Une image contenant personne, bleu, bleu vert, Bleu électrique

Description générée automatiquement

# Rapport de Proof of Concept (PoC) en Java

Table des matières

[Rapport de Proof of Concept (PoC) en Java 1](#_Toc143783591)

[Objectifs 2](#_Toc143783592)

[Contexte 2](#_Toc143783593)

[Méthodologie 2](#_Toc143783594)

[Conception architecture 2](#_Toc143783595)

[Implémentation 3](#_Toc143783596)

[Normes 3](#_Toc143783597)

[Tests 4](#_Toc143783598)

[Performances 4](#_Toc143783599)

[Enseignements 5](#_Toc143783600)

# Objectifs

Les objectifs du Proof of Concept étaient les suivants :

1. Évaluer la faisabilité technique et architecturale du futur projet

2. Démontrer la mise en œuvre réussite de diverses fonctionnalités telles que :

* La connexion sécurisée
* Récupération de données en temps réel
* Réservation et annulation de lits d’hôpitaux

3. Évaluer les performances, la scalabilité et la robustesse de la solution proposée.

# Contexte

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. Cette plateforme devra pallier les risques liés au traitement des recommandations de lits d'hôpitaux dans des situations d'intervention d'urgence.

# Méthodologie

## Conception architecture

L’architecture du projet a été imposée par la direction de MedHead. Le Proof of concept s’appuie donc sur une architecture micro-service avec un front distinct.

**Microservice d'authentification :** ce microservice est responsable de l'authentification et l'autorisation des utilisateurs. Il permet d’implémenter des mécanismes tels que la création de comptes, la connexion, la gestion des sessions et la génération de jetons d'accès sécurisés (comme des JWT) pour les utilisateurs. Il donne aussi accès aux fonctionnalités du deuxième microservice après authentification.

**Microservice de données hospitalières :** ce microservice fournit les fonctionnalités liées aux données hospitalières, telles que le nombre de lits disponibles, la récupération de la liste des différentes spécialités référencées par hôpitaux, la gestion de réservations de lits, etc. Il est conçu pour être modulaire et extensible, de sorte que de nouvelles fonctionnalités puissent être ajoutées facilement sans affecter l'ensemble du système.

**Front-end :** l'interface utilisateur de notre application médicale permet aux utilisateurs d’interagir avec les deux micro-services cités.

## Implémentation

|  |  |  |
| --- | --- | --- |
|  | Choix | Justification |
| Langage de programmation | Java | Imposé |
| Architecture | Micro-service | Imposé |
| Framework | Spring Boot | Framework reconnue pour la mise en place de micro-service |
| SGBD | Maria dB | Base de données relationnel générique et open source |
| Front-end | React JS | Respect des critères imposés dans les différents documents |
| IDE | Intellij et VS code | Prise en main facile avec une grande communauté |

## Normes

En raison de la nature du projet, nous sommes confrontés à deux types de normes totalement différentes :

* Les normes de développement (Convention de Nommage, Structuration du Projet, Documentation, Tests, etc.)
* Les normes liées aux données personnelles des utilisateurs de l’application (RGPG, ISO 13485, HL7)

## Tests

Des tests automatisés ont été mis en place pour chaque micro-service, y compris des tests unitaires et d'intégration. Cela garantit la qualité et la robustesse du système.

Le Continuous Integration (CI) et le Continuous Deployment (CD) sont des pratiques essentielles dans le développement logiciel moderne, visant à automatiser et à accélérer le processus de livraison de logiciels. Jenkins est l'un des outils les plus populaires pour mettre en œuvre ces pratiques.

Jenkins facilite l'intégration continue et permet de fusionner du code dans un référentiel commun plusieurs fois par jour. À chaque fusion, Jenkins déclenche automatiquement des builds, exécute des tests et signale les problèmes de manière proactive. Cela aide à identifier les erreurs et les conflits dès qu'ils surviennent, permettant ainsi de les résoudre plus rapidement et de maintenir la qualité du code.

## Performances

Afin de garantir une certaine performance et de s’assurer que la future application respectera les différentes métriques imposées. Différents tests de performance ont été appliqués à celles-ci.

Le test de performance de notre application Java Spring a été réalisé efficacement en utilisant des outils tels que JMeter et Postman.

**JMeter** offre la possibilité de simuler des charges de travail réalistes en envoyant des requêtes HTTP à l'application, mesurant ainsi sa réactivité et sa stabilité sous différentes conditions de charge. En configurant des scénarios de test dans JMeter, les développeurs peuvent évaluer les performances de l'application en termes de temps de réponse, de débit et de capacité de traitement.

D'autre part, **Postman** peut être utilisé pour des tests plus spécifiques au niveau de l'API. Il permet d'envoyer des requêtes HTTP individuelles et d'analyser les réponses pour s'assurer que les points d'extrémité de l'API fonctionnent correctement et efficacement.

En combinant l'utilisation de JMeter pour les tests de charge et de Postman pour les tests d'API ciblés, nous avons obtenu une vue complète des performances de leur application Java Spring.

# Enseignements

La mise en œuvre d'un POC basé sur Java Spring Boot pour la gestion des réservations de lits en fonction des spécialités hospitalières a révélé des enseignements précieux.

Ce PoC a permis de comprendre l'importance d'une architecture robuste et modulaire pour répondre aux besoins spécifiques de l'environnement hospitalier.

Tout d'abord, il est apparu crucial de bien modéliser les entités impliquées, telles que les lits et les spécialités médicales. L'utilisation de concepts de programmation orientée objet a facilité la gestion des relations entre ces entités, permettant ainsi une représentation claire et cohérente des données.

De plus, l'implémentation de l'injection de dépendances au sein du framework Spring Boot a favorisé la flexibilité et la maintenabilité du code.

En reliant les différents composants du système de réservation de lits, il est devenu plus facile d'apporter des modifications ou des améliorations sans perturber l'ensemble du système.

L'utilisation d'API REST pour échanger des données entre le système de réservation de lits et d'autres applications, telles que l’authentification simplifie la maintenance de l’application.

En outre, la mise en place de tests unitaires et de tests d'intégration a prouvé son efficacité pour garantir la qualité du code et la fiabilité du système. Ces tests ont permis de détecter rapidement les erreurs et les anomalies, tout en fournissant une documentation vivante des fonctionnalités mises en œuvre.

En résumé, le PoC basé sur Java Spring Boot pour la gestion des réservations de lits en fonction des spécialités hospitalières a mis en évidence l'importance de la modélisation précise, de l'architecture micro-service, de l'intégration transparente et des tests rigoureux. Ces enseignements sont transférables à d'autres projets de développement au sein du domaine de la santé, où la précision, la rapidité et la fiabilité sont des éléments essentiels.

# Evolution

En ce qui concerne les perspectives d'évolution, du Proof of Concept établi autour de Java Spring Boot pour la réservation de lits selon les spécialités, ouvre la voie à des développements significatifs. Sur le plan applicatif, une expansion vers des fonctionnalités telles que la gestion de la disponibilité en temps réel des lits, l'optimisation des parcours cliniques et la personnalisation des soins pourrait enrichir l'expérience des patients et du personnel médical.

Au niveau architectural, alors que l'approche actuelle repose sur une architecture de micro-services pour la modularité, la possibilité d'adopter une architecture pilotée par des événements mérite d'être examinée. Cette approche permettrait une meilleure décomposition fonctionnelle, réduisant les interdépendances entre les services et facilitant l'évolutivité tout en assurant une synchronisation précise des données. Les événements, tels que les notifications de changement d'état des lits ou les mises à jour de statut de lits, pourraient être transmis de manière asynchrone entre les composants, améliorant la réactivité et la cohérence du système dans un environnement hospitalier dynamique.

# Conclusion

En conclusion, le Proof of Concept basé sur Java Spring Boot pour la gestion des réservations de lits en fonction des spécialités hospitalière a illustré la pertinence d'une approche méthodique. Les éléments clés tels que la modélisation précise, l'architecture micro-service, l'intégration transparente et les tests approfondis se sont révélés cruciaux pour le développement de solutions efficaces et fiables. Ce PoC offre une base solide pour l'optimisation continue et l'expansion future des systèmes de gestion hospitalière au sein de notre groupe hospitalier.