MedHead

Document de reporting

Sommaire

1. Objectif du document

Ce document de reporting vise à fournir une vue d'ensemble complète des résultats obtenus lors de la réalisation de la preuve de concept (PoC) pour la plateforme d'intervention d'urgence du consortium MedHead. Il a pour objectif de détailler la conformité avec les exigences du projet, de justifier les choix technologiques et architecturaux effectués, et de présenter les résultats des tests de performance ainsi que les enseignements tirés. Ce rapport sert également à démontrer la capacité de la solution à répondre aux besoins critiques du consortium, en soulignant son potentiel à améliorer les opérations d'urgence et à sauver des vies.

2. Public cible

Ce livrable est destiné à toutes les personnes impliquées dans le projet. Il est particulièrement utile au comité d’architecture et aux parties prenantes liées à la réalisation du projet.

3. Technologies et Architecture

3.1. Choix des Technologies Backend et Frontend

Backend : Spring Boot

Justification : Spring Boot est un framework Java qui offre une grande flexibilité pour le développement de microservices. Il est robuste, bien documenté, et largement utilisé dans les applications à grande échelle, ce qui en fait un choix idéal pour cette PoC.

Frontend : Angular

Justification : Angular est un framework de développement frontend moderne, maintenu par Google. Il est adapté pour construire des applications web complexes et est largement adopté dans l'industrie. Angular permet de créer une interface utilisateur dynamique et réactive, tout en facilitant l'intégration avec une API backend.

3.2. Conformité aux Normes (RGPD et NHS)

Respect du RGPD et NHS: Les données personnelles sont protégées par une API Key, garantissant l'accès sécurisé aux informations sensibles. Aucune donnée personnelle n'est stockée ou traitée sans les protections adéquates. De plus en environnement de productions plusieurs sécurités seront évidemment mises en place (HTTPS…)

3.3. Principes d’Architecture

Microservices : L'architecture du projet est conçue pour s'intégrer dans une approche de type microservices, bien que la PoC elle-même ne soit pas entièrement décomposée en microservices. Nous avons adopté une architecture modulaire avec une séparation claire entre le front-end et le back-end, ce qui facilite une future évolution vers une architecture microservices.

Le back-end, développé avec Spring Boot, expose des services RESTful, ce qui permettrait à chaque service ou module d'être isolé et déployé indépendamment dans un environnement de production.

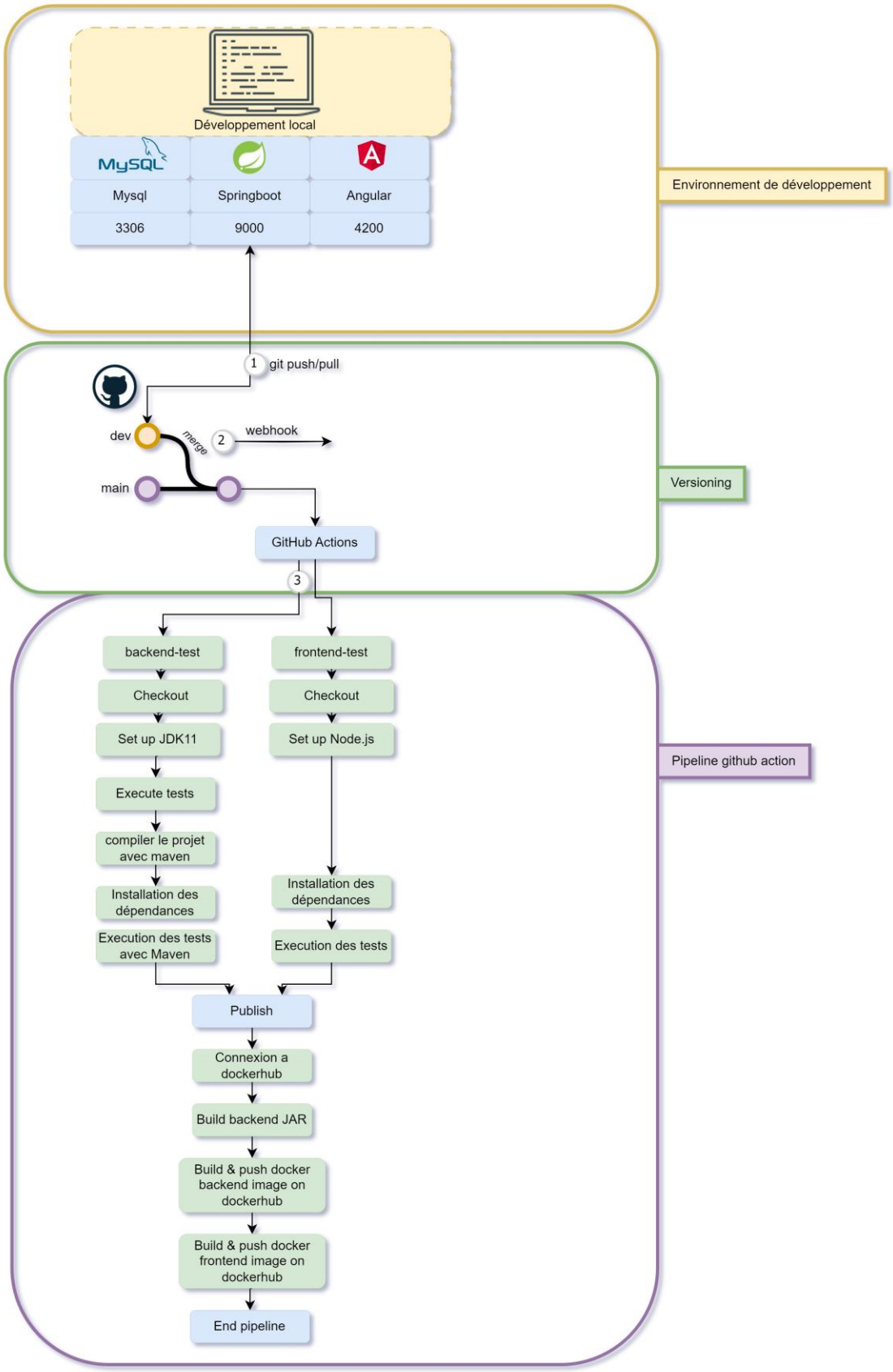
API REST Sécurisée : L'API est sécurisée par une clé API (X-API-KEY), assurant que seules les requêtes authentifiées peuvent accéder aux services. Ce modèle de sécurité est conforme aux meilleures pratiques pour les applications médicales.

Flexibilité et Scalabilité : L’architecture permet une adaptation rapide aux besoins spécifiques des différents membres du consortium. Les microservices peuvent être mis à jour ou étendus indépendamment, offrant une grande flexibilité.

4. Tests de la PoC

4.1. Pipeline CI/CD

Schéma général & Ci-Cd



Notre pipeline GitHub Actions est configuré pour automatiser l'intégration continue et le déploiement. Lorsqu'un code est poussé vers la branche principale, le pipeline s'exécute en plusieurs étapes. Il commence par les tests backend et frontend. Le job backend-test compile le projet, installe les dépendances, et exécute les tests de l'API en utilisant Maven et MySQL. Le job frontend-test installe les dépendances Node.js, configure Puppeteer pour les tests frontaux, et exécute les tests. Une fois les tests réussis, le job publish se charge de créer des images Docker pour le backend et le frontend, puis les pousse vers DockerHub. Ce processus garantit que le code est testé et prêt pour le déploiement automatisé.

4.2. Tests et Résultats

Tests Unitaires :

- **Outils** : JUnit et Mockito.

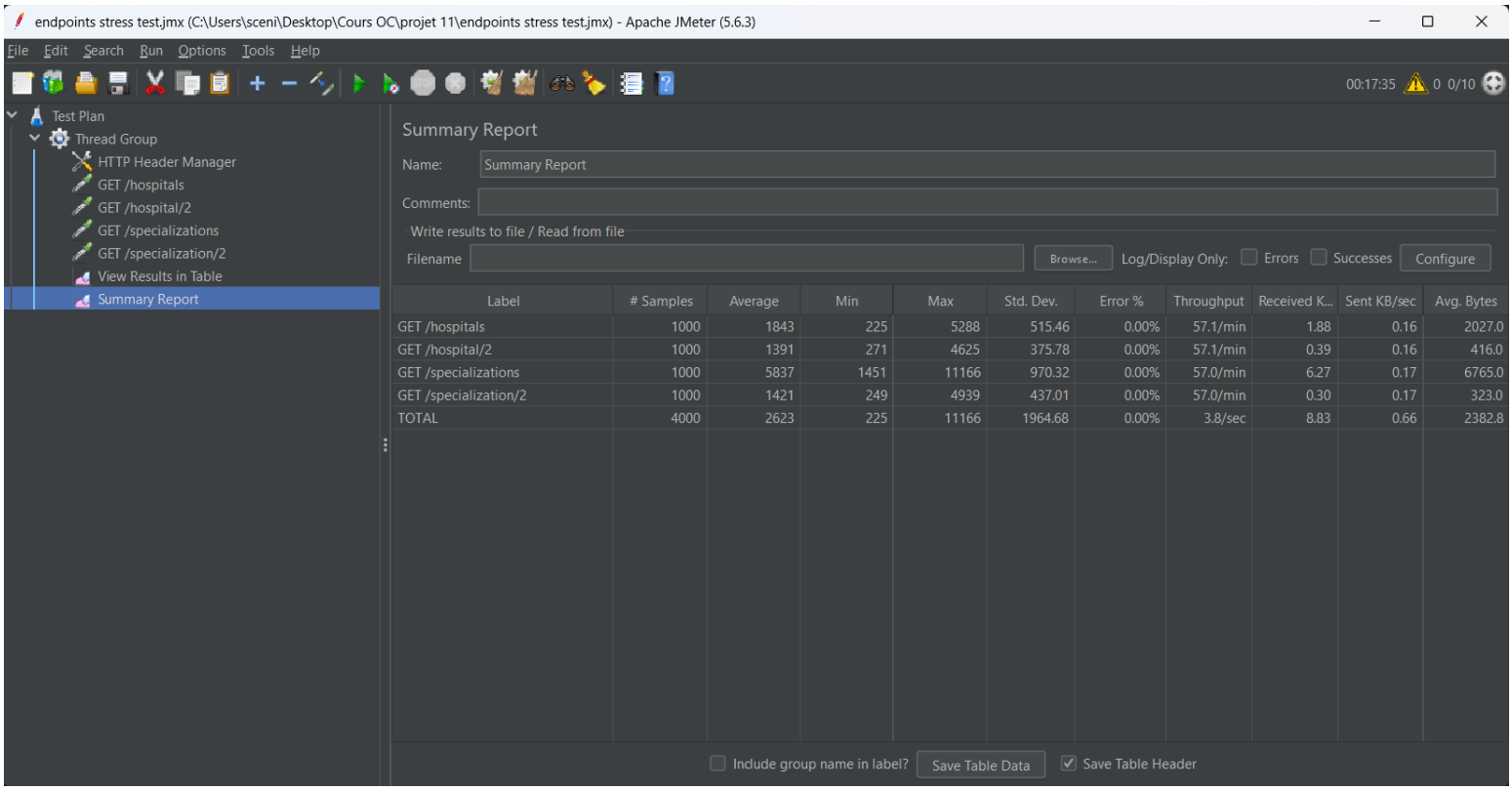
- **Couverture** : Les principaux cas d'utilisation sont couverts, avec des simulations pour les dépendances externes.

- Intérêt de Mocker les Tests :

Lors de l'écriture des tests unitaires, il est crucial de s'assurer que ces tests ne dépendent pas des autres parties du système, comme les bases de données, les services externes ou les composants qui ne sont pas directement liés à l'unité testée.

Mockito est un framework de mocking qui permet de simuler le comportement des objets dépendants au sein des tests unitaires. Par exemple, au lieu de faire des appels réels à une base de données ou à un service externe, Mockito permet de "mock" ces appels, en fournissant des réponses préprogrammées.

Tests de Stress :



Interprétation des résultats : Les tests de stress montrent des temps de réponse variables avec une moyenne légèrement élevée pour certaines requêtes. Les résultats sont cependant à considérer avec précaution, car les tests ont été réalisés dans un environnement local limité en ressources, ce qui explique un temps de réponse légèrement élevé du serveur Tomcat.

Le fait que l'application soit testée en local sur une machine personnelle plutôt que sur un serveur optimisé pour le déploiement de production a un impact direct sur les performances observées. Le matériel local ne peut pas reproduire les conditions réelles de production, comme un serveur cloud avec des capacités de scalabilité et de gestion de charge optimisées.

5. Analyse et Recommandations

5.1. Validité de la PoC

La PoC a démontré la validité des choix technologiques et de l’architecture pour répondre aux besoins du projet MedHead. Les tests ont montré que l'application est capable de gérer des charges importantes, mais nécessitera une infrastructure plus robuste pour un déploiement en production.

5.2. Recommandations pour l’Avenir

- **Déploiement Cloud** : Pour des tests plus représentatifs, il est conseillé de déployer la PoC sur une infrastructure cloud.

- **Optimisation des Performances** : Réaliser des optimisations spécifiques pour améliorer les temps de réponse, en particulier pour les endpoints critiques comme /specializations.

- **Renforcement de la Sécurité** : Bien que l'API Key soit une solution efficace, l’adoption de mécanismes comme OAuth ou JWT pourrait être envisagée pour renforcer encore la sécurité.

6. Conclusion

La preuve de concept (PoC) réalisée dans le cadre de ce projet a permis de valider l'architecture cible proposée pour le système d'intervention d'urgence du consortium MedHead. Nous avons démontré la faisabilité technique et la robustesse des choix technologiques effectués, notamment avec l'utilisation de Spring Boot pour le back-end et d'Angular pour le front-end.

Cette PoC a atteint son objectif principal : démontrer que la plateforme proposée peut répondre aux besoins critiques, en particulier dans des situations d'urgence où chaque seconde peut compter.