MedHead

Reporting de la PoC

MedHead

Reporting de PoC

Historique des révisions :

Version	Description	Auteur	Date
1.0	La première version	Samir Guemri	29/02/2024

29/02/2024
MedHead
Nice, France

Ce document est un reporting qui porte sur le projet de MedHead visant à faire adhérer les parties prenantes au projet de développement d'un système d'intervention d'urgence en temps réel.

Ce document permet de synthétiser et de justifier les résultats de la PoC afin de prouver sa conformité avec les exigences business et techniques.

Ce document inclut :

- Une justification des technologies utilisées.
- Une justification du respect des normes et des principes.
- Les résultats du bon fonctionnement de la PoC.
- Les enseignements tirés du processus de développement du projet.

Ce reporting présente des éléments de haut niveau, n’entrant pas dans les détails.

Samir Guemri

Sommaire

I. Introduction

1. Présentation du consortium

MedHead est un regroupement de grandes institutions médicales œuvrant au sein du système de santé britannique et assujetti à la réglementation et aux directives locales (NHS). Les organisations membres du consortium utilisent actuellement une grande variété de technologies et d'appareils. Ils souhaitent une nouvelle plateforme pour unifier leurs pratiques. La technologie Java est pour eux un socle technique fiable pour ce projet.

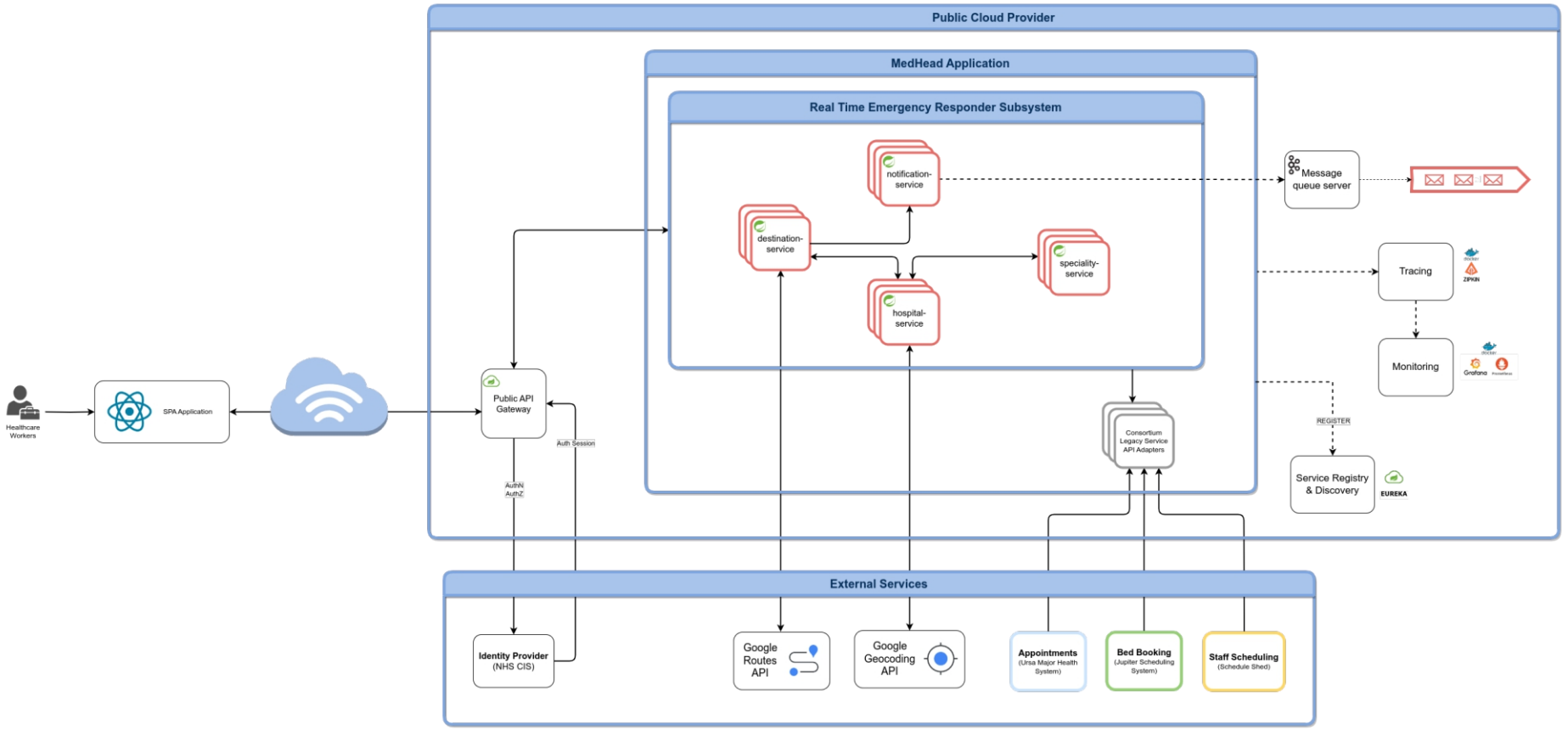
2. Objectifs business

La mise en œuvre d'une PoC pour le système d'intervention d'urgence en temps réel par l'équipe d'architecture métier permet de renforcer l'efficacité des traitements d'urgence, contribuant ainsi à sauver davantage de vies. Parallèlement, cette initiative vise à accroître la confiance des utilisateurs en démontrant la simplicité et l'efficacité du système, ce qui pourrait se traduire par une meilleure adoption de ce système.

II. Technologies utilisées

1. Architecture

L'adoption d'une **architecture microservices** pour ce projet offre une modularité qui facilite la maintenance et l'évolution de l'application. Cette approche permet de développe et de déployer des fonctionnalités spécifiques de manière indépendante, améliorant ainsi l'agilité et la résilience du système.
En segmentant l'application en un ensemble de services distincts, l'architecture microservices permet de dédier une équipe pour le développement de chaque service et facilite ainsi la complexité du développement d'une seule application monolithique. Enfin, cette architecture favorise une meilleure isolation des problèmes, réduisant les risques de défaillance systémique et simplifiant la prise en charge des bugs et les tests.



1. Architecture du système d'intervention d'urgence de MedHead

2. Développement backend

La technologie **Java** est choisi pour le développement de la partie backend de l'application.

2.1. Une technologie imposé par le consortium

Dans le document ***Exigences pour le développement d'une PoC*** , l'utilisation de la technologie Java est imposée par le consortium. Par ailleurs, le choix du framework Java Spring est justifié pour sa capacité à faciliter le développement rapide et efficace d'applications Java, particulièrement adaptées à une architecture microservices. Il offre une vaste gamme de modules et d'outils qui simplifient la configuration, le développement et le déploiement d'applications robustes, tout en assurant une intégration et une gestion aisées des dépendances, ce qui est crucial pour maintenir la flexibilité et l'évolutivité du système.

2.2. Une technologie moderne et robuste

En plus d'être la technologie imposée, Java est une option robuste et moderne pour le développement d'applications, parfaitement adaptée pour une architecture microservice :

- **Fiabilité** : Java est un langage de programmation mature et éprouvé, largement utilisé dans le développement d'applications d'entreprise. Sa robustesse, sa gestion d'exceptions et sa fiabilité en font un choix idéal pour les applications critiques comme les systèmes d'intervention d'urgence.
- **Performance** : Avec des améliorations constantes dans ses versions, Java offre des performances optimales qui sont essentielles pour le traitement en temps réel des situations d'urgence.
- **Sécurité** : Java fournit un cadre de sécurité robuste, essentiel pour protéger les informations sensibles traitées par le système d'intervention d'urgence. La gestion de la sécurité dans Java aide à prévenir les attaques externes et à assurer l'intégrité des données des utilisateurs.

En conclusion, le choix de la technologie Java pour développer la PoC s'aligne avec les exigences du projet d'offrir une solution robuste, sécurisée, performante et évolutive, tout en bénéficiant d'un large soutien et d'un écosystème riche.

3. Développement frontend

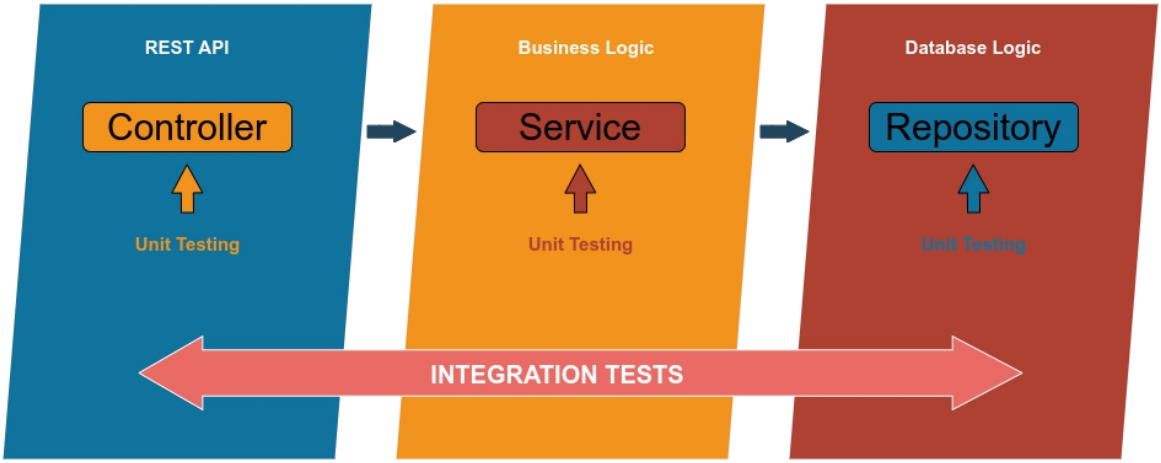
Le framework choisit pour le développement de l'interface graphique de l'application (parmi la liste imposé par le consortium) est **React**. Ce choix peut être justifié par :

- sa capacité à construire des interfaces utilisateur dynamiques et réactives, essentielles pour l'interaction en temps réel avec les API des microservices.
- avec son architecture basée sur des composants, React permet de créer des interfaces riches et modulaires, facilitant ainsi la maintenance et l'évolutivité.
- sa vaste communauté et son écosystème riche en bibliothèques et outils contribuent à accélérer le développement et à garantir une intégration fluide avec les technologies back-end.

4. Tests et validation

Dans le cadre de l'assurance qualité du développement et de la validation de la fiabilité des artefacts, l'application a été soumise à un protocole d'évaluation comprenant des tests unitaires, d'intégration, ainsi que des tests de charge et de performance. Les technologies sélectionnées pour ce processus sont les suivantes :

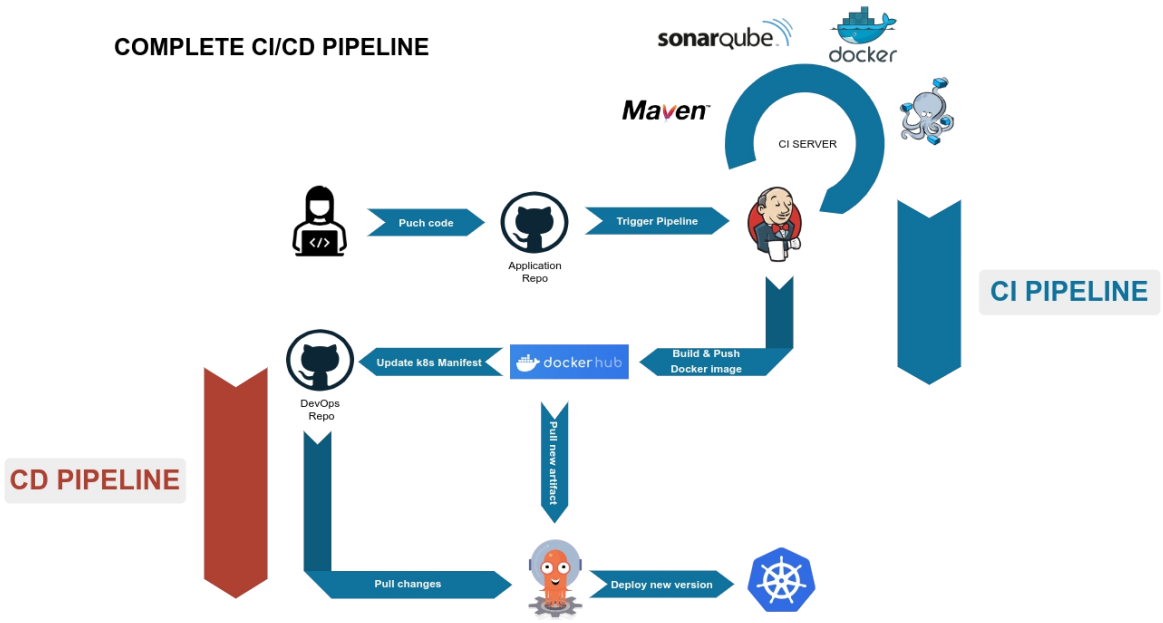
- **JUnit** : Il s'agit d'un cadre de test unitaire pour Java, conçu pour effectuer une vérification exhaustive des différentes composantes du code de manière isolée, ce qui est indispensable pour l'identification et la résolution rapide des anomalies.
- **JMock** : Ce framework complète les capacités de JUnit en simplifiant le mock d'objets, un aspect fondamental pour évaluer les interactions entre les composants sans dépendre des implémentations externes.
- **Testcontainers** : Cette technologie fournit un environnement de test intégré et reproductible, facilitant le déploiement de conteneurs Docker pour les bases de données ou tout autre service externe nécessaire lors des tests. Cela garantit le bon fonctionnement de l'application dans un cadre qui s’approche d’un environnement de production.
- **JMeter** : Utilisé pour les tests de charge et de performance, cet outil permet de simuler des scénarios d'usage réel et d'évaluer leur impact sur l'application, assurant ainsi que le système satisfait aux critères de performance et d'évolutivité exigés.



2. Illustration des tests unitaire et d'intégration

5. CI/CD

Nous avons choisi Jenkins comme outil d'intégration continue pour ce projet pour sa flexibilité, sa robustesse et son adaptabilité à divers environnements de développement. Jenkins automatise les phases de build, de test et de déploiement du code, assurant une intégration et une livraison continues efficaces, ce qui est essentiel pour maintenir la qualité et la stabilité de l'application dans un cycle de développement agile.



3. Pipeline CI/CD

III. Normes et principes

1. Normes & réglementation

Pour respecter les normes du Règlement Général sur la Protection des Données, le projet adhère aux exigences RGPD :

1.1. Tri des données

Avant la migration vers la nouvelle architecture :

- Les données collectées seront analysées et triées.
- Une purge de données sera faite

1.2. Droits des utilisateurs

Depuis un portail utilisateur:

- Les utilisateurs seront informés de manière claire et précise sur la manière dont leurs données sont collectées, traitées et utilisées.
- Le patient doit donner son consentement explicite avant de collecter, traiter ou utiliser ses données.
- A tout moment, le patient aura le droit de demander la suppression de ces données personnelles.

1.3. Sécurité des données

La conception de l'architecture logiciel du nouveau système prendra en compte la sécurisation et la protection des données :

- Les échanges de données avec des services externes seront sécurisés.
- Les bases de données seront cryptées.
- Un firewall sera mis en place.
- La communication entre les différents services de l'application sera sécurisée.

2. Principes de l'architecture

Le document ***Principes de l'architecture*** a définit l'ensemble de principes architecturaux à privilégier par domaine. Les principes de l'architecture informatique sont respectés.

Principe	Application
Principe B1 : Continuité des activités des systèmes critiques pour les patients	Une stratégie de migration sera mise en place afin de s'assurer de la continuité des activités des systèmes critiques.
Principe B2 : Clarté grâce à une	Le système est développé selon les principes

séparation fine des préoccupations	d'une architecture microservice afin de garantir une séparation des responsabilités. L'application est un ensemble de petits services indépendants qui communiquent entre eux via des APIs
Principe B3 : Intégration et livraison continues	Une pipeline CI/CD est mise en place afin de garantir une intégration continue des petites améliorations et une mise en production facile et automatique.
Principe B4 : Tests automatisés précoces, complets et appropriés	Des tests unitaires et d'intégrations sont développés et exécutés d'une manière automatique après chaque nouvelle fonctionnalité intégrée.
Principe B5 : Sécurité de type « shift-left »	La méthodologie TDD est privilégiée pour s'assurer de la qualité du code développé.
Principe B6 : Possibilité d'extension grâce à des fonctionnalités pilotées par les événements	Un service de notification est développé afin de permettre aux hôpitaux de souscrire au broker pour être notifié de toute réservation de lit.

PoC et les principes MedHead

IV. Résultats de la PoC

1. Exigences du Consortium

Le consortium avance quelques critères d'acceptation pour valider la PoC :

Indicateur	Valeur Cible
Acheminement vers l'hôpital compétent le plus proche	+90 %
Temps moyen de traitement d'une urgence	12 minutes
Temps de réponse pour une charge de travail de 800 requêtes/sec	< 200ms

Le consortium exige aussi le respect des **normes imposées** ainsi que le respect du **délai imparti**.

2. Résultats constatées

Pour effectuer les tests de performance et de charge, nous avons utilisés l'outil JMeter. Cet outil nous a permis de générer des rapports des résultats de nos tests sous forme de graphe et de tableau de bord.

2.1. Test de performance

Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**1**	**0**	**0.00%**	**259.00**	**259**	**259**	**259.00**	**259.00**	**259.00**	**259.00**	**3.86**	**4.58**	**1.20**
searchDestination	1	0	0.00%	259.00	259	259	259.00	259.00	259.00	259.00	3.86	4.58	1.20

Résultat de test de performance

2.2. Tests de charge

Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**800**	**0**	**0.00%**	**3945.93**	**1864**	**7199**	**3759.00**	**5678.80**	**5903.85**	**6604.28**	**108.17**	**128.34**	**33.70**
searchDestination	800	0	0.00%	3945.93	1864	7199	3759.00	5678.80	5903.85	6604.28	108.17	128.34	33.70

Résultat des tests de charge

Compte tenue de l'environnement de test et des performances de la machine sur la quelle les tests ont été effectués ainsi que les résultats de test de la PoC ci dessus, le projet de développer un système d'intervention d'urgence en temps réel est un projet réalisable tout en respectant les contraintes et les exigences du consortium.

V. Conclusion

Ce PoC nous a permis de démontrer la faisabilité technique du projet. Les résultats obtenus confirment que les objectifs fonctionnels sont atteignables tout en respectant les normes et les principes d'architecture.