Plateforme MedHead

Application POC pour le traitement des recommandations de lits dans les situations d'intervention d'urgence

Plan de test

Version 1.0

Auteur : Hervé Prevost - Architecte Logiciel - MedHead

Date : 14/09/2023

Version : 1.0

SOMMAIRE

OBJET DE CE DOCUMENT

Ce document a pour but de présenter le plan de test pour l'application POC dédiée au traitement des recommandations de lits disponibles en situation d'urgence.

RAPPEL DES PRINCIPES DE TEST

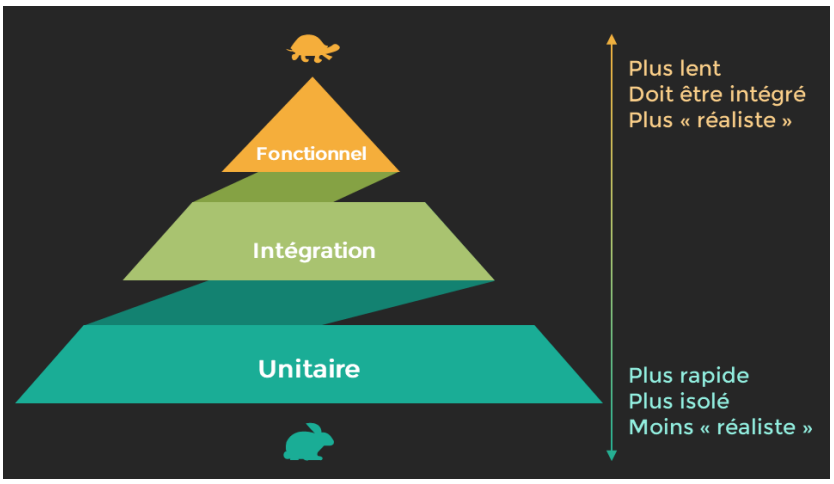
Test d'un composant logiciel

Les procédures de test sur la partie développement suivront le modèle du TDD (Test Driven Development) et des recommandations DevOps CI/CD. Ils seront automatisés et constitués des 3 étapes standards du modèle :

- **Les tests unitaires** ont un périmètre restreint et vérifient généralement le comportement de méthodes ou de fonctions individuelles.
- **Les tests d'intégration** permettent de s'assurer de la bonne cohabitation des composants. Cela peut impliquer plusieurs classes, ainsi que de tester l'intégration avec d'autres services.
- **Les tests d'acceptation** sont similaires aux tests d'intégration, mais ils se concentrent sur les business cases plutôt que sur les composants eux-mêmes.
- **Les tests d'interface utilisateur** permettent de s'assurer que l'application fonctionne correctement du point de vue de l'utilisateur.

Concernant les tests fonctionnels, ceux-ci devront être réalisés par un praticien de santé une fois l'interface de consultation des lits d'urgence mis en place.

Voici un schéma illustrant la pyramide des tests :



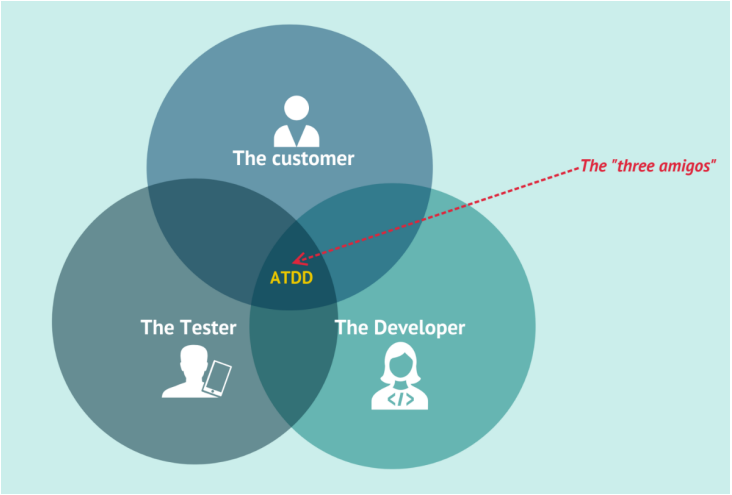
Le cycle BDD

Les test s'effectueront selon le modèle BDD (Business Driven Development)

Le cycle de BDD se décompose en 3 étapes :

1. **Discovery** : on recueille les besoins
2. **Formulation** : on reformule le besoin métier sous forme de cas d'usages, puis d'exemples (la formulation)
3. **Automatisation** : enfin, on testera la bonne mise en application des règles métiers directement dans notre code (l'automatisation)

La mise en place de cycles BDD implique la collaboration de membres de l'équipe représentatifs de chaque grande famille d'intervenants (3 amigos ou les 3 amis). Il s'agit d'une discussion entre tous les membres de l'équipe aussi appelée « Specification Workshop » pour « Atelier de spécification ».



Le BDD a un intérêt pour les business analysts, les product owners, les testeurs, les développeurs, les experts métiers...

3 amis : utilisateurs, équipe technique, personnes du métier
Les 3 amis construisent la vision commune (le centre du cercle)

Les 3 amis sont des personnes qui ont les rôles suivants :

- Business (le représentant métier) : définit le problème ou la fonctionnalité attendue, a besoin de réponses, définit la valeur business (Product Owner, Business Analyst sont aussi dans la boucle).
- Développeurs : suggèrent un moyen de corriger ce problème ou de créer la fonctionnalité.
- Utilisateur / Testeur : challenge tout le monde sur ce qui est attendu (cherche les problèmes et les failles dans le raisonnement).

En théorie à la fin de la réunion des 3 amis, tous les partis prenants doivent avoir atteint leur but.

Tests d'acceptance BDD

Les scénarios de tests d'acceptances décrits dans ce chapitre répondent aux critères et procédures d'acceptances visibles au cahier des charges.

Ces tests comportementaux ou Behaviour Driven Development en anglais (BDD) sont écrits en **Gherkin**, langage permettant une approche structurée de l'écriture de tests.

Suivant l'interconnexion des composants et si les tests se font en local ou sur les infrastructures de production, ces tests sont de type End to end (ou de bout en bout). Ils permettent de tester la fonctionnalité sur tous les composants (Applicatif, technologique, infrastructure, data) entrant en jeu pour son exécution.

Un seul test peut donc concerner l'interface utilisateur frontend, les services backend, les bases de données mais aussi l'infrastructure qui les hébergent.

Nos scénarios BDD seront écrits en Gherkin qui est le format des spécifications de Cucumber. Ils seront compréhensibles par toutes les parties prenantes et sont la source de notre documentation applicative.

PROCÉDURE DE TEST DE LA POC

Périmètre

Pour rappel, le développement de l'application POC a consisté à créer une API minimale permettant l'interrogation du lit disponible le plus proche en fonction de la localisation géographique du patient et la spécialité médicale requise pour l'intervention d'urgence. Ces 2 paramètres sont :

- l'id de l'enregistrement de la table spécialité concerné par l'intervention
- La position GPS du patient exprimé provisoirement sous la forme d'un entier (la requête d'interrogation de l'API procède à un simple calcul d'écart entre cette valeur et la position GPS de l'établissement auquel le lit est rattaché)

Les tests vont consister à vérifier que L'API retourne un objet JSON contenant la référence du lit disponible, le nom de l'établissement auquel il appartient et la position GPS de celui-ci.

Préparation

Avant de démarrer les tests il est bien évidemment nécessaire d'exécuter l'application Java pour mettre en route l'API sur un serveur http. On lancera pour cela l'exécution du fichier principal du projet POC nommé p11apiApplication.java (depuis l'IDE Eclipse par exemple). Cette opération lance automatiquement sur le poste local un serveur http Tomcat rendant l'API accessible à l'url suivante : http://localhost:9000/patient/urgence.

Les données constituant le jeu d'essai interrogeable via l'API devront être préalablement saisies dans la base de données relationnelle H2 connectée au serveur d'application Tomcat. Cette opération s'effectue grâce à la console H2 lancée dès l'exécution de l'application. Elle est accessible via un navigateur web à l'adresse suivante : http://localhost:9000/h2-console.

Pour terminer, nous devons disposer d'un outil permettant de consulter l'API via le protocole http. L'API doit être fonctionnelle quel que soit le dispositif de consultation. Pour le test nous utiliserons une installation locale de Postman dans laquelle nous allons créer une requête http de type "form-data" (POST) avec comme cible l'URL de l'API et comme paramètres ceux attendus par la fonction d'interrogation à savoir specialiteId et gpsPosition.

Mode opératoire

Concrètement, le test va consister à lancer l'exécution de l'application Java de l'API en local, saisir dans Postman les paramètres d'entrée souhaités pour la requête puis l'envoyer (Bouton "Send").

Le résultat de la requête sera vérifié dans l'onglet Body de la console Postman.

Jeu d'essai

Le script SQL **data.sql** permettant la création des données de base figure dans le dossier src/main/resources de l'application.

Pour les besoins du test 3 tables ont été nécessaires : lit, specialite, etablissement.

3 spécialités et 3 établissements ont été saisis dans leurs tables respectives. 2 lits par établissement ont été saisis dans la table des lits avec un panachage équilibré des spécialités associées (2 enregistrements par spécialité).

Validation SQL avec la console H2

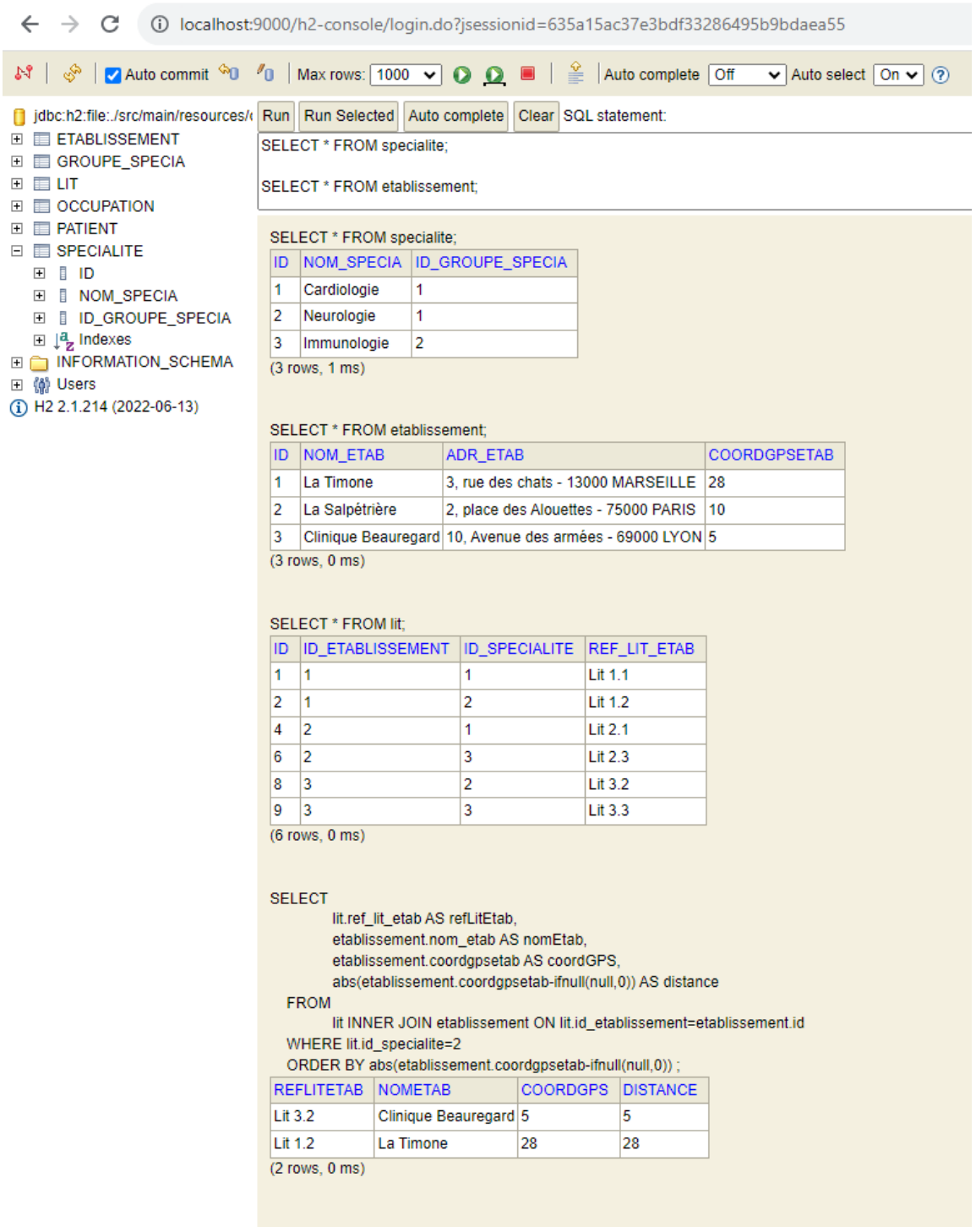
Avant d'effectuer des tests sur l'application POC il est nécessaire de s'assurer de la cohérence des données sources et la pertinence des résultats fournis par la requête SQL d'interrogation des lits disponibles servant de base au service urgence de l'API.

Pour cela il est fourni le script SQL suivant à lancer dans la console H2 :

```
SELECT * FROM specialite;

SELECT * FROM etablissement;

SELECT * FROM lit;

SELECT

        lit.ref_lit_etab AS refLitEtab,

        etablissement.nom_etab AS nomEtab,

        etablissement.coordgpsetab AS coordGPS,

        abs(etablissement.coordgpsetab-ifnull(7,0)) AS distance

  FROM

        lit INNER JOIN etablissement ON lit.id_etablissement=etablissement.id

  WHERE lit.id_specialite=2

  ORDER BY abs(etablissement.coordgpsetab-ifnull(7,0)) ;
```

(*) Noter qu'il est nécessaire de remplacer les valeurs 2 et 7 correspondant aux paramètres d'entrée de la requête (spécialité de l'urgence et position GPS du patient) par celles de son choix.

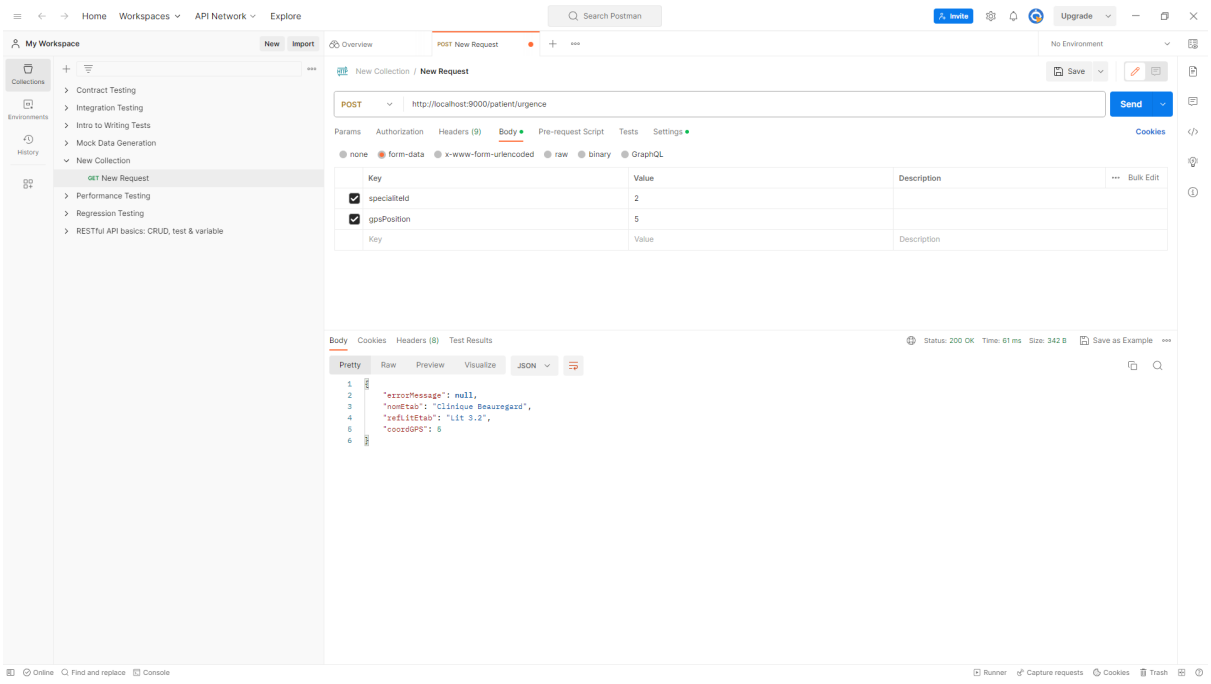
Une capture d'écran de la console H2 après exécution du script est fournie page suivante.



Vérification de l'api avec Postman

Afin de s'assurer à présent du bon fonctionnement de l'application, on doit tout d'abord la lancer puis consulter l'api urgence via le client postman. Pour cela on créera une nouvelle requête dans postman. Une fois saisis les

paramètres d'entrée dans l'onglet body(form-data) on appuiera sur le bouton "Send". Le résultat apparaît sous forme d'objet JSON dans la section retour en bas de l'interface ainsi que le statut (200 si succès, 403 sinon). On vérifiera la correspondance des résultats du script SQL décrit plus haut avec les données retournées lors de l'appel du service par postman.



SCÉNARIOS DE TESTS DE L'APPLICATION POC

L'application de test est constituée d'une seule classe concentrant tous les tests nécessaires, elle est nommée **P11apiApplicationTests.java**. Elle doit être exécutée avec le menu Eclipse Run as/JUnit.

Les tests sont regroupés en 3 parties.

Fonctionnement nominal

Cette partie consiste à contrôler que la requête du service de consultation des lits disponibles en situation d'urgence retourne les bons

enregistrements en fonction des paramètres d'entrée à savoir l'id de la spécialité souhaitée (**specialiteId**) et la position GPS du patient (**gpsPosition**). Pour un enregistrement résultat, les méthodes du service **findLit** (getNomEtab, getCoordGPS et getRefLitEtab) retournent respectivement le nom et la coordonnée GPS de l'établissement et la référence du lit disponible dans celui-ci. Le test va consister à vérifier que les valeurs retournées par ces méthodes correspondent bien à celles attendues.

Il est à noter que pour tous les cas figurant dans cette première partie, nous supposons que le type de donnée saisi pour les 2 paramètres d'entrée est correct (entier long). Les cas correspondant à des saisies de paramètres d'entrée incorrects sont gérés dans la partie 2 (cas limites). Concernant la position GPS du patient, la valeur 0 ou une valeur négative sont acceptables, la requête effectuant une simple soustraction en valeur absolue entre les positions GPS du lit et du patient pour déterminer leur proximité.

1er cas : Sélection du lit le plus proche pour une spécialité donnée

Paramètres d'entrée : specialitéId = 2, gpsPosition = 20

Résultat attendu (Objet JSON retourné) :

{

"nomEtab": "La Timone",

"coordGPS": 28,

"refLitEtab": "Lit 1.2"

}

2ème cas : Sélection du lit le plus proche pour la même spécialité et une position GPS différente

Paramètres d'entrée : specialitéId = 2, gpsPosition = 5

Résultat attendu :

{

"nomEtab": "Clinique Beauregard",

"coordGPS": 5,

"refLitEtab": "Lit 3.2"

}

3ème cas : Aucun lit trouvé pour une spécialité existante

Nous noterons que la saisie d'un code de spécialité inexistant n'est pas gérée par le service, elle nécessiterait de faire une pré-requête pour contrôler l'existence de la spécialité dans la table spécialité. Étant convenu qu'en production l'accès au service s'effectuera à travers une interface web, cette requête sera effectuée en amont pour alimenter la combo servant à saisir le service. Dans le cadre du test nous supposons simplement que le type de valeur saisie pour la spécialité correspond bien au type attendu c'est à dire un entier.

Dans le cas de test nous attendons que le service nous retourne un objet dans lequel les trois propriétés, spécialité, nom établissement et coordonnées GPS sont égales à la valeur *null* et la propriété errorMessage est "Aucun lit disponible pour cette spécialité.".

Paramètres d'entrée : specialitéId = 40 (sachant que seuls les id compris entre 1 et 3 sont représentés dans le jeu d'essai), gpsPosition = 7

Résultat attendu :

{

"nomEtab": null,

"coordGPS": null,

"refLitEtab": null,

“errorMessage” : “Aucun lit disponible pour cette spécialité.”

}

Code statut de la requête : 202 Accepted

Cas limites

Cette partie consiste à contrôler que le service de recherche de lits disponibles renvoie les bons messages d'erreur lorsque les paramètres d'entrée sont vides ou qu'ils ne correspondent pas au type attendu (entier).

Ces cas sont en dehors des périmètre de test ceux-ci ne pouvant tester que la pertinence des résultats et non l'implémentation du code.

Via postman nous pouvons tester que dans ce cas l'api retourne bien la valeur **1** signifiant que l'api a été atteinte mais avec le code statut **403 Forbidden.**

PLAN DE TEST

Scénarios BDD

Pour réaliser le plan de test nous allons reprendre le tableau des fonctionnalités à implémenter dans le cadre de l'architecture cible exposée dans les documents cahier des charges et définition d'architecture.

Les scénarios de tests d'acceptances décrits dans ce chapitre répondent aux critères et procédures d'acceptances visibles au cahier des charges. Ils testent les fonctionnalités des différents composants qui seront ajoutés à l'architecture actuelle et qui sont représentés dans le schéma d'architecture applicative du DDA (Document de Définition d'Architecture). Les tests à effectuer sont résumés dans le tableau suivant :

Fonctionnalité	**Scénario BDD**
Fonctionnalités de recherche du lit le plus proche pour accueillir un patient en urgence	
1 - Récupération de la référence du lit et de la localisation GPS de son établissement **Composants testés** :	**Contexte :** L'utilisateur (le praticien dans le futur) est connecté à la plateforme et accède au menu de recherche de lit en urgence. Dans le cadre de la POC, le serveur d'API est démarré, les données à consulter sont présentes en base et la requête Postman est correctement paramétrée (l'url cible de l'api et les paramètres d'entrée sont correctement nommés).

API de recherche de lit en urgence	**Scénario :** Vérifier que l'API retourne le lit le plus proche pour une spécialité donnée et la position GPS du patient **Given** Le praticien saisit l'id d'une spécialité référencée dans la base ainsi que la position GPS du patient (un entier) dans les paramètres d'entrée de la requête Postman. **When** Le praticien clique sur le bouton "Send" de la requête Postman **Then** la plateforme renvoie bien les références du lit le plus proche présent en base **Scénario :** Vérifier que l'API retourne le lit le plus proche pour la même spécialité et une position GPS différente **Given** Le praticien saisit l'id d'une spécialité référencée dans la base ainsi que une autre position GPS du patient (un entier) dans les paramètres d'entrée de la requête Postman. **When** Le praticien clique sur le bouton "Send" de la requête Postman **Then** la plateforme renvoie bien les références du lit le plus proche présent en base **Scénario :** Vérifier que l'API retourne un enregistrement vide avec le message d'erreur "Aucun lit trouvé pour cette spécialité" lorsque le praticien saisit un code de spécialité non référencée dans le jeu d'essai **Given** Le praticien saisit l'id d'une spécialité non référencée dans la base et une des positions GPS saisies précédemment pour laquelle la plateforme avait retourné un résultat. **When** Le praticien clique sur le bouton "Send" de la requête Postman **Then** la plateforme renvoie un résultat nul avec le message d'erreur et le statut attendu. **Scénario :** Vérifier que l'API retourne lorsque le praticien ne saisit aucune spécialité ou une valeur non numérique pour la spécialité **Given** Le praticien ne saisit aucune valeur pour le paramètre spécialité. **When** Le praticien clique sur le bouton "Send" de la requête Postman **Then** la plateforme retourne la valeur 1 et le statut http 403 dans la console postman.

Tests non réalisés

Tests de sécurité

Les tests de sécurité ne sont pas à réaliser dans la mesure où nous nous appuyons sur une plateforme avec un contrat de service garantissant ces aspects.

Tests de charge

Le test de charge permet de mesurer la performance d'un système en fonction de la charge d'utilisateurs simultanés. Ceux-ci ne peuvent être réalisés que dans la mesure où nous disposons d'une plateforme de test conforme à celle présente en production réelle ce qui est prématuré dans le cadre d'une POC

Test Fuzzing

Le Fuzzing, est une technique de test visant à injecter de grandes quantités de données aléatoires dans les entrées de l'application pour détecter tout crash ou comportement négatif.
Ils seront réalisés une fois l'application en production.

Détection des temps d'exécution et de réponse

Par le biais de nos outils de tests automatisés et de reporting ou des outils des fournisseurs cloud il sera important de tester le temps de réponse des applications frontend, mais aussi des services et API de MedHead pour s'assurer que le temps d'exécution et de réponse est acceptable (Inférieur à 400ms).

Tests de non-régression

Les tests de non régression vérifient que l'application ou les fonctionnalités déjà existantes fonctionnent toujours correctement après une modification ou mise à jour d'une autre partie de l'application.

Dans notre cas ils sont assurés, à chaque build ou mise en production d'un script de code, par l'automatisation des différents tests unitaires, d'intégration, système et d'acceptance.