Livrables	Reporting POC
Auteur	Yoann VALERO - Architecte Logiciel
Projet	MedHead

Objet de ce document :

Ce document de reporting inclut les éléments suivants :

- La vision d'architecture du POC et la justification des choix technologiques
- Les résultats obtenu de la preuve de concept (POC)

Historique des révisions :

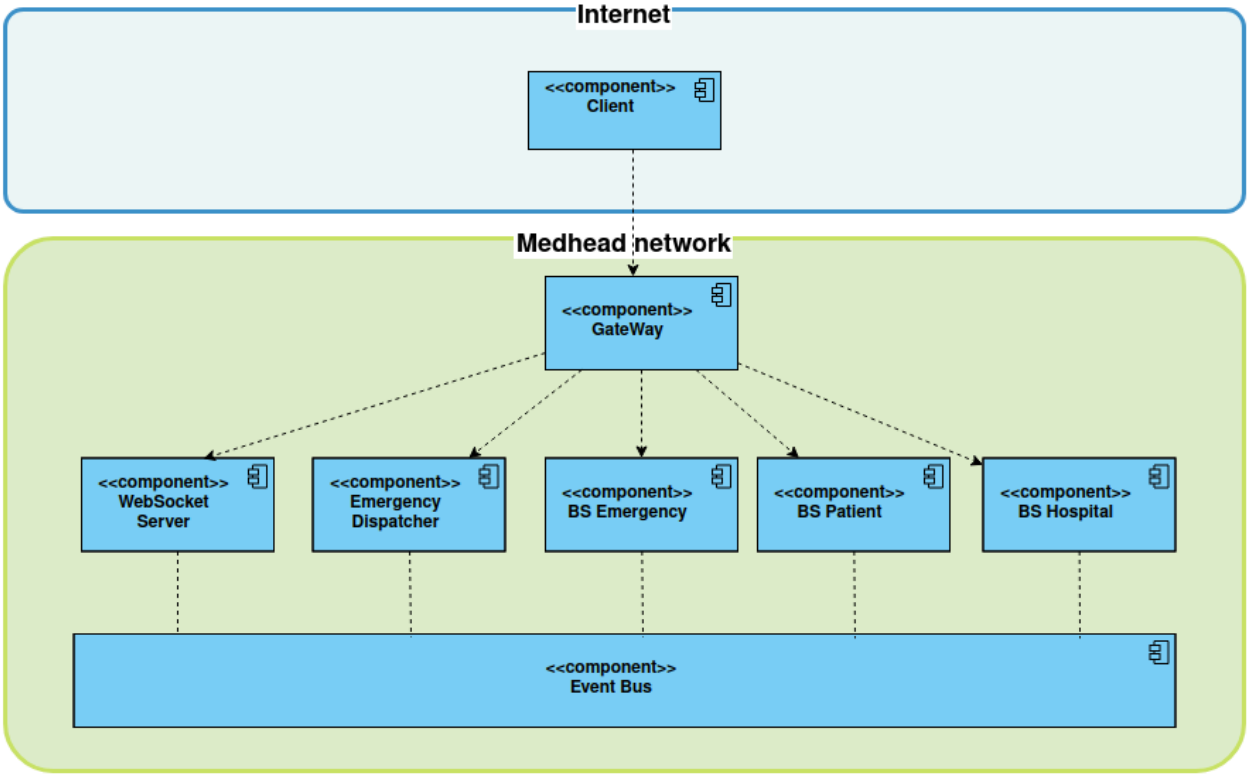
Date	Version	Commentaires
13/10/2023	0.01	Document fourni.

Sommaire

Vision d'architecture

L'architecture cible finale se décompose de la façon suivante :

Composant	Description
App Client	Application front pour accéder aux fonctionnalités MedHead
BS Patient	Micro-service de gestion des patients (données)
BS Hospital	Micro-service de gestion des hôpitaux (lits disponibles, spécialités, localisation)
BS Emergency	Micro-service de gestion des urgences (consultation, mise à jour)
Emergency Dispatcher	Micro-service permettant de dispatcher les urgences en fonction de leur spécificité (choix de l'hôpital le plus proche en fonction de la distance, des spécialités des hôpitaux, lits disponibles)
WebSocket Server	Serveur Websocket permettant de maintenir une connexion aux clients pour répandre les messages en temps réel.
Event Bus	Composant permettant la publication de messages et la souscription aux messages pour les différents micro-services



Technologies

Backend

Afin de satisfaire aux exigences, les services back-end (micro-services) ont été construits sur une base Java (v17). Le Framework Spring a été retenu pour sa forte popularité et sa capacité à répondre aux besoins des différents composants logiciels (Spring Boot principalement, mais d'autres dépendances à Spring peuvent exister en fonction des besoins de chaque micro-services).

Le service Emergency Dispatcher s'appuie sur une API tierce pour le calcul de la proximité d'une urgence par rapport aux hôpitaux : Distance Matrix API. Développée par Google, elle assure un très haut niveau de disponibilité permettant de répondre aux besoins de performance du projet.

L'architecture logicielle pour le système final sera orienté Event-Driven constitué de plusieurs micro-services pour répondre aux exigences du document de définition d'architecture fourni. Dans le cadre du POC, seulement une partie des composants seront développés afin de tester et répondre rapidement au document d'exigence fourni.

Frontend

Le module front-end a été construit via une technologie Javascript / Angular. Ce Framework a été retenu pour sa robustesse et son écosystème riche développé par Google. (https://angular.io/)
L'application finale devra être connectée à un serveur de WebSocket via la technologie NodeJS (lui-même à l'écoute des événements transitant dans le bus dédié) pour rendre l'affichage en temps réel et recharger les interfaces graphiques en fonction des événements soumis.
Cela permettra un confort pour les futurs utilisateurs et garantira l'exactitude des informations affichées.

Sécurité

Authentification

Les requêtes transitant entre les micro-services du système Medhead doivent être authentifiées via une clé api garantissant un accès sécurisé aux micro-service Spring-boot.

Pour l'architecture finale, une authentification OAuth devra être implémentée pour garantir un haut contrôle d'accès des utilisateurs et une gestion plus approfondie de leurs droits.

CORS

Le composant de configuration Spring-boot assure les contrôles d'origine dans le cadre du standard CORS. Cette approche permet d'éviter d'implémenter des listes de domaines autorisés côté backend et réduit le couplage front / back.

Règle de sécurité standards

Les composants développés sont exempts de failles de sécurité identifiées et plus particulièrement de celles relevées dans le Top 10 OWASP (Injection SQL, XSS …)

Proof Of Concept (POC)

Le document d'exigences pour le POC présente les éléments à fournir à cet effet. Les composants réalisés dans le cadre du POC sont les suivants :

Composant	Description
App Client	Application front pour accéder aux fonctionnalités MedHead.
BS Hospital	Micro-service de gestion des hôpitaux (lits disponibles, spécialités, localisation)
Emergency Dispatcher	Micro-service permettant de dispatcher les urgences en fonction de leur spécificité (choix de l'hôpital le plus proche en fonction de la distance, des spécialités des hôpitaux, lits disponibles)

Ils permettront de répondre aux exigences suivantes :

- Fournir une API Restful respectant les conditions suivantes
  - Technologie JAVA
  - l'API doit pouvoir s'inscrire dans une architecture microservice
- Fournir une interface graphique qui consomme l'API
  - Technologie cliente : Angular, React ou VueJS
- S'assurer que les données sont correctement protégées
- S'assurer que la PoC est entièrement validée avec des tests reflétant la pyramide de tests (tests unitaires, d'intégration, E2E, test de stress)
- Fournir une intégration et livraison continue (CI/CD)
- Le code est versionné à l'aide d'un workflow Git adapté
- Une documentation technique de la PoC pour son exécution et la ré-utilisation (test, installation, workflow)
- Fournir un document de reporting indiquant les résultats et enseignement de la Poc

CI / CD

La pipeline d'intégration et de livraison continue a été réalisée avec l'outil Github Action.
Le repository du code source héberge l'ensemble des micro-services.

Les étapes de la pipeline CI/CD se décomposent de la façon suivante lors de chaque push sur la branche MAIN:

N°	Etape	Description
1	Build	Création de l'environnement Linux, Java..
2	Check for change in MS X	Détection s'il y a eu des changements sur les différents micro-services pour effectuer les tests
3	Test	Si un changement a eu lieu sur un micro-service, la phase de test est exécutée
4	Publish	Une image est publiée sur le docker hub du projet Docker Hub

L'étape de déploiement n'est pas réalisée dans le cadre du POC, elle devra être mise en place lors de la mise à disposition de l'environnement de PRE-PROD, PROD.

Tests

Test unitaire

Les tests unitaires visent à tester principalement des « unités » de code. Dans le cadre du POC, les tests sont concentrés principalement sur les tests de plus “haut niveau”, tels que les tests fonctionnels, E2E et les tests de charge.

Test fonctionnel

Dans le contexte de Spring Boot, les tests fonctionnels, également appelés tests d'intégration, peuvent être réalisés en utilisant le module de test Spring Boot. La librairie principale qui facilite ces tests est SpringBootTest (https://spring.io/guides/gs/testing-web). Chaque endpoint des micro-services sont testés et exécutés par la pipeline CI/CD lors de la livraison de code sur le référentiel Github.

Test End to End

La technologie utilisée se base sur NodeJS (https://nodejs.org/) et Selenium (https://www.selenium.dev/documentation/webdriver/). Ils permettent de tester l’interface graphique afin de s’assurer du succès du processus de réservation de lit d'hôpital dans le cadre du POC. La pipeline CI/CD exécute les tests E2E à chaque livraison de code dans le repository Github.

Test de stress

Les tests de stress sont réalisés avec le logiciel Apache JMeter : (https://jmeter.apache.org/)
Ils sont disponibles disponible depuis le repository Medhead :
https://github.com/OC-P11-MedHead/medhead-app/tree/main/jmeter

Les principaux endpoint sont testés et doivent correspondre aux critères suivants :

Critères	Description
Temps de réponse	Comme spécifié dans les critères d’exigences liés à la performance, chaque requête doit répondre en moins de 200 ms avec une charge de 800 requêtes par seconde.
Status code	Chaque requête doit renvoyer un code http 20*.
Correspondance dans la réponse	Un élément attendu est testé dans la réponse de chaque requête pour assurer la validité des tests.

Résultats obtenus

Exigences pour la PoC	Validé ?
Fournir une API Restful respectant les conditions suivantes :	
• Technologies Java	✅
• l'API doit pouvoir s'inscrire dans une architecture microservice	✅
Fournir une interface graphique qui consomme l'API	✅
La technologie cliente respecte les choix de framework Angular, React ou VueJS	✅
S'assurer que les données sont correctement protégées	✅
S'assurer que la PoC est entièrement validée avec des tests reflétant la pyramide de tests (tests unitaires, d'intégration, E2E, test de stress)	✅
Fournir une intégration et livraison continue (CI/CD)	✅
Le code est versionné à l'aide d'un workflow Git adapté	✅
Une documentation technique de la PoC pour son exécution et la ré-utilisation (test, installation, workflow	✅
Le code est versionné à l'aide d'un workflow Git adapté	✅
Fournir un document de reporting indiquant les résultats et enseignement de la PoC	✅

Recommandations pour la solution finale

Le POC a permis d'obtenir des micro-services découplés et facilement extensibles. Ces éléments pourront servir de modèle de construction pour la solution finale.

Des ajustements sont à réaliser pour le produit final :

- Modification du système de base de données H2 vers une base de données de production (SQL ou NoSQL)
- Mise en place de la phase de déploiement dans le pipeline CI/CD (vers environnement de pré-prod, production)
- Mise en place de l'authentification OAuth
- Mise en place du système de publication de message
- Mise en place du serveur de websocket pour un affichage en temps réel
- Mise en place d'une documentation API
- Amélioration générale des tests unitaires, fonctionnels pour garantir une meilleure couverture de code.
- Les tests de stress sont à intégrer dans la pipeline CI/CD. Il reste néanmoins à effectuer ce type de test à grande échelle sur le micro-service de calcul de distance en production de par les limites de l'api Distance Matrix pour la version de développement.