Hypothèse de validation des principes

LABOUCHE Bastien

20 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 :	Hypothèse de validation des principes	**Date de publication :**	20 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	20 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 inquiétudes 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.

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.

4.1 Scénarios

Pour répondre aux interrogations du consortium, plusieurs scénarios de test ont été envisagés :
1/

- **Étant donné** Que des hôpitaux sont disponibles
- **Quand** L'utilisateur lance une recherche en indiquant des coordonnées suffisamment précises
- **Alors** Le système retourne l'hôpital le plus proche

2/

- **Étant donné** Que des hôpitaux sont disponibles
- **Quand** L'utilisateur lance une recherche en indiquant des coordonnées trop peu précises
- **Alors** Le système retourne une erreur

3/

- **Étant donné** Que le système connaît l'emplacement de chaque hôpital et que des hôpitaux sont (ou pas) enregistrés
- **Quand** L'utilisateur lance une recherche en étant à une distance supérieur ou égale à 500km de l'hôpital le plus proche
- **Alors** Le système retourne une erreur

4/

- **Étant donné** Que des hôpitaux sont disponibles
- **Quand** L'utilisateur en demande un
- **Alors** Le système retourne l'hôpital le plus proche

5/

- **Étant donné** Que des hôpitaux sont disponibles mais que le système reçoit plus de 800 requêtes par secondes
- **Quand** L'utilisateur demande un hôpital
- **Alors** Le système lui retourne l'hôpital le plus proche en moins de 200ms

6/

- **Étant donné** Que le système est en ligne
- **Quand** Un problème survient
- **Alors** Le système reste accessible et fonctionnel

5 Stratégie

Les scénarios 1, 2, 3 et 4 seront testés sur tous les étages de la Pyramide des tests, tandis que les scénarios 5 et 6 ne seront testés que lors des tests fonctionnels

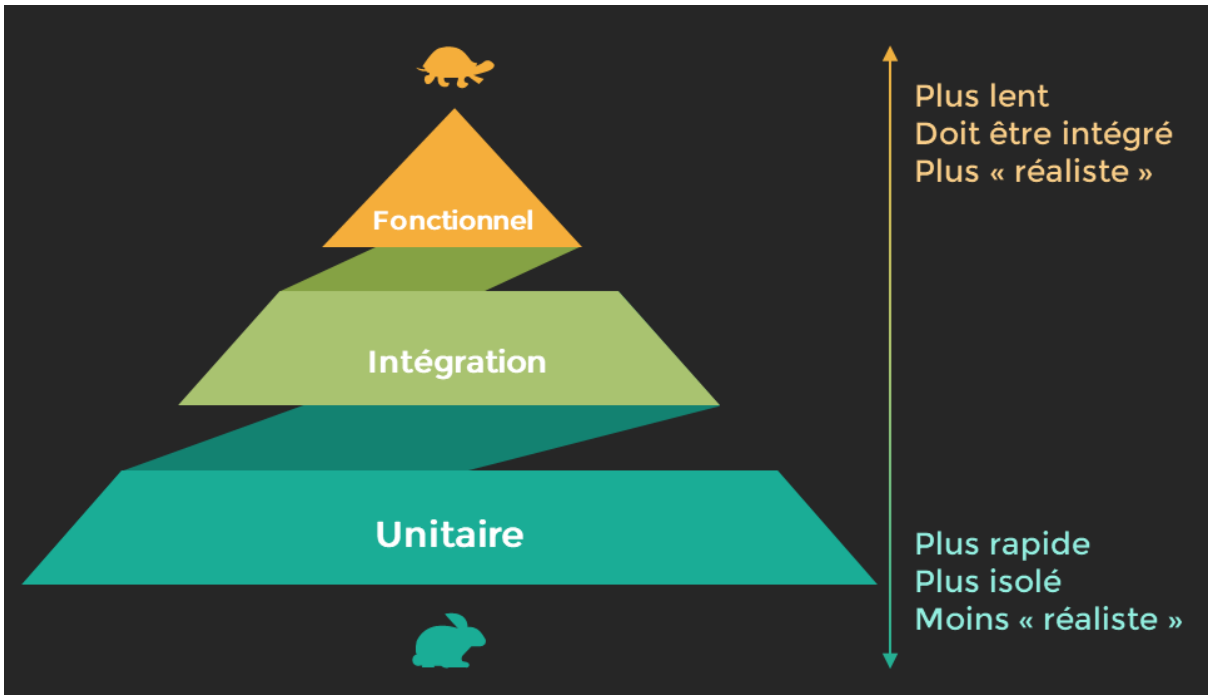

Figure 1: Pyramide des tests

5.1 Données utilisées

Obtenues via le site du nhs : Liste des hôpitaux du nhs

Pour faciliter le développement de la preuve de concept, j'ai estimé qu'il était pertinent de n'avoir qu'une seule base de données manipulant la liste des hôpitaux et de toute information pertinente les concernant, tel que la liste des spécialités accessibles ou de la quantité de lits disponibles.
Pour ça j'ai attribué de manière aléatoire à chaque hôpital des spécialités ainsi qu'un nombre de lit avant de transformer le fichier .xlsx en fichier en fichier sql respectant le schémas de base de données suivant :

Figure 2: Schémas de la base de données

Comme vous pouvez le constater, les spécialités sont prise en compte dans la conception de la preuve de concept même si elles ne sont pas utilisées, elles ont été ajoutée dès la conception de manière à anticiper les prochaines extensions de la preuve de concept et son intégration dans d'autres systèmes.

5.2 Déclenchement des tests - Pipeline CI/CD

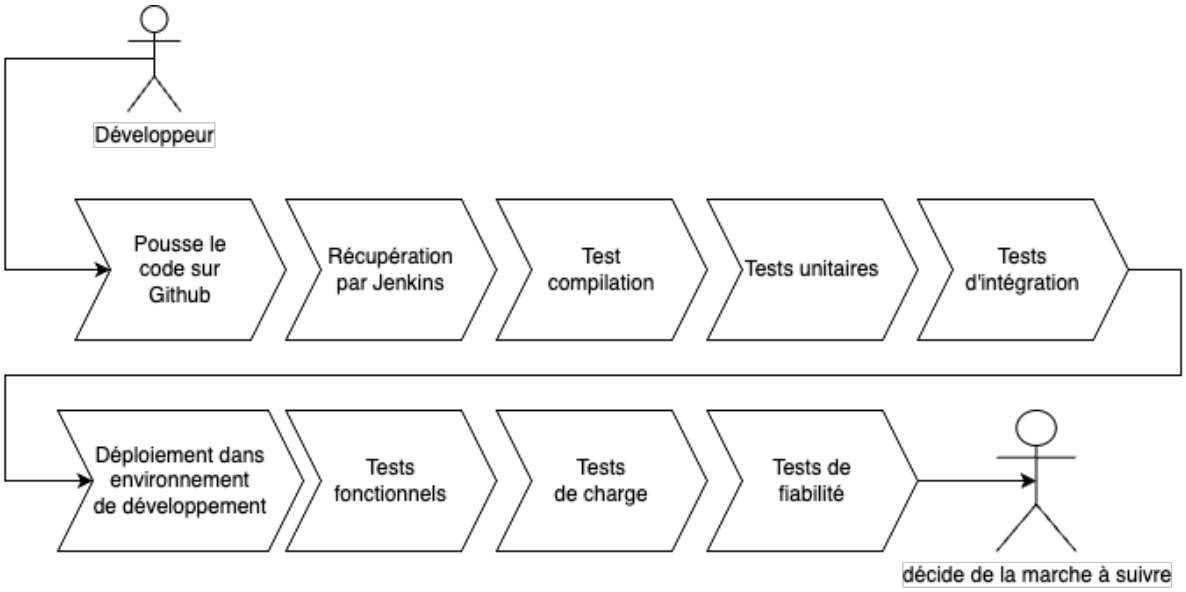

Figure 3: Pipeline CI/CD

Ici l'environnement de dev est un cluster Kubernetes sur lequel est déployé Jenkins. La récupération du code sur le repo se fait automatiquement toutes les 15 minutes par Jenkins et la pipeline n'est déclenchée que si du code a été ajouté.
Les tests fonctionnels des scénarios 1, 2, 3 et 4 sont exécutés dans une pipeline Jenkins en utilisant les fonctionnalités fournies par le plugin Http Request.

6 Exécution des tests

6.1 Tests unitaires

Les tests unitaires vont ici venir tester le contrôleur en charge des hôpitaux.
Sera "mocké" le service en charge des hôpitaux auquel fait appel le contrôleur. Sont simulées les fonctions permettant de lister les hôpitaux, de les retrouver via leur identifiant et de les sauvegardées. Mais vu que cette fonction n'est pas directement utilisée par les tests nous n'en reparlerons pas plus.

Les tests unitaires consistent donc à l'appel des différentes fonctions du contrôleur en se servant de ce service mocké.

6.2 Tests d'intégration

Pour les tests d'intégration pas de mock, on intègre la base de données. Les requêtes http sont simulées via Mockito.

6.3 Tests fonctionnels

Les tests fonctionnels sont exécutés via de véritables requêtes HTTP lancées dans une pipeline grâce au plugin Http Request

6.4 Tests particuliers

- **Tests de charge :** en utilisant k6 on monte progressivement de 0 à 800 UVs en 10 secondes, avant de stagner à 800 UVs pendant 10 minutes puis de redescendre à 0 UVs en 30 secondes.
  UV = Utilisateur Virtuel.
  Chaque utilisateur virtuel est en réalité une boucle "while" envoyant une requête http toutes les secondes.
  Critères de validation :
  - Moins de 1% d'erreurs
  - Au moins 95% des requêtes ont une réponse en moins de 200ms
- **Test de fiabilité :** un endpoint de test "/stop-spring" a été créé pour simuler un crash de l'application : Lors de la réception d'une requête GET dessus l'application se ferme. Pour valider ce test, un pod doit être supprimé et un autre doit prendre sa place.

7 Signatures

LABOUCHE Bastien