

Table des matières

[1. Analyse préliminaire 2](#_Toc185575451)

[1.1 Introduction 2](#_Toc185575452)

[1.2 Objectifs 2](#_Toc185575453)

[1.3 Gestion de projet 2](#_Toc185575454)

[1.4 Planification initiale 2](#_Toc185575455)

[2. Test 3](#_Toc185575456)

[2.1 Politique de Test 3](#_Toc185575457)

[2.2 Stratégie de test 4](#_Toc185575458)

[2.3 Plan de test 5](#_Toc185575459)

# Analyse préliminaire

## Introduction

Le projet « P\_DevOps\_Test » est réalisé dans le cadre du module 450, qui a comme thème les processus DevOps et en spcilité la mise en place d’un environnement de test automatisé. Le but est de créer un environnement de tests ainsi que des tests unitaires pour une application de To Do List, tout en utilisant les outils et méthodes vue en classe. Le projet utilise GitHub Project pour la gestion des taches et un journal de travail afin de suivre le déroulement du projet. Le chef de projet est Aurélie Curchod notre enseignante.

## Objectifs

Les objectifs du projet « P\_DevOps\_Test » sont les suivants :

**Mettre en place un environnement de Test automatisé** :

1. Mise en place de test unitaires.
2. Mise en place de test End to End.

**Mettre en place une politique et une stratégie de test incluant :**

1. Les types de tests effectués.
2. Les outils utilisés.
3. L’environnement de test utilisé.
4. Les critères de succès des tests.

**Mettre en place un plan de test pour les fonctionnalités suivantes :**

1. Création du profil.
2. Authentification.
3. Gestion du profil.
4. Ajout d’un TODO
5. Gestion des TODO

**Mise en place des rapports de test associés :**

## Gestion de projet

Ce projet est effectué en groupe de deux. Pour cela, nous utilisons l'outil GitHub Project afin de lister les tâches et attribuer les tâches à réaliser. Nous utilisons également GitHub Desktop ainsi que des commits pour que chacun puisse voir une version du projet à jour.

Un journal personnel est aussi utilisé sous forme de fichier CSV. Ce journal permet de suivre l'avancement du projet et de recenser les problèmes rencontrés. Ainsi, si une des deux personnes est absente, il est toujours possible de suivre l'avancement du projet. Le but est qu'aucun événement ne ralentisse l'avancement et le bon déroulement du projet.

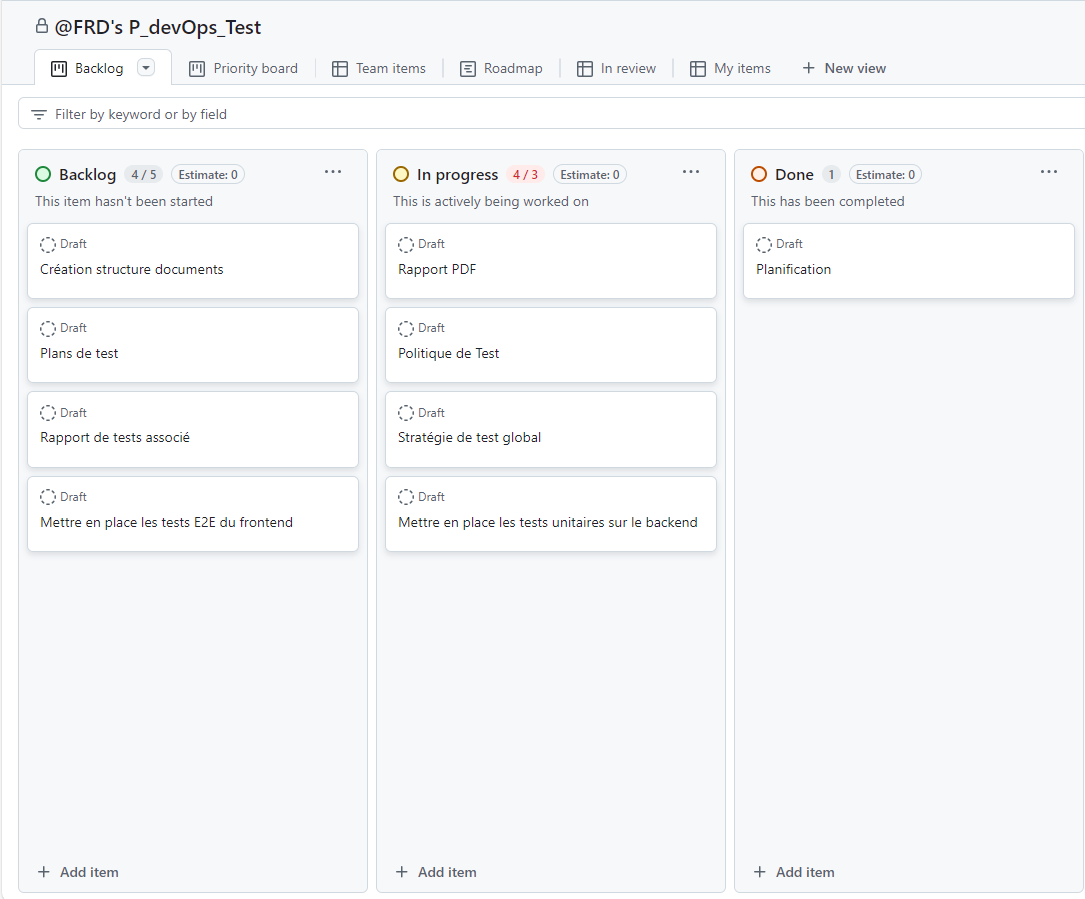
Le nommage des commits est également un point important, car il permet au professeur d'évaluer le projet à 80 % et d'obtenir un retour.

## Planification initiale

Dans ce projet, nous partons avec une toute petite expérience de test dû à la création de test unitaire en C# Cependant c’est bien la première fois que nous allons effectuer ce type de test en Web de plus il est important de noter que mise à part un peu de théorie nous n’avons jamais utilisé des outils tel que «Jest » « supertest » et « cypress »  . Nous nous sommes donc concentrés sur ce qui nous était spécifiquement demandé et avons cherché à comprendre le CDC afin de pouvoir au mieux planifier les tâches principales.

Nous avons réfléchi à ce qui était le plus important à faire au début, comme la mise en place de l’environnement, ainsi qu’aux différentes étapes nécessaires pour avancer efficacement, petit à petit, dans le projet.

Nous n’allons pas directement commencer par la mise en place des tests dans le code, mais d'abord par l’écriture de la politique, la stratégie et le plan de test :



Planification

# Test

## Politique de Test

***Introduction***

Cette politique de test définit les approches, les outils et les méthodologies pour assurer la qualité de l'application Todo, qui comprend un frontend Vue.js, un backend Node.js/Express, et une base de données MongoDB.

***Objectifs***

* Valider le fonctionnement des fonctionnalité prioritaire (création de profil, authentification, gestion de profil, gestion des tâches).
* Identifier les potentiels problèmes des fonctionnalités déjà codé
* Vérifier le bon fonctionnement complet du code via des test unitaire pour le backend.

## Stratégie de test

***Type de tests***

* Tests unitaires pour le BackEnd.
* Test End to End pour le FrontEnd.

***Tests unitaires***

Outils :

* Jest.
* Supertest

Environnements :

* Local
* Intégration

***Test End to End***

Outils :

* Cypress

Environnements :

* Production

***Environnement***

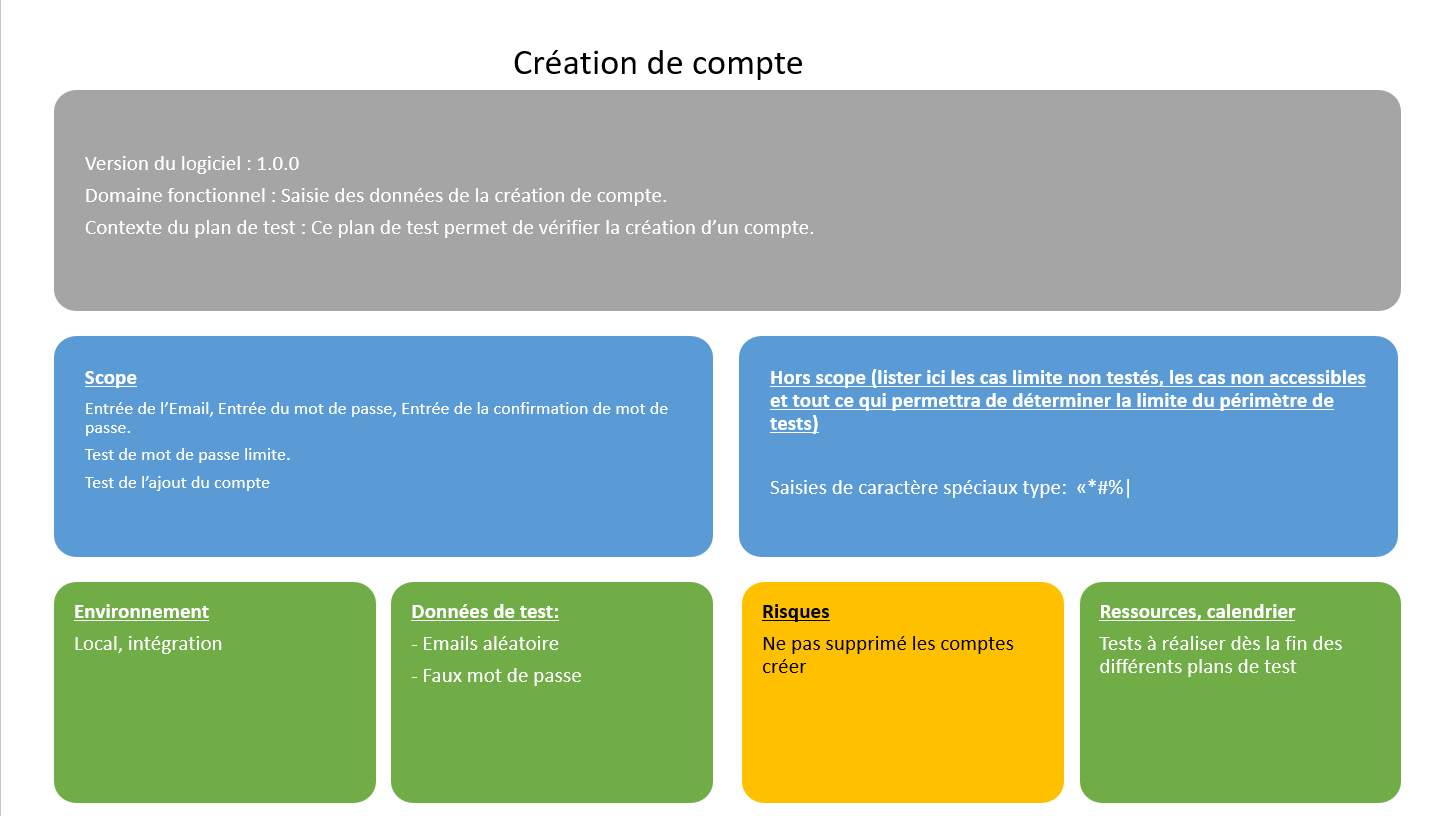
* Environnement de développement local.
* Environnement intégration (CI/CD, GithubAction).
* Environnement production.

***Critères de succès***

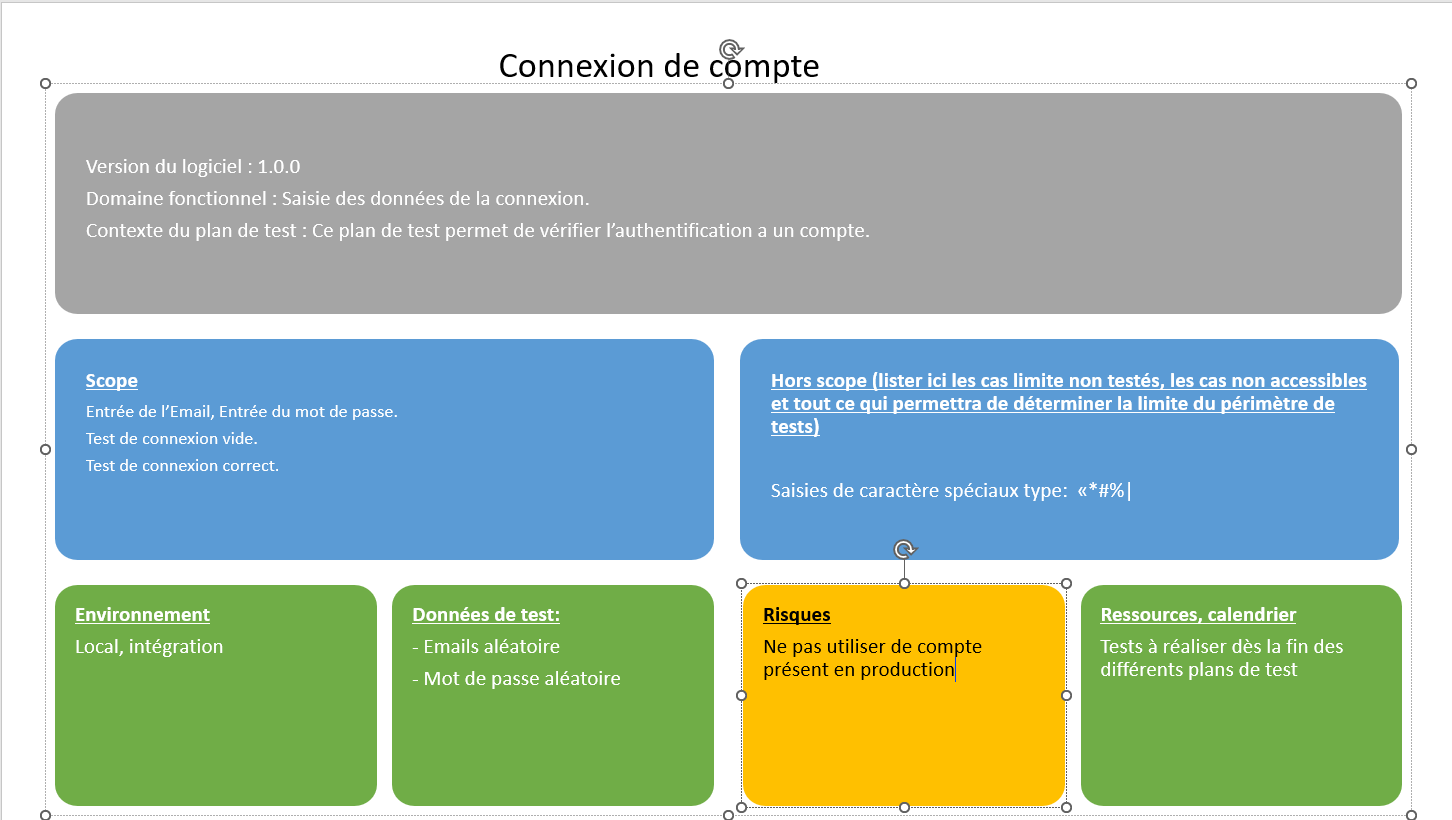
* 100% des test unitaire réussis

## Plan de test

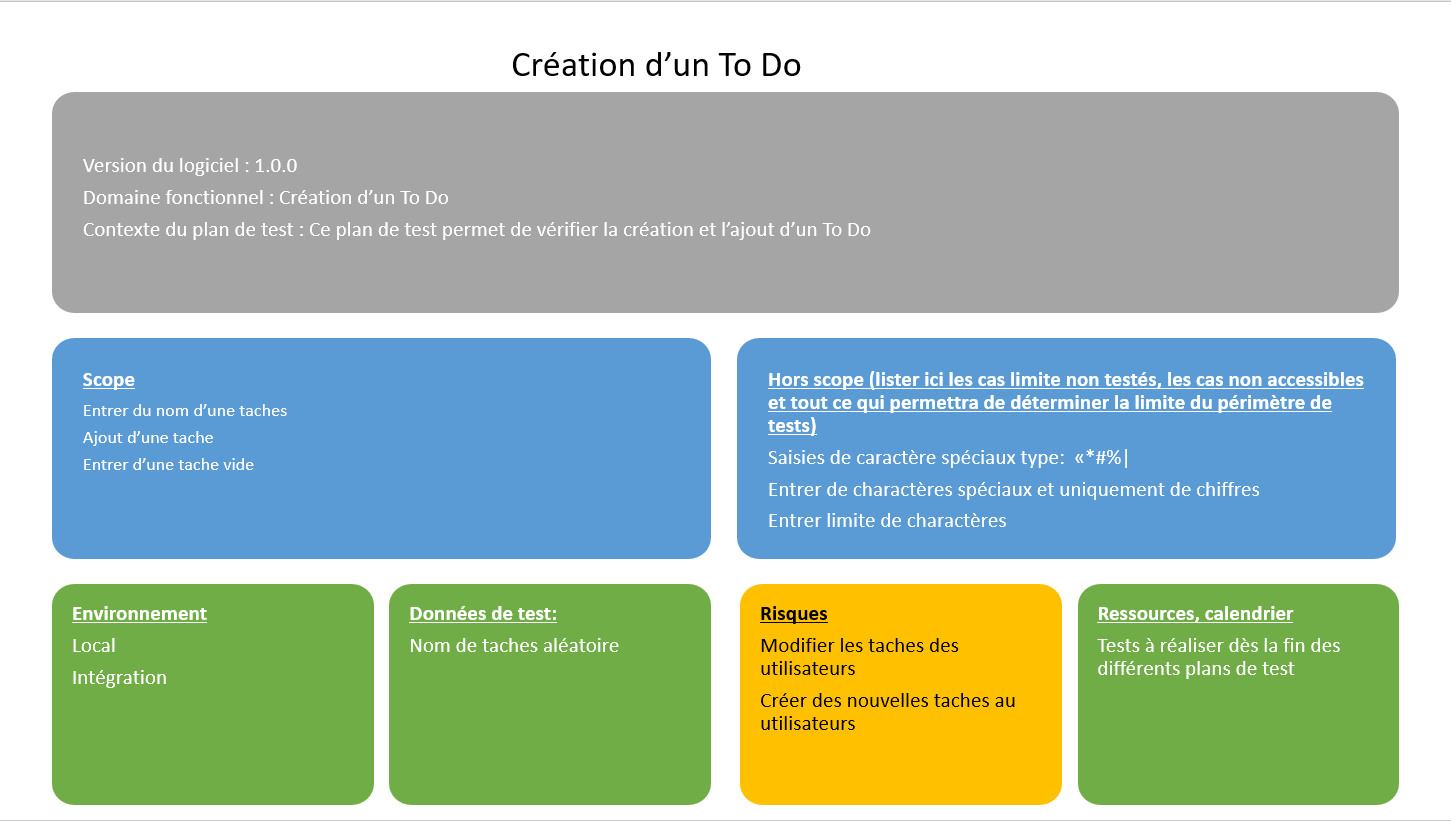
***Création de compte***



***Connexion au compte***

******

***Création Des Todo***

******

***Gestion Des Todo***

******

## Test Backend

### Authentification

Ces tests ont pour but de vérifier le système d’authentification de l’application pour d’assurer que le fonctionnement et les sorties de la fonctionnalité soit correct.

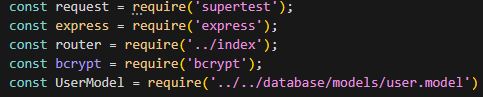
Pour se faire nous avons testé deux cas distincts le premier étant l’authentification avec des données valide, et le deuxième avec des données invalide.

***Module Requis***

Voici la liste des différents module/importation requise pour mettre en place les tests :

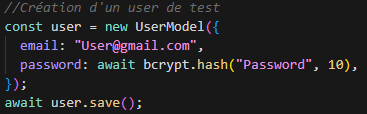
* SuperTest
* Express
* Le fichier index.js du dossier routes
* Bcrypt
* Le fichier user.model.sj du dossier models

Voici comment cela se mets en place dans le code.

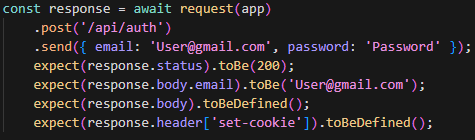


***Authentification avec données valide***

La première étape consiste à créer un nouveau model d’utilisateur avec la variable « UserModel ».

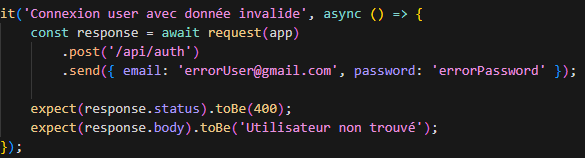


La seconde envoie une demande d’authentification avec les informations du model venant d’être créé et vérifie les réponses du server.



***Authentification avec données invalides***

La seule étape de ce test est une tentative de connexion sans avoir créer de model au préalable.



***Résultats Authentification avec données valide***

|  |  |
| --- | --- |
| **Résultat attendu** | **Résultat Obtenu** |
| Statut du server 200 | Statut du server 200 |
| Email dans le body de la réponse du server | Email dans le body de la réponse du server |
| Présence d’un body dans la réponse du server | Présence d’un body dans la réponse du server |
| Présence de cookies dans le header de la réponse du server | Présence de cookies dans le header de la réponse du server |

***Résultats Authentification avec données invalide***

|  |  |
| --- | --- |
| **Résultat attendu** | **Résultat Obtenu** |
| Statut du server 400 | Statut du server 400 |
| Message « Utilisateur non trouvé » dans le body de la réponse du server | Message « Utilisateur non trouvé » dans le body de la réponse du server |

### Test Login

Ces tests ont pour but de vérifié le système de création de compte de l’application, ils passeront en revue la création d’un utilisateur avec des donnée valide ainsi que la création d’un utilisateur sans Email.

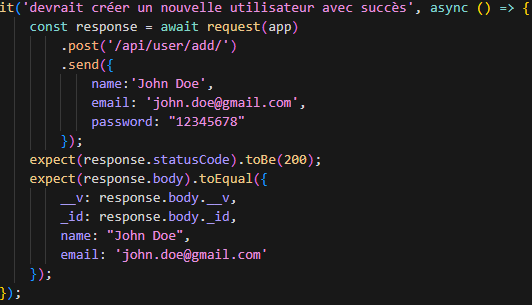
***Modules Requis***

Voici la liste des module/importation requise pour ces tests :

* SuperTest
* Express
* Le fichier index.js du dossier routes

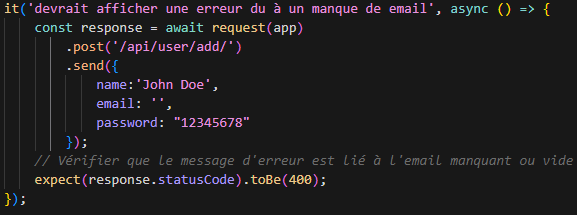
***Création d’un compte avec des données valides***

La seule étape de ce consiste à envoyer une requête avec le nom, l’Email et le mot de passe dans le corps de la requête afin de créer un nouvel utilisateur.



***Création d’un compte avec des données invalides***

Ce test tente de créer un nouvel utilisateur mais sans fournir d’email afin de provoquer une erreur type 400.



***Résultat Création de compte avec données valides***

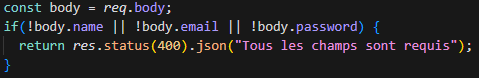
|  |  |
| --- | --- |
| **Résultat attendu** | **Résultat Obtenu** |
| Statut de la réponse 200 | Statut de la réponse 200 |
| Renvoie le nom et l’email de l’utilisateur créé dans le body de la réponse | Renvoie le nom et l’email de l’utilisateur créé dans le body de la réponse |

***Résultat Création de compte avec données invalides***

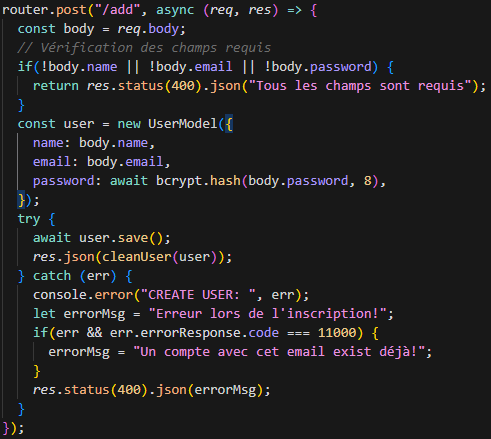
|  |  |
| --- | --- |
| **Résultat attendu** | **Résultat Obtenu** |
| Statut de la réponse 400 | Renvoie d’une erreur de base de données car le code ne gère pas le manque d’email lors de la création d’un utilisateur. |

***Solution proposée***

La solution serait de vérifier si tous les champs demandés sont rentrés avant de créer l’utilisateur comme ceci.



Une fois ce code ajouter à la route entière.



Le code renvoie bien une erreur 400 si un des champs est manquant.

### Gestion de l’utilisateur

Ces tests vérifient le bon fonctionnement des fonctionnalités lié à la gestion d’un utilisateur de l’application, la gestion des utilisateurs prend en compte :

* Suppression d’un utilisateur
* Mise à jour du nom de l’utilisateur
* Mise à jour de l’adresse de l’utilisateur
* Mise à jour du NPA
* Mise à jour du lieu

Les tests vérifieront donc tous ces points ainsi que la mise à jour de plusieurs champs en même temps et la tentative de modification sans token dans les cookies de la requête.

***Module requis***

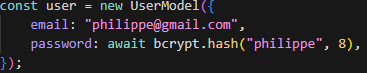
Les modules suivants seront requis afin de mettre en place les tests de gestion de l’utilisateur :

* SuperTest
* Express
* Index.js
* User.model
* JsonWebToken
* La clef privée du projet
* Cookie-parser
* Bcrypt

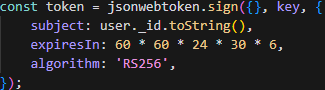
***Suppression d’un utilisateur***

Ce test vérifie la suppression d’un utilisateur, pour se faire il est divisé en trois étapes :

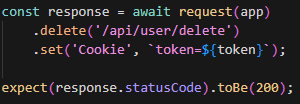
* Création d’un utilisateur avec le UserModel



* Signature d’un token grâce à la clef privée

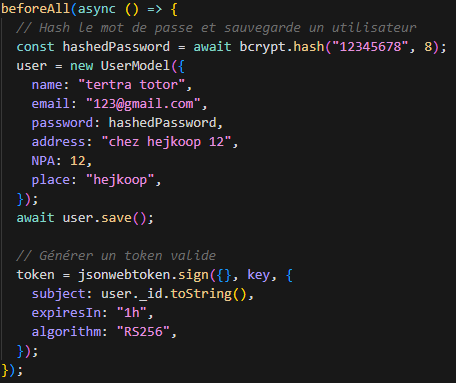


* Envoie de la requête de suppression et vérification de la réponse du server



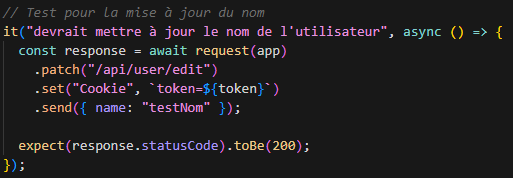
***Prérequis Modification Utilisateur***

Avant de pouvoir mettre en place les différents tests il est important de s’assurer qu’un utilisateur est créé, pour se faire il est nécessaire d’utiliser la fonction « beforeAll » qui sera exécuté avant tous les tests de modification.



***Mise à jour du nom d’utilisateur***

Ce test consiste en une simple requête vers la route « Edit » avec en corps la valeur du nouveau nom de l’utilisateur.



***Mise à jour de l’adresse de l’utilisateur***

Ce test consiste en une simple requête vers la route « Edit » avec en corps la valeur de la nouvelle adresse de l’utilisateur.

