

18 May 2024

**CLÉMENT MARCEL**

# **REPORTING**

# **POC**

1. Contexte	3
2. Hypothèses	3
3. Méthodologie	3
4. Objectifs du POC	3
5. Principes architecturaux	3
6. Les microservices	4
7. Définition des mesures clés	4
8. Collecte des données	5
9. Technologies	6
a. Back-End	6
b. Front-End	6
c. Pipeline CI/CD	6
10. Versionnement	7
11. Résultats	8
a. Stratégie de test	8
b. Résultats des tests	8
c. Test de stress	8
d. Test E2E	9
12. Recommandations	10
13. Conclusion	10

# 1. CONTEXTE

La PoC vise à développer une application web permettant aux utilisateurs de trouver des hôpitaux proches offrant des services médicaux spécifiques. L'objectif est de valider l'architecture technique et de démontrer la faisabilité et l'efficacité de la solution proposée.

## 2. HYPOTHÈSES

- **Performance** : La plateforme doit être suffisamment rapide et réactive pour répondre aux besoins des utilisateurs.
- **Valeur ajoutée** : La nouvelle solution doit apporter une valeur ajoutée significative par rapport aux solutions existantes.
- **Rétention** : Les utilisateurs doivent continuer à utiliser la plateforme sur une longue période.

## 3. MÉTHODOLOGIE

La méthodologie adoptée pour cette PoC est le "prototypage rapide", permettant de créer rapidement un modèle fonctionnel afin de recueillir des retours et de valider le projet.

## 4. OBJECTIFS DU POC

L'objectif principal est d'améliorer la qualité des soins aux patients. En combinant les forces de chaque entreprise du consortium, la plateforme partagée vise à soutenir l'innovation des solutions centrées sur le patient et à améliorer les activités quotidiennes des soins.

## 5. PRINCIPES ARCHITECTURAUX

1. **Modularité** : Utilisation d'une architecture microservices pour permettre la modularité et la scalabilité.
2. **Séparation des préoccupations** : Distinction claire entre les couches frontend, backend et données.

3. **Scalabilité** : Capacité à scaler indépendamment chaque microservice en fonction de la charge.
4. **Sécurité** : Mise en place de mesures de sécurité pour protéger les données des utilisateurs.

## 6. LES MICROSERVICES

- **Authentification et Autorisation** : Utilisation d'OAuth 2.0 avec des tokens JWT pour authentifier et autoriser les requêtes.
- **API Gateway** : Centralisation des appels via une API Gateway pour gérer l'authentification, l'autorisation et la limitation de débit.
- **Chiffrement TLS/mTLS** : Chiffrement des communications avec TLS et mutual TLS pour une authentification mutuelle renforcée.
- **Ségrégation des Réseaux** : Isolation des microservices dans des réseaux virtuels distincts, avec pare-feu et groupes de sécurité pour contrôler le trafic.
- **Surveillance et Logging** : Utilisation d'outils de surveillance et de logging pour détecter rapidement les anomalies et les accès non autorisés.

## 7. DÉFINITION DES MESURES CLÉS

Métrique	Technique de mesure	Valeur cible	Justification
Urgences acheminées	Chronométrage	12 minutes	Réduire le temps de traitement des urgences
Temps de réponse	Monitoring	< 200 ms	Assurer une haute réactivité de la plateforme
Nombre de requêtes/s	Monitoring	Jusqu'à 800	Supporter un haut volume de trafic
Taux de conversion	Google Analytics	Augmenter de 10%	Améliorer l'engagement utilisateur

Taux de rebond	Google Analytics	Réduire de 20%	Accroître l'interaction des utilisateurs avec le contenu
Temps de chargement	PageSpeed Insights	< 3 secondes	Optimiser l'expérience utilisateur

## 8. COLLECTE DES DONNÉES

Les données collectées incluent la position géographique, la spécialité NHS sélectionnée par le patient et les mesures de taux d'échec. En conformité avec la RGPD, aucune donnée sensible ou personnelle n'est stockée par l'application web.

De plus l'API est sécurisé par une clé nécessaire pour communiquer avec cette dernière.

Dans une future version, il est préconiser d'ajouter une double authentification afin de sécuriser davantage l'application et ses utilisateurs.

### Synthèse RGPD:

- **Collecte minimale des données :** Seules les données strictement nécessaires pour le fonctionnement de l'application sont collectées. Cela inclut des données non sensibles telles que la localisation géographique et la spécialité médicale choisie par l'utilisateur.
- **Consentement explicite :** Avant toute collecte de données, les utilisateurs doivent donner leur consentement explicite, en conformité avec les exigences du RGPD.
- **Anonymisation et pseudonymisation :** Les données collectées sont anonymisées ou pseudonymisées pour garantir qu'aucune information personnelle identifiable n'est stockée ou traitée.
- **Droit des utilisateurs :** Les utilisateurs ont le droit de demander la suppression de leurs données personnelles à tout moment, et cette demande sera traitée rapidement conformément aux dispositions du RGPD.
- **Chiffrement :** Toutes les données en transit et au repos sont chiffrées à l'aide d'algorithmes robustes pour assurer la confidentialité et la sécurité des informations des utilisateurs.

## 9. TECHNOLOGIES

### a. Back-End

- **Langage** : Java (Imposé)
- **Framework** : Spring Boot
- **Base de données** : H2 Database

### b. Front-End

- **Langage** : TypeScript
- **Librairie** : React - Choisi pour sa flexibilité, sa performance et sa large adoption dans le développement front-end moderne.

### c. Pipeline CI/CD

- **Outil** : CircleCI
- **Étapes** :
  1. Backend : Exécution du build du backend, les tests se lancent automatiquement avant le build, le tout sous un environnement Linux
  2. Frontend : Execution des tests unitaires puis exécution du build du frontend sous un environnement Linux

The screenshot displays a CircleCI pipeline status for the repository 'OC\_P11\_Code' (commit 96). The pipeline is named 'build\_and\_deploy' and is currently in a 'Success' state, indicated by a green checkmark and the word 'Success' in a green box. The pipeline is running on the 'main' branch. The commit hash 'd4fdd5a' is shown, along with the commit message 'update readme'. Below the pipeline status, a list of jobs is shown under the heading 'Jobs'. Two jobs are listed: 'build\_hospital\_service' (duration 131) and 'build\_frontend' (duration 132). Both jobs are marked as successful with green checkmarks.

Jobs	Status	Duration
build_hospital_service	Success	131
build_frontend	Success	132

## 10. VERSIONNEMENT

Pour une gestion efficace du développement et des déploiements, il est recommandé d'adopter le modèle Gitflow :

Branches Principales :

- **Main** : Cette branche contient le code stable et prêt pour la production. Seules les versions validées passent sur cette branche.
- **Develop** : Cette branche est utilisée pour l'intégration de nouvelles fonctionnalités et les tests. Tout le développement passe par cette branche avant d'être fusionné dans main.

Processus de Développement :

- **Développement** : Les nouvelles fonctionnalités et corrections de bugs sont développées sur des branches de fonctionnalités individuelles dérivées de develop.
- **Intégration** : Les branches de fonctionnalités sont fusionnées dans develop une fois le développement terminé et les tests unitaires réussis.
- **Pré-production** : La branche develop est déployée dans un environnement de pré-production pour des tests d'intégration et de validation.
- **Production** : Après validation en pré-production, develop est fusionnée dans main, qui est ensuite déployée en production.

Ce modèle garantit une gestion structurée et organisée du code source, facilitant ainsi la collaboration entre les développeurs et assurant la stabilité du code en production.

# 11. RÉSULTATS

## a. Stratégie de test

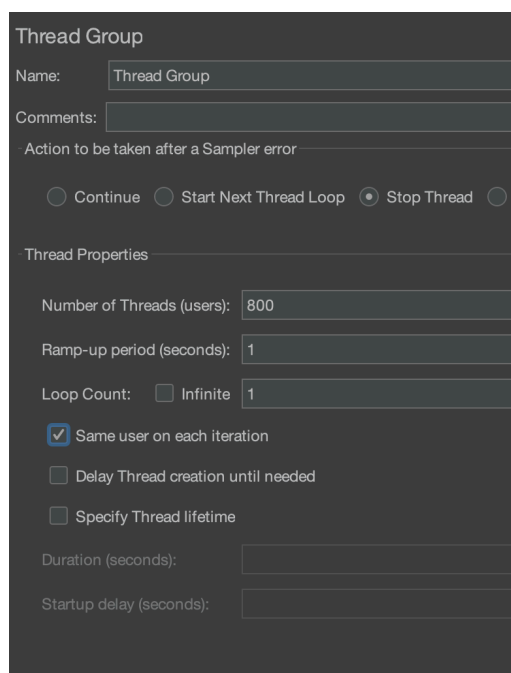
- **Tests unitaires** : Valident les fonctionnalités de base de chaque comportement.
- **Tests d'intégration** : Vérifient les interactions entre plusieurs composants.
- **Tests end-to-end** : Vérifient le fonctionnement global de l'application dans un scénario réaliste.

## b. Résultats des tests

- **Unitaires** : Tous les tests unitaires passent avec succès, validant le comportement.
- **Intégration** : Les tests d'intégration montrent que les composants communiquent correctement entre eux.
- **End-to-end** : Les tests E2E valident les fonctionnalités principales du point de vue de l'utilisateur.

## c. Test de stress

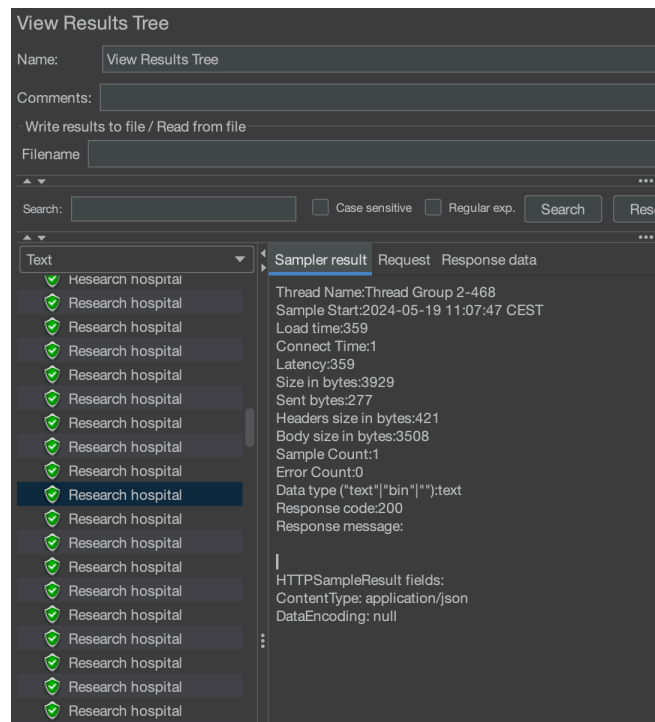
- **Objectif** : Évaluer la capacité de l'application à gérer des charges importantes.
- Configuration :



The screenshot shows the 'Thread Group' configuration window in JMeter. It includes fields for 'Name' (Thread Group), 'Comments', and 'Action to be taken after a Sampler error' (with radio buttons for Continue, Start Next Thread Loop, Stop Thread, and Stop Thread and Save State). Under 'Thread Properties', there are fields for 'Number of Threads (users)' (800), 'Ramp-up period (seconds)' (1), and 'Loop Count' (with 'Infinite' unchecked and '1' entered). There are also checkboxes for 'Same user on each iteration' (checked), 'Delay Thread creation until needed', and 'Specify Thread lifetime'. At the bottom, there are fields for 'Duration (seconds)' and 'Startup delay (seconds)'.



- **Résultats** : L'application a montré une bonne scalabilité, avec des temps de réponse acceptables sous des charges élevées, légèrement plus lent que ce qui est attendu, du à l'environnement de développement.



#### d. Test E2E

Les tests End-to-End réalisés avec Cypress montrent qu'un utilisateur peut rechercher un hôpital, sélectionner une spécialité, réserver un lit et annuler la réservation. Voici les étapes principales :

##### Recherche d'un hôpital :

1. Saisie de la localisation et sélection de la spécialité.
2. Vérification de l'affichage du résultat de recherche.

##### Vérification des informations de l'hôpital :

Confirmation de la visibilité des détails de l'hôpital affiché.

Le lit est bien réservé si, le nombre de lit est mis à jour.

## 12. RECOMMANDATIONS

Pour améliorer les résultats de la PoC, il est recommandé de :

- **Environnement de test** : Exécuter la PoC dans un environnement de test plus performant pour obtenir des résultats plus précis et représentatifs.
- **Intégration de Docker** : Utiliser Docker pour les applications front-end et back-end, facilitant ainsi la gestion des déploiements et des environnements de développement.
- **CI/CD avec Docker** : Intégrer Docker dans le pipeline CI/CD, permettant de créer des images Docker pour les applications, les tester, puis les déployer sur le cloud. Cela améliore la cohérence entre les environnements de développement, de test et de production.
- **Modularité et scalabilité** : Continuer à exploiter l'architecture microservices pour permettre une modularité et une scalabilité indépendantes des différents composants.
- **Séparation des préoccupations** : Maintenir une distinction claire entre les couches frontend, backend et données pour assurer une meilleure organisation et maintenabilité du code.
- **Sécurité** : Renforcer les mesures de sécurité, incluant l'authentification et l'autorisation via OAuth 2.0 avec tokens JWT, le chiffrement TLS/mTLS pour les communications, et l'isolement des microservices dans des réseaux virtuels distincts.
- **Surveillance et logging** : Mettre en place des outils de surveillance et de logging pour détecter rapidement les anomalies et les accès non autorisés.

Ces recommandations visent à améliorer la performance, la sécurité et la scalabilité de la solution, tout en facilitant les déploiements et la maintenance à long terme.

## 13. CONCLUSION

La PoC a démontré la faisabilité de l'application de recherche d'hôpitaux en utilisant une architecture microservices.

Les technologies choisies se sont révélées adaptées, offrant de bonnes performances et une scalabilité suffisante. Le respect des normes RGPD a été assuré mais devront évoluer, et les tests end-to-end ont validé les fonctionnalités principales.