HYPOTHESE DE DEVELOPPEMENT D'UNE PREUVE DE CONCEPT POUR LE SOUS-SYSTEME D'INTERVENTION D'URGENCE EN TEMPS REEL

Projet : Preuve de concept (l'allocation de lits d'hôpital pour les urgences)

Client : Consortium MedHead

Auteur : Loïc PIRIOU

HYPOTHESE DE DEVELOPPEMENT

Table des matières

HYPOTHESE DE DEVELOPPEMENT

1 Déclaration d'hypothèse

Nous pensons que la mise en œuvre d'une preuve de concept pour le sous-système d'intervention d'urgence en temps réel par l'équipe d'architecture métier du Consortium MedHead permettra :

- D'améliorer la qualité des traitements d'urgence et de sauver plus de vies
- De gagner la confiance des utilisateurs quant à la simplicité d'un tel système

Nous saurons que nous avons réussi quand nous verrons :

- Que plus de 90 % des cas d'urgence sont acheminés vers l'hôpital compétent le plus proche du réseau
- Que le temps moyen de traitement d'une urgence passe de 18,25 minutes (valeur actuelle) à 12,00 minutes (valeur souhaitée)
- Que nous obtenons un temps de réponse de moins de 200 millisecondes avec une charge de travail allant jusqu'à 800 requêtes par seconde, par instance de service
- Que la mise en œuvre explique les normes qu'elle respecte et pourquoi
- Que les instructions pour mettre en production la PoC sont fournies
- Que la mise en œuvre est terminée dans le délai imparti.

2 Exemple de comportement et description de la capacité

Le sous-système d'intervention d'urgence en temps réel est destiné à recevoir une ou plusieurs spécialités médicales (voir les Données de référence sur les spécialités) et une banque de données d'informations récentes sur les hôpitaux afin de suggérer l'hôpital le plus proche offrant un lit disponible, associé à une ou plusieurs spécialisations correspondantes. Le lieu de l'incident d'urgence doit également être fourni.
Par exemple, SUPPOSONS trois hôpitaux, comme suit :

Hôpital	Lits disponibles	Spécialisations
Hôpital Fred Brooks	2	Cardiologie, Immunologie
Hôpital Julia Crusher	0	Cardiologie
Hôpital Beverly Bashir	5	Immunologie, Neuropathologie diagnostique

ET un patient nécessitant des soins en cardiologie.
QUAND vous demandez des soins en cardiologie ET que l'urgence est localisée près de l'hôpital Fred Brooks
ALORS l'hôpital Fred Brooks devrait être proposé
ET un événement devrait être publié pour réserver un lit.

MedHead

HYPOTHESE DE DEVELOPPEMENT

3 Exigences convenues de la POC

Les exigences suivantes ont été convenues lors de la définition de cette hypothèse :

- Fournir une API RESTful qui tient les intervenants médicaux informés en temps réel sur : le lieu où se rendre et ce qu'ils doivent faire.
- S'assurer que toutes les données du patient sont correctement protégées.
- S'assurer que votre PoC est entièrement validée avec des tests d'automatisation reflétant la pyramide de test (tests unitaires, d'intégration, d'acceptation et E2E) et avec des tests de stress pour garantir la continuité de l'activité en cas de pic d'utilisation.
- S'assure que la PoC peut être facilement intégrée dans le développement futur : rendre le code facilement partageable, fournir des pipelines d'intégration et de livraison continue (CI/CD) et documenter votre stratégie de test.
- S'assurer que les équipes de développement chargées de cette PoC sont en mesure de l'utiliser comme un jeu de modules de construction pour d'autres modules.

4 Méthodologie

La documentation et la PoC qui en résulteront seront présentées aux membres du Conseil d'administration pour décrire les enseignements tirés de la PoC. Des rapports sur les méthodes CI/ CD seront présentés au personnel technique afin d'expliquer comment mettre à jour le système.
Afin de répondre aux exigences du POC, il faudra mettre en place des tests unitaires, d'intégrations et fonctionnelles répondant à la pyramide de test.
Les tests unitaires permettront de tester de manière isolée les fonctionnalités du POC.
Les tests d'intégrations permettront de tester les unités de code ensemble, on vérifie les interactions entre ses bouts de code et on s'assure déjà du bon fonctionnement du POC dans son ensemble.
Les tests fonctionnels permettront de tester l'ensemble de l'API sans prendre en compte les unités ou les blocs de code mais plutôt de voir si celle-ci réponds au exigences voulu.
Pour répondre aux exigences de charge et de réponse dans un temps voulu pour l'API, la mise en place de tests de charge sont essentielles. Ils consistent à mesurer la performance du système en fonction de la charge d'utilisateurs simultanées. L'objectif est de prévoir la charge maximale que peut encaisser l'API. Il permet également de mettre en évidence les points de vigilances du système, de les corriger et enfin de valider les performances de l'API.
Une pipeline CI/CD d'intégration et de livraison continue s'assurant de la qualité du code, permettra aussi le déploiement facile de l'API. Elle permettra :

- Le lancement automatisé des tests unitaires et d'intégrations
- La vérification de la qualité du code et de s'assurer que celui-ci peut être déployé en production
- La génération du jar de l'API
- La possibilité du déploiement en production

5 Résultats

Les tests unitaires et d'intégrations sont lancés via Maven et le résultat des tests peut être obtenu via le plugin Jacoco et Surefire. Ci-dessous nous pouvons voir le résultat du lancement des tests pour la POC :

- Résultat Jacoco :

ms-emergency

ms-emergency

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.medhead.msemergency.model		20 %		5 %	99	134	4	21	19	54	0	4
com.medhead.msemergency		10 %		0 %	28	34	6	8	9	15	0	2
com.medhead.msemergency.service		36 %		5 %	27	37	4	20	8	17	0	1
com.medhead.msemergency.repository		15 %		n/a	5	13	29	37	5	13	1	5
com.medhead.msemergency.configuration		47 %		10 %	7	13	26	53	2	8	0	2
com.medhead.msemergency.dto		100 %		n/a	0	2	0	7	0	2	0	1
com.medhead.msemergency.controller		100 %		n/a	0	2	0	2	0	2	0	1
Total	1 276 of 1 735	26 %	236 of 248	4 %	166	235	69	148	43	111	1	16

- Résultat Surefire :
  - Controller :

```
-------------------------------------------------------------------------------
Test set: com.medhead.msemergency.controller.EmergencyControllerIntegrationTest
-------------------------------------------------------------------------------
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.714 s - in com.medhead.msemergency.controller.EmergencyControllerIntegrationTest
```

```
-------------------------------------------------------------------------------
Test set: com.medhead.msemergency.controller.EmergencyControllerUnitTest
-------------------------------------------------------------------------------
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.39 s - in com.medhead.msemergency.controller.EmergencyControllerUnitTest
```

  - Service :

```
-------------------------------------------------------------------------------
Test set: com.medhead.msemergency.service.EmergencyServiceUnitTest
-------------------------------------------------------------------------------
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 4.503 s - in com.medhead.msemergency.service.EmergencyServiceUnitTest
```

  - DTO :

```
-------------------------------------------------------------------------------
Test set: com.medhead.msemergency.dto.NearestHospitalReservationTransformUnitTest
-------------------------------------------------------------------------------
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.004 s - in com.medhead.msemergency.dto.NearestHospitalReservationTransformUnitTest
```

Nous pouvons constater que l'ensemble du code est couvert en terme de tests unitaires et d'intégrations. L'ensemble des tests passent correctement et aucune anomalie n'est constaté dans le code du POC.
Concernant les tests de charge ceux-ci sont lancés via l'application JMeter qui nous permet de mettre en place un cas d'utilisation utilisateur donc d'effectuer un test fonctionnel de l'API. JMeter nous permet aussi de mettre en place des tests de charge afin d'évaluer la capacité du nombre d'utilisateurs que peut prendre en charge l'APi et le temps de réponse de l'API suivant cette capacité.

HYPOTHESE DE DEVELOPPEMENT

En prenant les paramètres suivants :

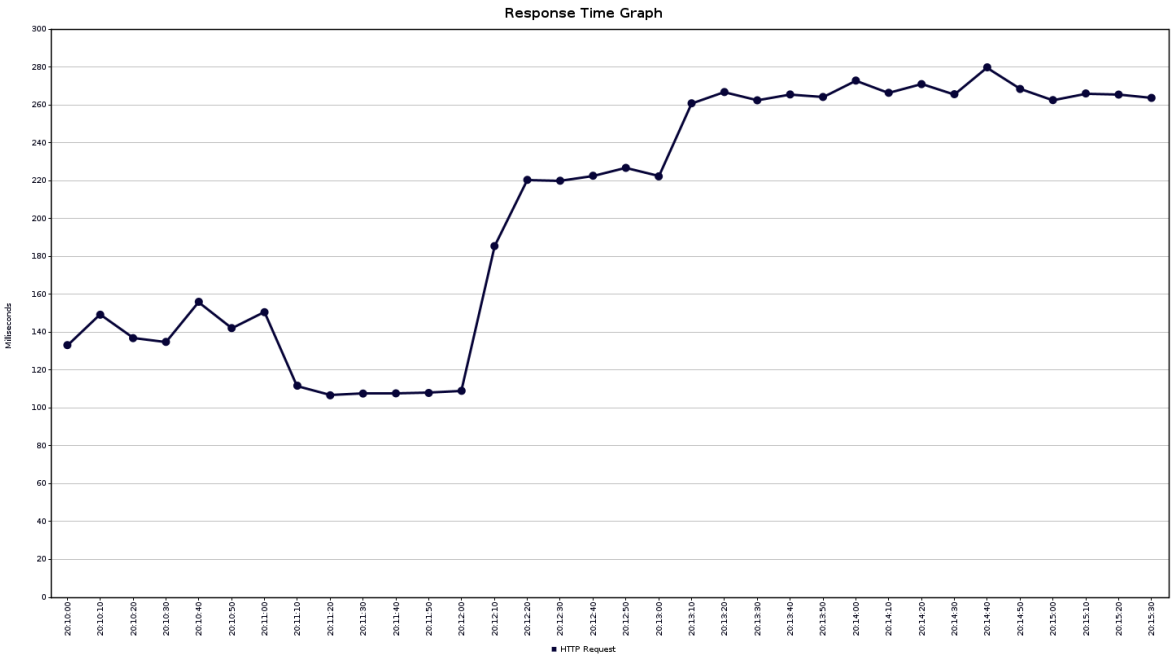
- Nombre de thread : 10
- Nombre de requêtes par secondes :

Nombres de requêtes au début	Nombres de requêtes à la fin	Durée en secondes
1	5	4
5	5	60
5	10	4
10	10	60
10	100	10
100	100	60
100	500	10
500	500	60
500	800	15
800	800	60

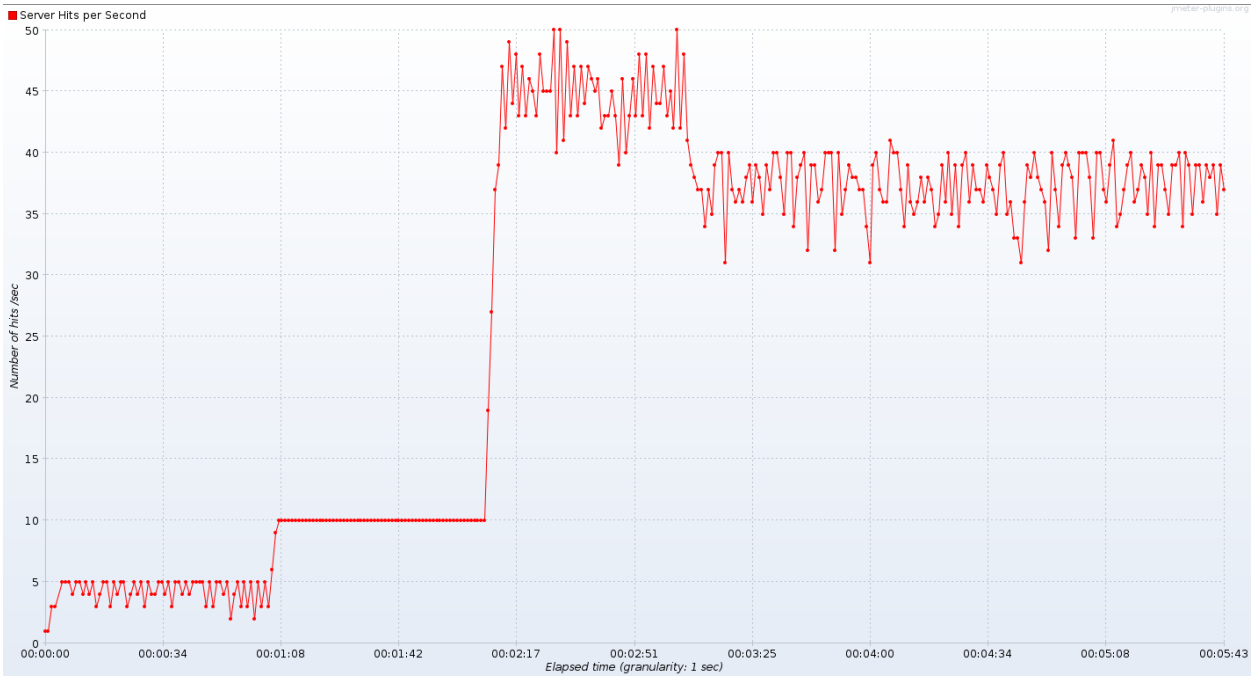
HYPOTHESE DE DEVELOPPEMENT

Voici les résultats obtenus sur l'API pour ces paramètres :

- Graphique sur le temps de réponse de l'API :



- Réponse des requêtes faites sur le serveur par seconde :



HYPOTHESE DE DEVELOPPEMENT

En prenant en compte que ces tests ont été faits sur une machine avec 8 cœurs et 32Go de RAM, on peut voir que l'API répond correctement tant que le nombre d'utilisateurs ne dépassent pas 10 utilisateurs, au-delà l'API ne répond plus sous les 200 millisecondes et ne dépasse pas 50 requêtes par secondes. On est alors très loin de l'objectif d'une réponse sous 200 millisecondes pour 800 requêtes par secondes.
On pourra retrouver l'ensemble des résultats de lancement des requêtes dans le fichier « Annexe_1-Resultats.csv ».
Pour permettre de répondre favorablement aux exigences sur cette API, plusieurs critères seront à mettre en place :

- La machine sur laquelle sera déposé l'API devra avoir des caractéristiques supérieures au capacité actuelle en local
- La mise en place d'un Load Balancing sur l'API Emergency permettra de ne pas surcharger les demandes et permettra d'obtenir des réponses plus rapides de celle-ci.
- Les micro services Hospital et BedAvailable ne doivent pas impacter les performances et doivent donc également avoir une infrastructure avec un Load Balancing et une machine avec des caractéristiques techniques conséquentes pour répondre rapidement sans être impacté par le nombre de requêtes parvenant à ces micro services.