Plan de test

LABOUCHE Bastien

16 mars 2023

Sommaire

1 Informations sur le document

Nom du projet :	MedHead - Preuve de concept Système d'intervention d'urgence		
Préparé par :	Bastien Labouche	**Version du document :**	1.0
Titre :	Plan de test	**Date de publication :**	16 mars 2023
Relu par :	N/A	**Date de relecture :**	N/A

Liste de distribution

De :	**Date :**	**Téléphone/Fax/Email**
Bastien		

Pour :	**Action**	**Date :**	**Téléphone/Fax/Email**
Kara Trace	Relecture		
Anika Hansen	Relecture		
Ashley Ketchum	Relecture		
Chris Pike	Relecture		

Historique des versions

Version	**Date**	**Révisé par**	**Description**	**Nom du fichier**
1.0	16 mars 2023		Première version	

2 introduction

De la décision de fusionner tous les savoirs faire présent au sein du consortium a résulté le projet de création d'une plateforme nouvelle génération pour la prise en charge des patients.

Dans ce document nous nous intéressons à un sous-système précis de la plateforme : le système d'intervention d'urgence en temps réel (que j'appellerai "système d'intervention d'urgence" dans le reste de ce document) qui est responsable de la prise en charge d'un patient à la suite d'un accident, l'objectif étant d'indiquer aux secours l'hôpital le plus proche pouvant prendre en charge le/les patients.

Cependant un tel système soulève plusieurs inquiètude auprès des membres du consortium, dont certaines à l'origine de la création de la preuve de concept :

- Est-ce que le système d'intervention d'urgence est adapté aux incidents (techniques) ?
- Est-ce que le système d'intervention d'urgence gère la latence concernant la disponibilité des lits des hôpitaux du réseau ?
- Est-ce que le système d'intervention d'urgence peut répondre dans les 200 millisecondes à la demande de lit ?

Pour répondre aux interrogations ci-dessus a été décidé la création d'une preuve de concept dont l'objectif est de renvoyer dans un délai impartis l'hôpital le plus proche ayant des lits disponibles. La preuve de concept devra être testée en suivant la pyramide des tests :

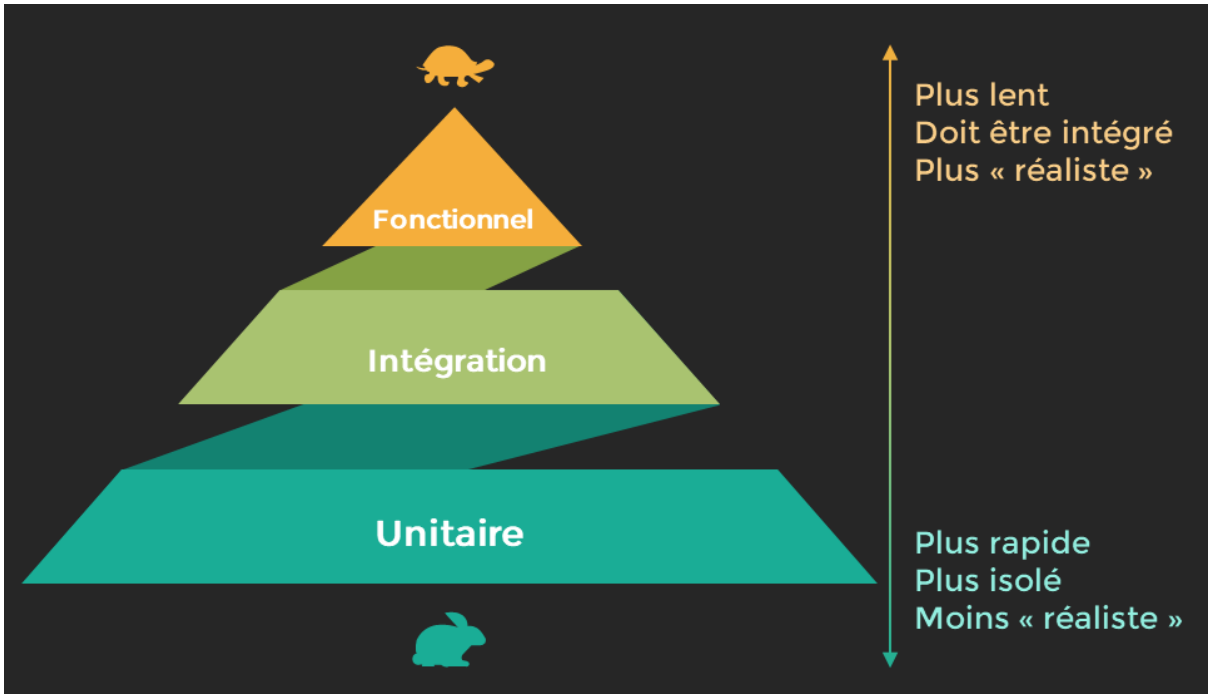

Figure 1: Pyramide des tests

3 Cadre

3.1 Global

L'objectif ici est de tester uniquement le POC du système d'intervention d'urgence sur certains points particuliers :

- Le système d'intervention d'urgence est fiable, en cas d'erreur critique sur une instance le service reste disponible
- Le système d'intervention d'urgence peut répondre en moins de 200 millisecondes avec une charge de travail allant jusqu'à 800 requêtes par seconde

- Le système d'intervention d'urgence gère la latence concernant la disponibilité des lits des hôpitaux du réseau

4 Hors cadre

Les autres services de la plateforme ne sont pas concernés par ce document qui se concentre uniquement sur la preuve de concept du système d'intervention d'urgence.
Cette preuve de concept, même si elle peut être utilisée avec d'autres systèmes est conçue pour pouvoir fonctionner de manière autonome et sera donc testée en tant que tel.

5 Plan de test

HospitalService = Classe chargée de gérer les requêtes concernant les hôpitaux envoyées à la base de données

5.1 Tests unitaires

5.1.1 Avant tous les tests

1. Initialisation du Mock de la classe HospitalService et de son comportement.
   - Initialisation du comportement de l'objet "mocké" lors de l'appel de la fonction visant à générer une liste contenant l'ensemble des hôpitaux du système
   - Initialisation du comportement de l'objet "mocké" lors de l'appel de la fonction visant à récupérer un hôpital particulier via son ID en base.
2. Génération du kd-tree dans le contrôleur avec les données du service "mocké"

5.1.2 Avant chaque test

- Reset de la liste des hôpitaux

5.1.3 Tests

1. Test que le contrôleur renvoie l'ensemble des hôpitaux transmis par le service "mocké"
2. Test que le contrôleur renvoie l'hôpital le plus proche des coordonnées transmises parmis ceux de la liste fournie par le service "mocké". Vérification que le nombre de lits disponibles est décrémenté de 1.
3. Test que le contrôleur renvoie l'hôpital le plus proche (ayant des lits disponibles) des coordonnées transmises parmis ceux de la liste fournie par le service "mocké". Vérification que le nombre de lits disponibles est décrémenté de 1.
4. Test qu'une erreur est renvoyée si les coordonnées ne sont pas assez précises
5. Test qu'une erreur est renvoyée si les coordonnées sont trop éloignées de l'hôpital le plus proche. Dans les tests le système cherche les hôpitaux dans un rayon de 500km autour des coordonnées mais ce rayon est paramétrable.

5.2 Tests d'intégration

À la différence des tests unitaires qui se contentaient de simuler les données échanges avec la base de données, les tests d'intégration la rajoute pour voir si le système entier fonctionne par lui-même.

5.2.1 Avant tous les tests

N/A

5.2.2 Avant chaque test

N/A

5.2.3 Tests

1. Test que le contrôleur peut renvoyer l'ensemble des hôpitaux contenus dans la bdd
2. Test que le contrôleur renvoie l'hôpital le plus proche des coordonnées transmises parmis ceux enregistrés dans la bdd. Vérification que le nombre de lits disponibles est décrémenté de 1.
3. Test que le contrôleur renvoie l'hôpital le plus proche (ayant des lits disponibles) des coordonnées transmises parmis ceux enregistrés dans la bdd. Vérification que le nombre de lits disponibles est décrémenté de 1.
4. Test qu'une erreur est renvoyée si les coordonnées ne sont pas assez précises
5. Test qu'une erreur est renvoyée si les coordonnées sont trop éloignées de l'hôpital le plus proche. Dans les tests le système cherche les hôpitaux dans un rayon de 500km autour des coordonnées mais ce rayon est paramétrable.

5.3 Tests fonctionnels

Les tests fonctionnels sont effectués durant la pipeline CI/CD sur un serveur différent de celui faisant tourner les applications. Pour les réaliser, de véritables requêtes GET sont envoyées.

5.3.1 Tests

1. Test que le contrôleur renvoie l'hôpital le plus proche des coordonnées transmises. Vérification que le nombre de lits disponibles est décrémenté de 1.
2. Test que le contrôleur renvoie l'hôpital le plus proche ayant des lits disponibles des coordonnées transmises. Vérification que le nombre de lits disponibles est décrémenté de 1.
3. Test qu'une erreur est renvoyée si les coordonnées ne sont pas assez précises
4. Test qu'une erreur est renvoyée si les coordonnées sont trop éloignées de l'hôpital le plus proche. Dans les tests le système cherche les hôpitaux dans un rayon de 500km autour des coordonnées mais ce rayon est paramétrable.
5. Test de charge: le système doit supporter 800 requêtes par seconde et quand-même être capable d'indiquer un hôpital en moins de 200ms
6. Test de fiabilité: en cas de crash d'un pod il doit être automatiquement recréé.

6 Signatures

LABOUCHE Bastien