète pour garantir que toutes les fonctionnalités sont testées et que les éventuels bogues sont détectés.

Tests fonctionnels



Fonctionnement simplifié de l'affichage de la page /Search

```
it("GET /planet should get all planets", async function () {
    const response = await fetch('http://localhost:8090/planet', {
        method: 'GET',
        headers: {
            'Content-Type': 'application/json'
        },
```

```
        });

        expect(response.status).to.equal(200); // Vérifie le code de statut
de la réponse

        const responseBody = await response.json(); // Analyse le corps de la
réponse comme JSON

        expect(responseBody).to.be.an('array'); // Vérifie que la réponse est
un tableau

        responseBody.forEach(planet => {
            expect(planet).to.have.property('name');
        });
    });
```

Ce test vérifie que la requête GET /planet renvoie correctement toutes les planètes. Il s'assure que le code de statut de la réponse est 200, que le corps de la réponse est un tableau JSON et que chaque élément du tableau a une propriété 'name'.

Ces tests fonctionnels sont cruciaux pour garantir que l'application répond aux besoins des utilisateurs et fonctionne comme prévu.

Dans notre rapport de test, nous fournirons également d'autres exemples de tests fonctionnels réalisés sur différentes fonctionnalités de l'application.

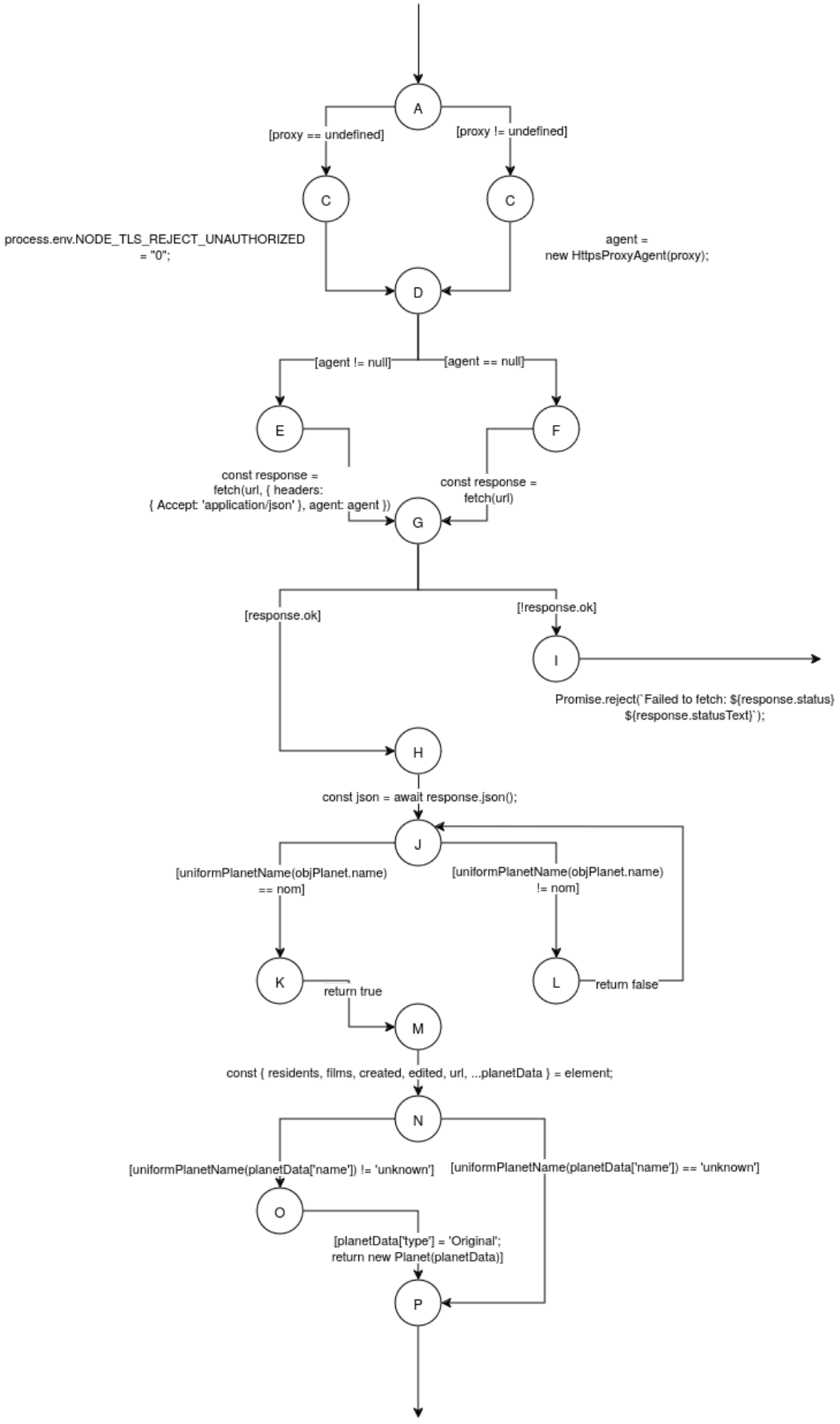
Tests structurels

Graphe de flot de contrôle de la fonction findPlanetByNomSWAPI, qui renvoie les planètes de l'API :

La fonction findPlanetByNomSWAPI est chargée de récupérer toutes les planètes de l'API SWAPI, les planètes rajoutées par les utilisateurs n'y sont donc pas.

En effet, nous avons décidé, pour des raisons pratiques, de diviser notre fonction de récupération de planètes en deux plus petites fonctions : une pour les planètes de l'API et l'autre pour les planètes de notre BDD (Base de données).

Il est à noter que nous possédons aussi une fonction findPlanet qui fera les appels à ces deux fonctions.



Graphe de flot de contrôle de la fonction findPlanetByNomSWAPI,

chargé de trouver les planètes rajoutées par les utilisateurs

Liste finie de chemins potentiels :

- ch1 = (A, B, D, E, G, H, J, K, M, N, O, P) (pas de proxy, requête correcte, nom de la planète vérifié correcte, nom de la planète en minuscule incorrecte)
- ch2 = (A, C, D, F, G, H, J, K, M, N, O, P) (proxy, requête correcte, nom de la planète vérifié correcte, nom de la planète en minuscule incorrecte)
- ch3 = (A, B, D, E, G, I) (pas de proxy, requête incorrecte)
- ch4 = (A, C, D, F, G, I) (proxy, requête incorrecte)
- ch5 = (A, B, D, E, G, H, J, L, J, K, M, N, O, P) (pas de proxy, requête correcte, nom de la 1ere planète vérifié incorrecte, nom de la planète en minuscule incorrecte)
- ch6 = (A, B, D, E, G, H, J, L, J, K, M, N, P) (pas de proxy, requête correcte, nom de la 1ere planète vérifié incorrecte, nom de la planète en minuscule correcte)
- ch7 = (A, B, D, E, G, H, J, K, M, N, P) (pas de proxy, requête correcte, nom de la planète vérifié correcte, nom de la planète en minuscule correcte)
- ch8 = (A, B, D, F, G, H, J, K, M, N, P) (pas de proxy, mais agent de proxy non vide, requête correcte, nom de la planète vérifié correcte, nom de la planète en minuscule correcte)
- ch9 = (A, C, D, E, G, H, J, K, M, N, O, P) (proxy, mais agent de proxy vide, requête correcte, nom de la planète vérifié correcte, nom de la planète en minuscule i,correcte)
- ch10 = (A, C, D, F, G, H, J, K, M, N, P) (proxy, requête correcte, nom de la planète vérifié correcte, nom de la planète en minuscule correcte)

Expression de l'ensemble des chemins potentiels :

- Exp = A * ((B, D, E) + (C, D, F)) * G * (I + (H, J, (L, J)*, K, M, N, (O,P) + (P))

DT sensibilisant les chemins:

- La DT1 = { Ne pas avoir de proxy, Être connecté à Internet, nom = "Tatooine" } sensibilise le ch1 qui est exécutable
- La DT2 = { Être connecté à Internet, nom = "TatOoIn e ",} sensibilise le ch2 qui est exécutable
- La DT3 = { Ne pas avoir de proxy, Ne pas être connecté à Internet} sensibilise le ch3 qui est exécutable
- La DT4 = { Avoir un proxy, Ne pas être connecté à Internet} sensibilise le ch4 qui est exécutable
- La DT5 = { Ne pas avoir de proxy, Être connecté à Internet, nom = "PlottuIV"} sensibilise le ch5 qui est exécutable
- La DT6 = { Ne pas avoir de proxy, Être connecté à Internet, nom = "PlottU I V "} sensibilise le ch6 qui est exécutable
- La DT7 = { Avoir un proxy, Être connecté à Internet, nom = "Ta tOoIne "} sensibilise le ch7 qui est exécutable
- La DT10 = { Être connecté à Internet, nom = "Tatooine",} sensibilise le ch10 qui est exécutable

- CT1(DT1, Rajout de la planète à la BDD, et le type de planète a été changé)

- CT2(DT2, Rajout de la planète à la BDD (mais en ayant utilisé un proxy), et le type de planète a été changé)
- CT3(DT3, Rejette la promesse et affiche un message d'erreur "Failed to fetch: .[...]")
- CT4(DT4, Rejette la promesse et affiche un message d'erreur "Failed to fetch: .[...]" (avec proxy))
- CT5(DT5, Rajout de la planète à la BDD, sans proxy, la première planète comparé à notre planète est différente, et le type de planète a été changé)
- CT6(DT6, Rajout de la planète à la BDD, sans proxy, et la première planète comparé à notre planète est différente)
- CT7(DT7, Rajout de la planète à la BDD)
- CT10(DT10, Rajout de la planète à la BDD (mais en ayant utilisé un proxy))

- Aucune DT ne peut sensibiliser le ch8 et le ch9, qui ne sont pas exécutable car les deux premières conditions de notre programme sont liés, et il est impossible de ne pas avoir de proxy mais d'avoir un agent de proxy (qui est sensé utiliser le proxy pour accéder à internet), et inversement.

Nous pouvons donc voir que notre programme présente au moins un défaut : La création de l'agent du proxy et le fetch qui y est lié pourraient être fait grâce à **un seul élément conditionnel**.

C'est une démonstration que *des tests ont permis de détecter (au moins) une faute et ont aidé à la corriger.*

Précision : la dernière condition vérifie que le nom de la planète est correct et ne possède pas d'espaces ou de majuscules en trop.

Exemples de tests effectués sur findPlanetByNomSWAPI :

```
   it('findPlanetByNomSWAPI devrait retourner une liste de planètes
correspondant au nom fourni avec les attributs corrects', async () => {
       const planet = await planeteDao.findPlanetByNomSWAPI('Tatooine');
       expect(planet).to.be.an('array').with.length.greaterThan(0);
       expect(planet.every(info => info instanceof Planet)).to.be.true;

       // Vérifier les attributs de la première planète retournée
       const firstPlanet = planet[0];
       expect(firstPlanet.name).to.equal('Tatooine');
       expect(firstPlanet.description).to.equal(undefined);
       expect(firstPlanet.rotation_period).to.equal('23');
       expect(firstPlanet.orbital_period).to.equal('304');
       expect(firstPlanet.diameter).to.equal('10465');
       expect(firstPlanet.climate).to.equal('arid');
       expect(firstPlanet.gravity).to.equal('1 standard');
       expect(firstPlanet.terrain).to.equal('desert');
       expect(firstPlanet.surface_water).to.equal('1');
       expect(firstPlanet.population).to.equal('200000');
```

```
        expect(firstPlanet.type).to.equal('Original');
    });
```

Ceci est un exemple d'un test que nous avons dû faire pour respecter les 7 Cas de Test à effectuer.

Ainsi, ce test correspond **au CT10**, c.a.d. le cas ou la fonction fonctionne parfaitement, mais en utilisant un proxy (ce test a été fait à l'IUT, ou il y a un proxy)

Ce test vérifie donc si la fonction retourne une liste de planètes correspondant au nom fourni avec les attributs corrects.

- Vérifiez si la valeur retournée est un tableau non vide.
- Vérifie si chaque élément du tableau est une instance de la classe `Planet`.
- Vérifie les attributs de la première planète retournée, tels que le nom, la description, la période de rotation, la période orbitale, le diamètre …

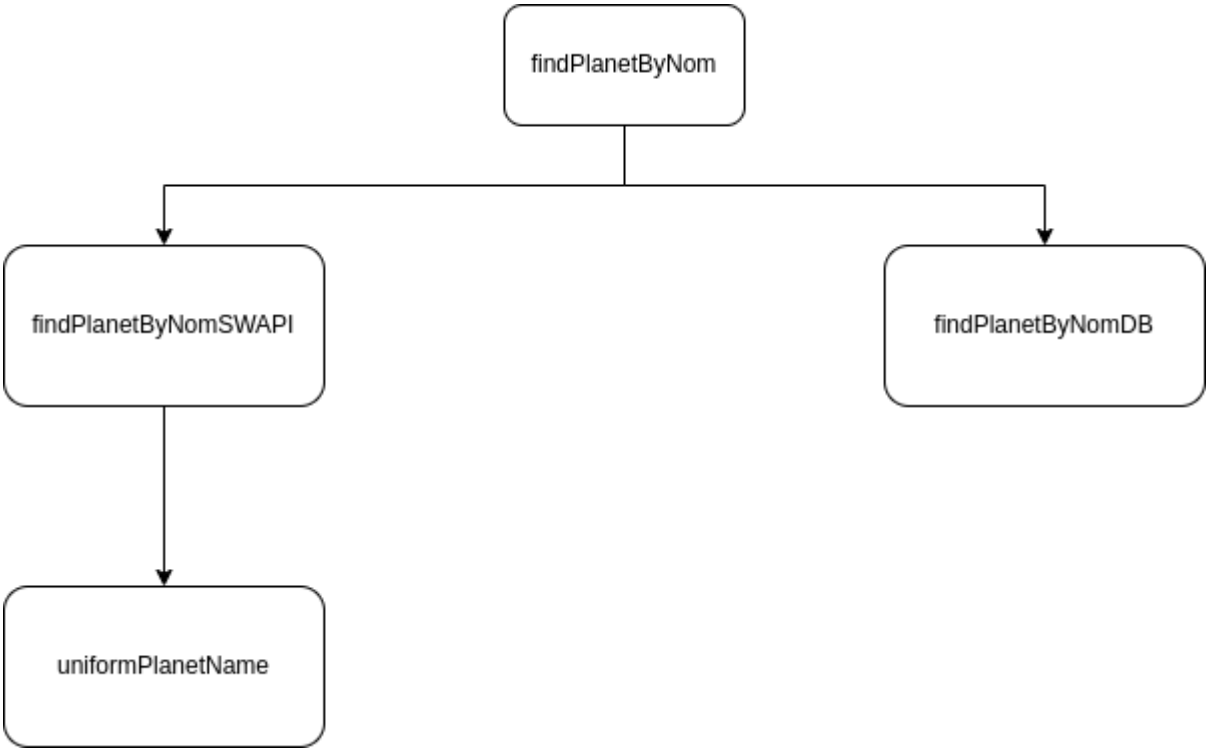
```
    it('findPlanetByNomSWAPI devrait retourner une liste vide si aucun nom ne
correspond', async () => {
        const planets = await planeteDao.findPlanetByNomSWAPI('notfound');
        expect(planets).to.be.an('array').that.is.empty;
    });
```

Ce test vérifie si la fonction retourne bel et bien une fonction vide.

Ainsi, nous pourrions comparer ce test au ch6, à la seule différence que dans le chemin 6, seulement la 1ere planète de l'API n'est pas égale à ce que l'on cherche, alors que dans notre cas, c'est le cas pour *toutes les planètes*.

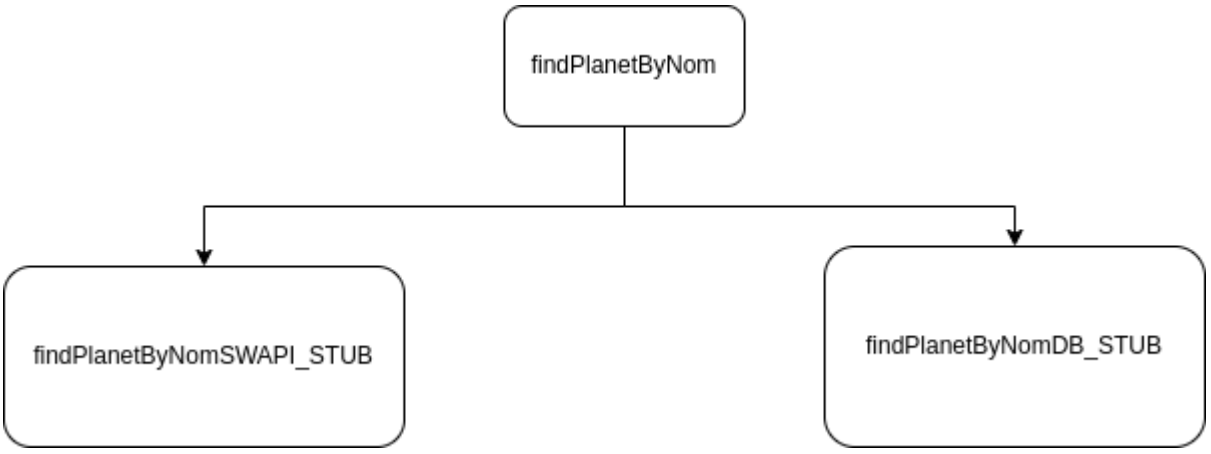
Tests d’Intégration

Nous avons également fait plusieurs tests d’intégration, visant à vérifier le bon fonctionnement des différentes parties de notre projet lorsqu'elles sont combinées, afin de garantir interaction et de la cohérence de l'ensemble de l'application, ainsi, nous avons décidé de faire plusieurs diagrammes représentant les différentes étapes de test de findPlanetByNom :

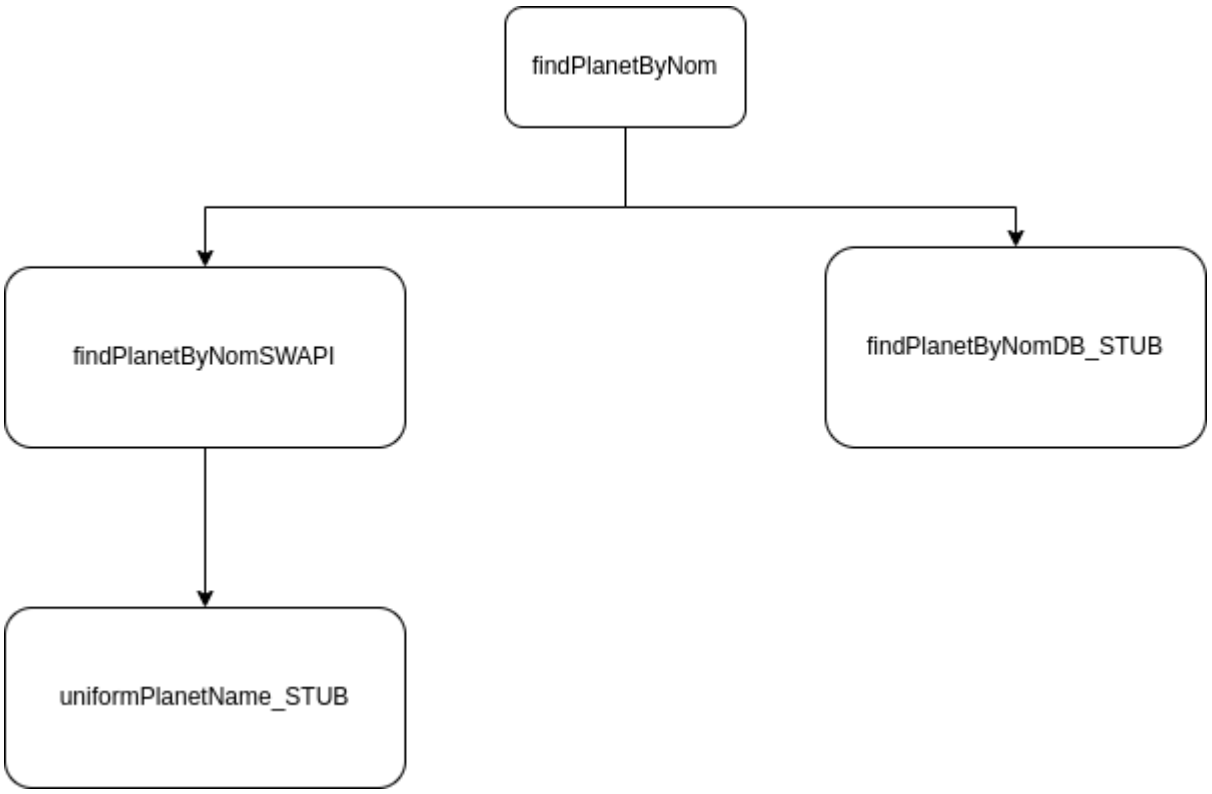


Dans cet exemple, nous avons utilisé une approche *top-down* en *largeur*.

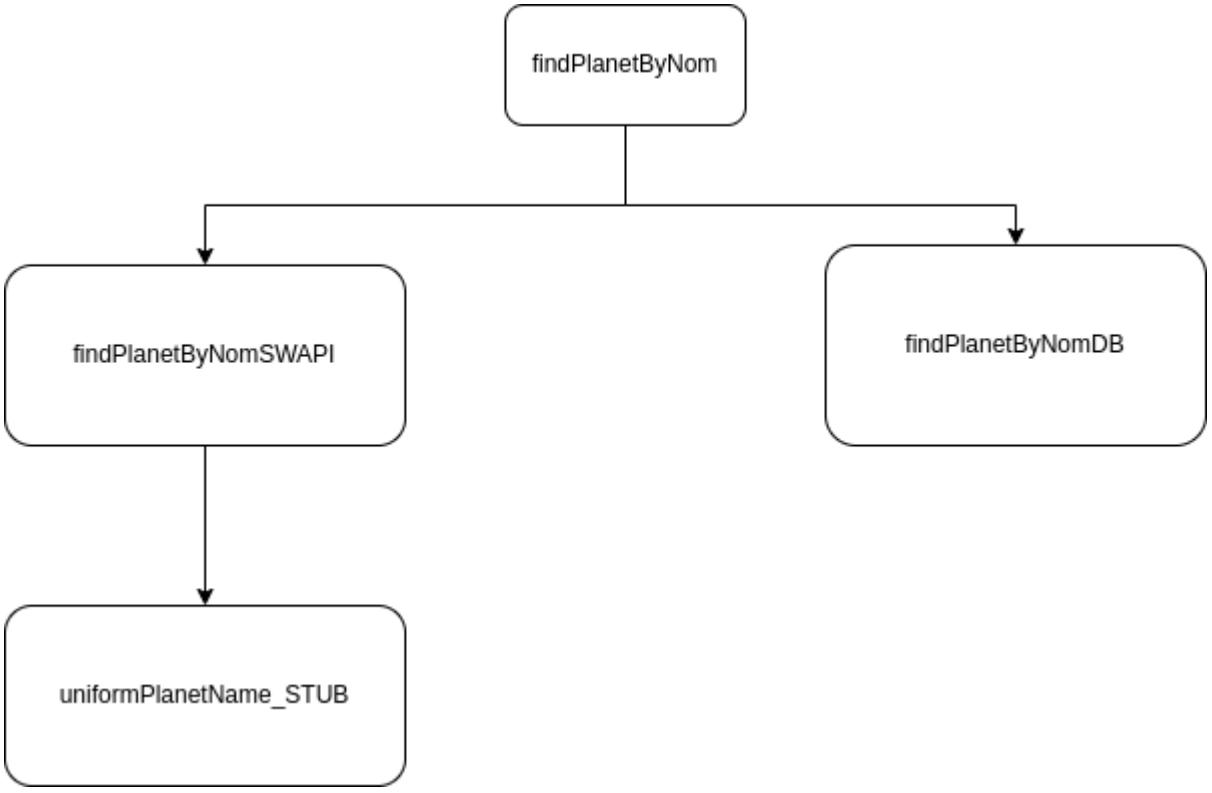
Plus précisément, nous avons testé dans un premier temps avec un stub de **findPlanetByNomSWAPI** et **findPlanetByNomDB**.



Puis nous testons avec un stub de **uniformPlanetName** et **findPlanetByNomDB**.

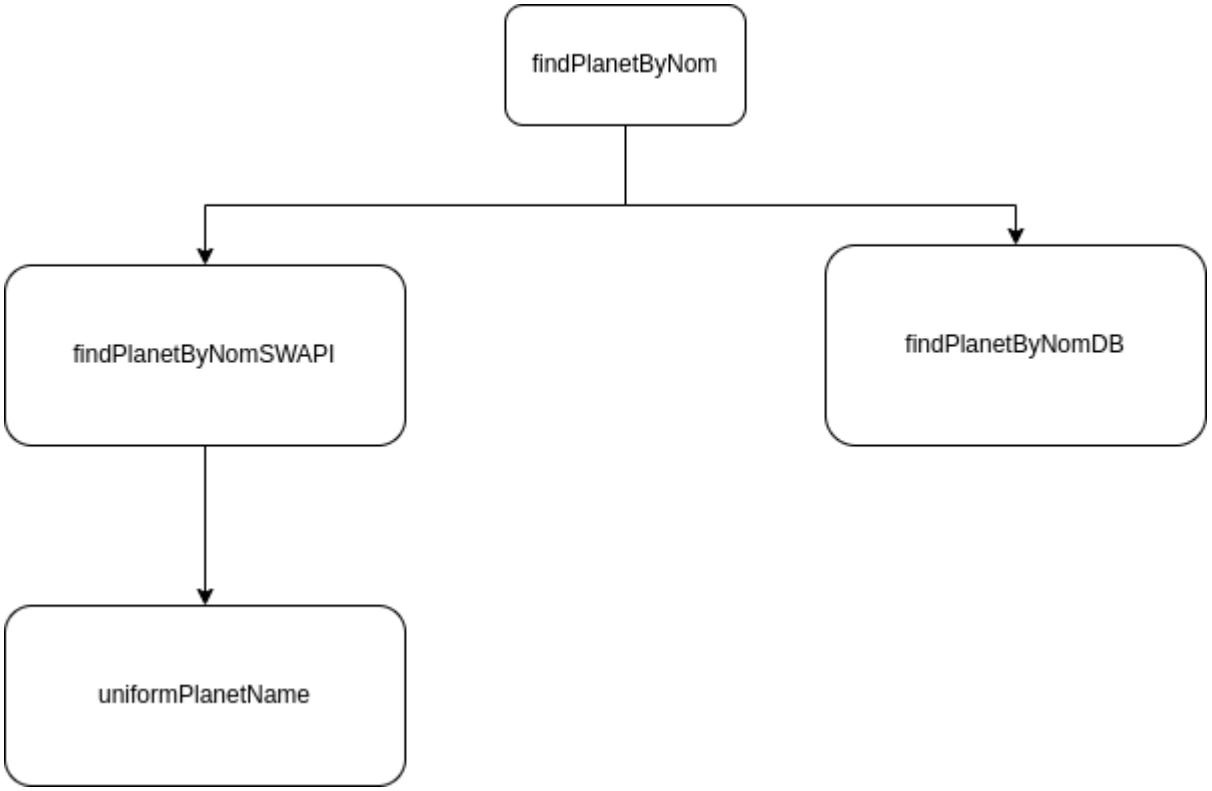


… puis nous continuons avec un stub de **uniformPlanetName**.



Enfin, nous avons fini par tester la fonction originale, *sans* stub.

Le test de la fonction est fini.



Conclusion

Dans le cadre de notre projet Insight Galaxy, nous avons mis en place une approche itérative et systématique, combinant des tests fonctionnels et structurels pour garantir la qualité et la non-régression à chaque étape du développement.

Notre couverture de test complète a permis de détecter et de corriger efficacement les défauts, comme en témoigne notre analyse de mutation.

En utilisant des méthodes agiles, nous avons assuré une communication continue et une révision proactive du travail de l'équipe.

En résumé, notre approche rigoureuse des tests et notre développement itératif ont permis d'assurer la qualité et la fiabilité de notre application Insight Galaxy, offrant ainsi une expérience utilisateur optimale.