MedHead Consortium

Rapport d'évaluation du POC

Système d'intervention d'urgence

Objet : *Création d'une plateforme qui atténue les risques associés au traitement des demandes de lits d'hôpitaux dans les situations d'urgence. Ce POC démontre une version primitive de la fonctionnalité attendue du produit final.*

Auteur

Nom / e-mail	**Rôle**
Andrej ILIEVSKI / andrej.ilievski@medhead.com	Architecte Logiciel

SOMMAIRE

1. INTRODUCTION

L'évolution rapide de la technologie a ouvert la voie à des solutions innovantes dans divers secteurs, y compris celui des soins de santé. La démonstration de faisabilité présentée dans ce document témoigne de cette évolution. Conçue pour combler le fossé de la communication entre les intervenants d'urgence et les hôpitaux, cette démonstration de faisabilité s'appuie sur des architectures logicielles et des technologies modernes pour assurer une communication en temps réel, efficace et fiable.

L'objectif principal de ce PoC est de démontrer la faisabilité de l'utilisation d'une approche basée sur les microservices, en particulier avec Java Spring Boot, pour créer un système évolutif et facile à maintenir. Le système comprend des fonctionnalités allant de la gestion des hôpitaux au chat en temps réel, garantissant que les intervenants d'urgence peuvent rapidement trouver les hôpitaux les plus proches et communiquer avec eux, optimisant ainsi les soins aux patients.

En outre, le PoC intègre des technologies avancées telles que Redis pour la messagerie en temps réel et la mise en cache, ce qui garantit que le système n'est pas seulement fonctionnel, mais qu'il fonctionne également de manière optimale, même en cas de charge importante.

Le présent document se penche sur les résultats obtenus par le PoC, en soulignant ses réussites, ses domaines d'amélioration et le potentiel de sa mise en œuvre future à plus grande échelle.

2. SYNTHÈSE

Le secteur des services de santé a constamment besoin d'avancées technologiques pour améliorer les soins aux patients, rationaliser les processus et renforcer la communication entre les différentes entités. Consciente de ce besoin, notre équipe s'est lancée dans le développement d'une preuve de concept (PoC) qui répond à l'un des défis critiques auxquels sont confrontés les intervenants d'urgence : la communication en temps réel avec les hôpitaux.

L'objectif principal de ce PoC était de démontrer la viabilité d'un système basé sur des microservices qui facilite la communication instantanée entre les intervenants d'urgence et les hôpitaux. En s'appuyant sur Java Spring Boot et en intégrant Redis, le système visait à fournir une solution évolutive, maintenable et performante.

Les principales caractéristiques sont les suivantes :

1. **Système de gestion des hôpitaux** : Il permet d'ajouter, de mettre à jour et de supprimer des dossiers hospitaliers, garantissant ainsi aux intervenants d'urgence l'accès aux données les plus récentes.

2. **Chat en temps réel** : Basée sur Redis, cette fonction permet aux intervenants d'urgence de communiquer instantanément avec les hôpitaux, ce qui réduit les délais d'intervention et améliore les résultats pour les patients.

3. **Évolutivité et performances** : L'architecture microservices garantit que chaque composant du système peut évoluer indépendamment, répondant ainsi aux exigences d'une base d'utilisateurs croissante.

En résumé, le PoC a permis de démontrer la faisabilité du système proposé. Les tests ont montré que le système pouvait gérer efficacement les communications en temps réel, avec un temps de latence minimal. Le système de gestion s'est avéré robuste, permettant des mises à jour et une récupération rapides des données pour les hôpitaux, les intervenants d'urgence et les patients.

En conclusion, les résultats obtenus par le PoC sont prometteurs et indiquent qu'avec un développement et un perfectionnement supplémentaires, le système peut être mis en œuvre à plus grande échelle, apportant des améliorations significatives au processus d'intervention d'urgence dans le secteur des soins de santé.

3. MÉTHODOLOGIE

Le chapitre sur la méthodologie décrit l'approche systématique adoptée pour atteindre les objectifs de la validation du concept. Étant donné la nature technique du projet, il est essentiel de détailler les étapes, les outils et les stratégies utilisés. Ce chapitre est divisé en deux parties : la première se concentre sur la conception et le développement du système, tandis que la seconde se penche sur les processus de test et de validation.

3.1. Conception et développement du système

La phase de conception et de développement du PoC a été cruciale pour garantir que le système réponde aux exigences et aux attentes des utilisateurs prévus : les intervenants d'urgence et les hôpitaux. Cette phase a été caractérisée par une série de processus itératifs, de la conceptualisation initiale à la mise en œuvre finale.

3.1.1. Analyse des besoins

Le fondement de tout projet réussi repose sur une analyse approfondie et complète des besoins. Cette phase n'a pas seulement consisté à comprendre ce que le système devait faire, mais aussi pourquoi il devait le faire et comment il s'intégrerait dans l'écosystème plus large de l'intervention d'urgence et de la gestion hospitalière.

Entretiens avec les parties prenantes

Nous avons commencé l'analyse des besoins en menant des entretiens approfondis avec les principales parties prenantes. Il s'agissait notamment d'intervenants en cas d'urgence, d'administrateurs d'hôpitaux et de personnel informatique. Ces discussions ont fourni des informations précieuses sur les défis quotidiens auxquels chaque groupe est confronté et sur les solutions potentielles qu'il envisage.

Le point de vue des intervenants d'urgence

Les intervenants d'urgence nous ont fait part de l'importance cruciale du temps dans leurs activités. Chaque seconde compte, et l'accès immédiat à des informations en temps réel sur la disponibilité des hôpitaux peut faire la différence entre la vie et la mort. Ils ont exprimé le besoin d'un système capable de fournir des mises à jour instantanées, réduisant ainsi le temps nécessaire à la prise de décision en cas d'urgence.

Le point de vue des hôpitaux

Les administrateurs et le personnel des hôpitaux ont souligné les défis auxquels ils sont confrontés dans la gestion et la communication de la disponibilité des lits, en particulier pendant les périodes de pointe ou en cas d'urgence. Ils ont souligné l'importance d'un système centralisé où ils pourraient mettre à jour l'état des lits en temps réel, garantissant ainsi que les intervenants d'urgence disposent de l'information la plus précise possible. En outre, ils ont exprimé le besoin d'un système capable de traiter plusieurs requêtes simultanément sans aucun décalage, étant donné le rythme soutenu de leurs opérations.

Exigences techniques

Les informaticiens ont donné un aperçu des aspects techniques. Ils ont souligné l'importance d'un système évolutif, compte tenu de la nature imprévisible des situations d'urgence et de la possibilité de pics d'utilisation soudains. La sécurité était une autre préoccupation importante, surtout si l'on considère la nature sensible des données médicales. Le système devait être robuste face aux cybermenaces potentielles et garantir l'intégrité des données à tout moment.

Exigences fonctionnelles et non fonctionnelles

À l'issue de ces discussions, nous avons classé les exigences en deux catégories : les exigences fonctionnelles et les exigences non fonctionnelles. Les exigences fonctionnelles détaillent les caractéristiques et les fonctionnalités que le système doit avoir, comme le chat en temps réel, la gestion des données hospitalières, etc. Les exigences non fonctionnelles, quant à elles, portaient sur les performances du système, la sécurité, l'évolutivité et l'expérience de l'utilisateur.

Feedback loops

Enfin, un mécanisme de retour d'information a été mis en place pour que les parties prenantes puissent apporter leur contribution au fur et à mesure du développement du système, afin que le produit final corresponde à leurs attentes et à leurs besoins.

En substance, la phase d'analyse des besoins ne consistait pas seulement à dresser une liste de caractéristiques, mais aussi à comprendre les besoins et les défis les plus profonds des utilisateurs finaux. Cette approche holistique a permis de s'assurer que le PoC n'était pas seulement solide sur le plan technique, mais aussi pertinent sur le plan contextuel et centré sur l'utilisateur.

3.1.2. Sélection des outils et des technologies

Sur la base des exigences, l'étape suivante a consisté à sélectionner les outils et les technologies appropriés. Java Spring Boot a été choisi comme cadre de développement principal en raison de sa robustesse, de son évolutivité et du soutien important de la communauté. Redis, une structure de stockage de données en mémoire très performante, a été intégrée pour faciliter la fonctionnalité de chat en temps réel, en raison de son efficacité dans le traitement des données en temps réel.

3.1.3. Architecture microservices

La décision d'adopter une architecture microservices a été influencée par le besoin d'évolutivité et de maintenabilité. En décomposant le système en services plus petits et indépendants, on s'est assuré que chaque composant pouvait être développé, déployé et mis à l'échelle de manière indépendante. Cette architecture a également permis une meilleure allocation des ressources, en garantissant que chaque service dispose de la puissance de calcul et de la mémoire nécessaires.

3.1.4. Développement du système de gestion hospitalière

Le système de gestion hospitalière a été le premier composant à être développé. Il sert de colonne vertébrale au PoC, en stockant et en gérant les données relatives aux hôpitaux. Le système permet d'ajouter, de mettre à jour et de supprimer des dossiers hospitaliers. Une attention particulière a été accordée à l'interface utilisateur, en veillant à ce qu'elle soit intuitive et conviviale. Le système a été conçu pour traiter un grand volume de données, garantissant des temps de récupération et de mise à jour rapides.

3.1.5. Intégration du chat en temps réel

La fonctionnalité de chat en temps réel, alimentée par Redis, a ensuite été intégrée. Cette fonction était cruciale pour combler le fossé de communication entre les intervenants d'urgence et les hôpitaux. Le système de chat a été conçu pour être léger et garantir une latence minimale, ce qui est crucial dans les situations d'urgence.

3.1.6. Mesures de sécurité

Compte tenu de la nature sensible des données traitées, la sécurité était une priorité absolue. Des mesures ont été mises en place pour garantir l'intégrité des données et les protéger contre les accès non autorisés. Le système a été configuré pour traiter les demandes en toute sécurité, en accordant une attention particulière aux méthodes POST et UPDATE. En outre, des configurations CORS (Cross-Origin Resource Sharing) ont été mises en place pour garantir que le système puisse interagir en toute sécurité avec différentes origines, en particulier pendant la phase de test.

3.1.7. Intégration et déploiement continus

Pour garantir que le processus de développement se déroule sans heurts et que les modifications apportées n'introduisent pas de nouveaux problèmes, un pipeline d'intégration et de déploiement continus (CI/CD) a été mis en place. Cela a permis de tester automatiquement le nouveau code et de s'assurer que le système était toujours en état d'être déployé.

En conclusion, la phase de conception et de développement a été caractérisée par une approche méticuleuse, garantissant que chaque composant du système était robuste, évolutif et répondait aux besoins de ses utilisateurs. La nature itérative du processus a permis un retour d'information continu, garantissant que tout problème soit traité rapidement.

3.2. Approche du développement

Compte tenu de la complexité et du besoin d'évolutivité, une architecture microservices a été choisie pour le développement du PoC. Cette approche a permis un développement, un déploiement et une mise à l'échelle indépendants de chaque service, garantissant ainsi la flexibilité et la robustesse.

Développement itératif : Le processus de développement a été itératif, chaque itération se concentrant sur un ensemble spécifique de fonctionnalités. Cela a permis un retour d'information et des ajustements continus, garantissant que le produit final répondait aux exigences et aux attentes.

Développement piloté par les tests (TDD) : Avant de mettre en œuvre une fonctionnalité, des tests ont été rédigés pour définir le comportement attendu. Cela a permis non seulement de s'assurer que le code était correct, mais aussi qu'il le restait au fur et à mesure des modifications apportées.

3.3. Stratégie d'intégration

Intégration continue (CI) : Un pipeline d'intégration continue a été mis en place pour construire et tester automatiquement l'application à chaque modification. Cela a permis d'identifier et de résoudre rapidement tout problème d'intégration.

Découverte de services : Avec plusieurs microservices en jeu, la découverte de services est devenue cruciale. Des outils comme Eureka de la suite Spring Cloud ont été envisagés pour gérer et équilibrer les charges entre les instances de service.

Gateway API : Le gateway Spring Cloud a été utilisée comme point d'entrée principal pour toutes les demandes des clients. Elle gère le routage, l'équilibrage des charges et fournit une couche de sécurité.

3.4. Gestion des données

Compte tenu de la nature de l'application, la cohérence et la disponibilité des données étaient d'une importance capitale.

Sélection de la base de données : La base de données en mémoire H2 a été utilisée pour le développement en raison de sa légèreté et de ses capacités d'accès rapide aux données. Redis, un magasin de clés-valeurs en mémoire très performant, a été utilisé pour la mise en cache et les fonctionnalités en temps réel.

Synchronisation des données : De nombreux services accédant aux données et les modifiant, des mécanismes ont été mis en place pour assurer la cohérence des données. Des architectures événementielles, utilisant des outils comme Apache Kafka, ont été envisagées pour propager les changements de données entre les services.

3.5. Évolutivité et performances

Évolution horizontale : Les microservices prennent intrinsèquement en charge la mise à l'échelle horizontale. Au fur et à mesure que la demande pour un service particulier augmentait, davantage d'instances de ce service pouvaient être activées pour gérer la charge.

Tests de performance : Des outils tels que JMeter et Gatling ont été envisagés pour les tests de performance. Ces outils ont permis de simuler des charges élevées sur l'application, afin de s'assurer qu'elle pouvait répondre aux demandes du monde réel.

Mise en cache : Redis a été utilisé non seulement pour ses capacités en temps réel, mais aussi pour mettre en cache les données fréquemment consultées, ce qui a permis de réduire la charge sur les bases de données et d'améliorer les temps de réponse.

3.6. Feedback et itération

Le feedback a joué un rôle crucial dans le processus de développement. Après chaque itération, les fonctionnalités développées ont été présentées aux parties prenantes et leurs commentaires ont été intégrés dans les itérations suivantes. Cela a permis de s'assurer que l'application évoluait en fonction des besoins et des attentes des utilisateurs.

4. MISE EN ŒUVRE

La phase de mise en œuvre est celle où le cadre conceptuel, dérivé de notre analyse approfondie des besoins, a été transformé en un système tangible et fonctionnel. Cette phase a été caractérisée par une planification méticuleuse, un développement itératif et des tests rigoureux. Nous nous penchons ici sur les spécificités des trois microservices qui constituent le cœur de notre système, ainsi que sur la pile technologique qui les alimente.

4.1. L'architecture microservices

La décision d'adopter une architecture de microservices a été motivée par le besoin d'évolutivité, de modularité et de facilité de maintenance. Les microservices, de par leur conception, sont des unités indépendantes qui communiquent entre elles par l'intermédiaire d'un réseau, généralement HTTP. Cette architecture permet à chaque service d'être développé, déployé et mis à l'échelle de manière indépendante, ce qui lui confère souplesse et résilience.

Les avantages de cette approche sont les suivants :

- **Évolutivité :** Chaque microservice peut être mis à l'échelle de manière indépendante en fonction de sa charge de travail.
- **Flexibilité :** Les microservices peuvent être développés en utilisant la technologie la mieux adaptée à leur fonctionnalité spécifique.
- **Résilience :** La défaillance d'un microservice n'entraîne pas nécessairement l'effondrement de l'ensemble du système.
- **Développement et déploiement rapides :** Les équipes peuvent travailler simultanément sur différents microservices, ce qui permet d'accélérer les cycles de développement et d'augmenter la fréquence des mises à jour.

4.1.1. Microservice hôpital

Le microservice Hôpital, comme nous l'avons vu précédemment, sert de plaque tournante pour la gestion et la diffusion d'informations en temps réel sur la disponibilité des lits d'hôpitaux. Construit à l'aide de Java Spring Boot, ce microservice offre des points d'extrémité pour les opérations CRUD liées aux données hospitalières. Il intègre également Redis pour la mise en cache, ce qui garantit des temps de réponse rapides même en cas de forte charge. La fonctionnalité de chat en temps réel, alimentée par Redis, facilite la communication instantanée entre les hôpitaux et les intervenants d'urgence.

4.1.2. Microservice patient

Le microservice Patient est essentiel pour la gestion des données des patients, y compris leurs antécédents médicaux, leur état de santé actuel et d'autres détails pertinents. Ce microservice garantit que les intervenants d'urgence disposent de toutes les informations nécessaires sur un patient, ce qui leur permet de prendre des décisions en connaissance de cause. Comme le microservice hospitalier, il est construit sur le framework Java Spring Boot, ce qui garantit la cohérence et la facilité d'intégration dans l'ensemble du système.

4.1.3. Microservice emergency responder

Le microservice "Emergency Responder" est conçu pour répondre aux besoins spécifiques des intervenants en cas d'urgence. Il offre des fonctionnalités telles que le suivi de la localisation en temps réel, des canaux de communication avec les hôpitaux et l'accès aux données des patients. Ce microservice contribue à garantir que les intervenants d'urgence peuvent travailler efficacement, en disposant de toutes les informations nécessaires.

4.2. Gateway API

Compte tenu de l'architecture microservices de notre système, la passerelle API joue un rôle crucial en assurant une communication transparente entre les différents services. Elle sert de point d'entrée unique pour toutes les demandes des clients et les dirige vers le microservice approprié. Construite à l'aide de Spring Cloud Gateway, elle gère également des questions telles que l'équilibrage de la charge, l'authentification et CORS, garantissant ainsi l'évolutivité et la sécurité de notre système.

4.3. Stack technologique

Le stack technologique choisie pour ce projet était un mélange de cadres backend robustes, de bases de données efficaces et d'outils frontaux modernes. Cette combinaison a permis d'obtenir une application transparente, évolutive et réactive. Vous trouverez ci-dessous une description détaillée des technologies utilisées :

Composant	Technologie/Outil	Objectif/Description
Frontend	Next.js (TypeScript)	Fournit une expérience réactive et conviviale aux utilisateurs finaux.
Backend	Java Spring Boot	Framework principal pour tous les microservices, offrant évolutivité, robustesse et un soutien important de la communauté.
Base de données	H2 Database	Base de données relationnelle en mémoire utilisée à des fins de développement et de test.
	Redis	Utilisé pour la mise en cache et les fonctionnalités en temps réel, telles que la fonction de chat.
Communication	WebSocket avec STOMP	Assure une communication bidirectionnelle à faible latence pour les fonctions en temps réel.
Securité	Spring Security	Offre des fonctions de sécurité complètes, garantissant l'intégrité des données et la protection contre les menaces potentielles.
Gestion de l'API	Spring Cloud Gateway	Il sert de point d'entrée principal pour les demandes des clients et gère l'équilibrage de la charge, l'authentification et les CORS.

Ce stack a été méticuleusement choisi pour que chaque composant du système soit optimisé pour son rôle spécifique, tout en assurant une intégration et une communication transparentes entre les différentes parties.

En conclusion, la phase de mise en œuvre s'est caractérisée par un mélange de technologies de pointe et de bonnes pratiques. En nous appuyant sur une architecture de microservices, nous avons veillé à ce que chaque composant de notre système soit modulaire, évolutif et puisse fonctionner indépendamment, tout en faisant partie d'un ensemble cohérent. Cette approche a non seulement facilité un développement rapide, mais elle a également permis d'intégrer de manière transparente toute extension ou modification future.

5. RÉSULTATS ET CONCLUSION

Le développement et l'essai de la preuve de concept du système d'intervention médicale d'urgence (PoC) ont donné lieu à une série de résultats significatifs. Ces résultats sont essentiels pour comprendre les capacités du système, son potentiel d'application dans le monde réel et les domaines à améliorer. Les résultats sont classés en trois catégories principales : Les réalisations fonctionnelles, les mesures de performance et le retour d'information de l'utilisateur.

5.1. Réalisations fonctionnelles

Le PoC a été conçu pour répondre à des exigences fonctionnelles spécifiques, et ses résultats dans ce domaine sont particulièrement remarquables.

5.1.1. Communication en temps réel

La capacité du système à faciliter la communication instantanée a été rendue possible par l'intégration de Redis. Cette fonctionnalité de chat en temps réel a comblé le fossé de la communication entre les intervenants d'urgence et les hôpitaux, en garantissant que les informations critiques soient échangées sans délai.

5.1.2. Intégration des microservices

L'architecture du système s'articule autour de plusieurs microservices, chacun répondant à une fonctionnalité spécifique :

Microservice d'hôpital : Il fournit une vue d'ensemble détaillée des hôpitaux, y compris leur emplacement, la disponibilité des lits et les domaines de spécialisation. Il est devenu un outil essentiel pour permettre aux intervenants d'urgence de prendre des décisions éclairées sur l'acheminement des patients.

Microservice Patient : Une base de données complète des patients, comprenant leurs antécédents médicaux et leur état actuel, a permis aux hôpitaux d'être bien préparés à fournir des soins immédiats dès l'arrivée d'un patient.

Microservice pour les intervenants d'urgence : Ce microservice tient un registre de tous les intervenants d'urgence actifs, de leur localisation actuelle et de leurs domaines de compétence, ce qui garantit une répartition efficace en fonction de la nature des urgences.

5.1.3. Utilisation de la gateway API

La passerelle Spring Cloud a joué un rôle essentiel dans la fonctionnalité du système. Elle a veillé à ce que toutes les demandes entrantes soient correctement acheminées vers les microservices concernés, en fournissant un point d'entrée unifié et sécurisé et en simplifiant les interactions externes du système.

5.2. Mesures de performance

Le PoC ne visait pas seulement à atteindre des objectifs fonctionnels ; il s'agissait également de s'assurer que ces fonctions étaient exécutées de manière optimale.

5.2.1. Évolutivité

Grâce à l'architecture microservices, le système a fait preuve d'un haut degré d'évolutivité. Au fur et à mesure que la demande ou la charge augmentait, de nouvelles instances de services spécifiques pouvaient être lancées sans affecter les performances globales du système.

5.2.2. Réponse rapide

La combinaison de Redis pour les opérations en temps réel et de la base de données en mémoire H2 a permis d'obtenir des temps de réponse impressionnants, un facteur essentiel dans les situations d'urgence.

5.2.3. Fiabilité du système

Tout au long de la phase d'essai, le système a maintenu un temps de fonctionnement élevé. L'architecture modulaire a permis de garantir que même si un composant rencontrait des problèmes, le système dans son ensemble resterait fonctionnel.

5.3. Conclusion

L'élaboration de la preuve de concept du système d'intervention médicale d'urgence a été à la fois stimulante et instructive. Lorsque nous réfléchissons au processus et aux résultats, plusieurs éléments clés se dégagent qui non seulement valident les efforts déployés, mais tracent également la voie à suivre.

5.3.1. Validation de la vision

L'objectif premier de ce PoC était de vérifier si le système envisagé pouvait apporter des améliorations tangibles au domaine de la réponse médicale d'urgence. Les résultats, tant en termes de réalisations fonctionnelles que d'indicateurs de performance, valident cette vision. La communication en temps réel, l'intégration transparente de divers microservices et les temps de réponse rapides sont autant d'éléments qui indiquent que le système peut réellement améliorer l'efficience et l'efficacité des interventions médicales d'urgence.

5.3.2. La puissance des microservices

L'une des caractéristiques marquantes de ce projet a été l'architecture de microservices. Elle a non seulement apporté au système l'évolutivité et la modularité, mais elle a également permis de développer, de tester et de déployer chaque composant de manière indépendante. Cette architecture a fait ses preuves dans le cadre de ce PoC et témoigne de son potentiel dans le développement de systèmes complexes à grande échelle.

5.3.3. Le chemin à parcourir

Bien que le PoC ait été un succès, il est essentiel de se rappeler que ce n'est qu'un début. Les idées et les commentaires recueillis au cours de cette phase constituent une feuille de route pour les développements futurs. L'intégration de fonctions telles que la visualisation de cartes, l'amélioration des fonctionnalités de chat et peut-être même l'incorporation d'analyses prédictives pilotées par l'IA ne sont que quelques-unes des possibilités passionnantes des prochaines phases.

En conclusion, le PoC du système de réponse médicale d'urgence a jeté des bases solides pour une solution transformatrice dans le domaine des urgences médicales. Il a démontré qu'il était possible de combler les lacunes en matière de communication, d'améliorer la coordination et, en fin de compte, de sauver davantage de vies. À mesure que nous avançons, les enseignements tirés de ce PoC guideront l'évolution du système, en veillant à ce qu'il reste à la pointe de l'innovation technologique et continue à avoir un impact significatif dans les scénarios du monde réel.