Document de reporting POC

Table des matières

I. Introduction

A. Contexte du POC

Le système d'intervention d'urgence en temps réel vise à répondre à un besoin crucial dans le domaine de la santé : celui d'optimiser la prise en charge des patients nécessitant une hospitalisation d'urgence. En situation critique, chaque minute compte, et il est impératif de diriger le patient vers l'établissement de santé le plus approprié en fonction de sa condition et des spécialités requises.

B. Objectifs du POC

Le but ultime de ce POC est de démontrer l'efficacité de notre système en identifiant rapidement et avec précision l'hôpital le plus proche correspondant aux besoins spécifiques du patient.

II. Technologies utilisées

A. Technologies Back-end

Les microservices sont développés en Java avec Spring Boot comme framework principal. Spring Boot permet de construire facilement des microservices autonomes et de les exécuter avec un serveur web intégré.

Maven est utilisé comme outil de build et de gestion de dépendances. Il permet d'automatiser la compilation, les tests, le packaging et le déploiement des microservices.

Spring Cloud fournit des composants pour implémenter des patterns d'architecture microservices comme la découverte de services, le load balancing, les appels de procédures distantes, etc.

La base de données H2 en mémoire est utilisée lors des tests unitaires et d'intégration pour simuler une base de production.

L'accès aux données relationnelles se fait avec JPA (Java Persistence API) qui permet de mapper les objets Java vers les tables de la base de données.

Les tests unitaires sont réalisés avec JUnit pour les tests fonctionnels et Mockito pour mocker les dépendances. Ils permettent de valider le fonctionnement des services de façon isolée.

Un appel externe vers l’API de Google Maps est réalisé afin de réaliser le calcul de distance entre le patient et l’hôpital.

B. Technologies Front-end

Le frontend est développé avec Vue.js, un framework JavaScript open-source pour construire des interfaces utilisateur.

Vue permet de créer des composants réutilisables pour structurer l'interface. Il utilise une approche déclarative avec un système de templates pour rendre les données.

Le routing entre les pages est implémenté avec Vue Router.

Les appels API vers les microservices backend sont réalisés avec Axios pour les requêtes HTTP.

Les tests E2E sont réalisés avec Cypress, qui permet de tester l'application de bout en bout, du frontend jusqu'aux API du backend.

III. Normes et principes

A. Architecture Microservice

Le projet est basé sur une architecture microservice avec 3 microservices principaux développés de manière indépendante :

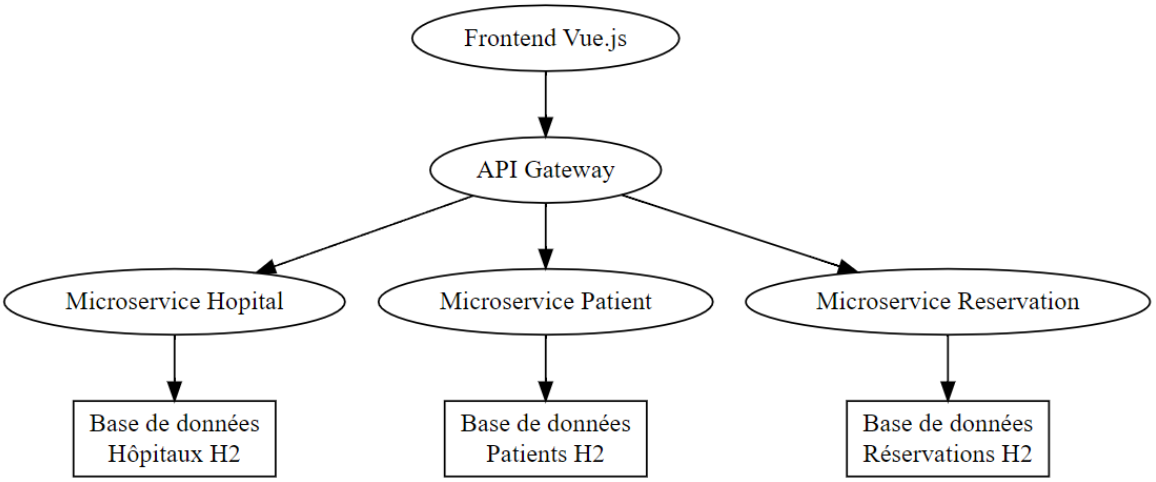
- Le microservice Hôpital gère les données des hôpitaux (nom, adresse, spécialités, nombre de lits, etc.).

- Le microservice Patient gère les informations des patients (nom, prénom, date de naissance, etc.).

- Le microservice Réservation permet d'enregistrer les réservations de lits pour les patients dans les hôpitaux.

Chaque microservice possède sa propre base de données et communique avec les autres via des API REST. Ils sont développés avec Spring Boot pour faciliter la mise en place d'une architecture microservice.

Cette architecture apporte de la flexibilité et permet le déploiement indépendant de chaque microservice. Chacun peut évoluer à son rythme avec sa propre stack technologique.



B. RGPD

Le projet est soumis au Règlement Général sur la Protection des Données (RGPD) étant donné qu'il traite des données personnelles de patients.

Pour se conformer au RGPD les mesures suivantes ont été mises en place :

- Un formulaire de consentement RGPD a été ajouté lors de l'inscription des patients afin de recueillir leur accord sur l'utilisation de leurs données conformément à la politique de confidentialité.

- Les données collectées sont limitées au strict nécessaire (nom, prénom, date de naissance, etc.) et leur utilisation est restreinte aux services fournis par l'application.

- Les données sont sécurisées lors de la transmission et du stockage, notamment via le chiffrement pour éviter les fuites.

- Les patients peuvent exercer leurs droits d'accès, de rectification et d'effacement de leurs données personnelles.

- En cas de fuite de données, une procédure de notification des autorités et des personnes concernées est prévue.

C. Normes de développement

Les normes de développement visent à établir des bases solides pour la conception, le déploiement et la maintenance du projet, celles qui ont été mises en oeuvre pour ce projet sont les suivantes :

Frontend

Framework Vue.js : Vue.js offre une gestion de l'état de l'application et une structure composante qui facilite le développement et la maintenance du code.

Tests E2E avec Cypress : L'utilisation de Cypress pour les tests end-to-end garantit la validation des parcours utilisateurs.

Vue Router : Vue Router facilite la gestion de la navigation entre les pages, offrant une expérience utilisateur fluide.

Requêtes HTTP avec Axios : Les requêtes HTTP vers l'API sont effectuées grâce à Axios.

Intégration continue : L'intégration continue est réalisée à l'aide de Github Actions qui assure un déploiement régulier du code et permet la détection des problèmes dans le code source.

Respect des Standards HTML et CSS : Le respect des normes HTML et CSS garantit une compatibilité et une accessibilité maximale de l'application sur les différents navigateurs.

Backend

Architecture Microservices : L'utilisation de Spring Boot en Java pour le développement des microservices offre une approche robuste et évolutive, cela permet à chaque microservice de pouvoir être déployé et maintenu de façon indépendante.

Tests Unitaires : Les tests unitaires réalisés avec Mockito assurent la fiabilité du code en s'assurant que chaque composant fonctionne correctement de façon isolée.

Tests de charge : Les tests de charge avec JMeter permettent d'évaluer les performances du système dans des situations où le trafic est élevé.

Communication entre microservices via API REST : La communication entre les microservices se fait grâce à l'API REST.

Bases de données H2 indépendantes : Chaque microservice dispose de sa propre base de données H2, ce qui permet d'isoler les données et d'offrir une maintenance indépendante.

Découpage en couches : L'architecture est découpée en couches (controllers, services, repository), cela améliore la lisibilité du code et facilite la maintenance.

D. Principes de l'architecture

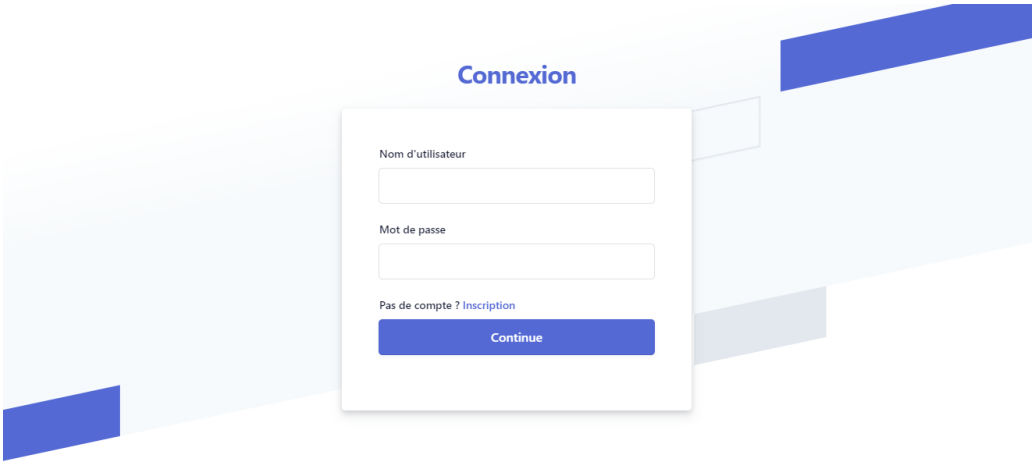
Les membres du consortium ont collaboré à la définition d'un ensemble de principes architecturaux par domaine, ils sont indiqués dans le document de définition des principes d'architecture.

IV. Résultats et enseignements de la POC

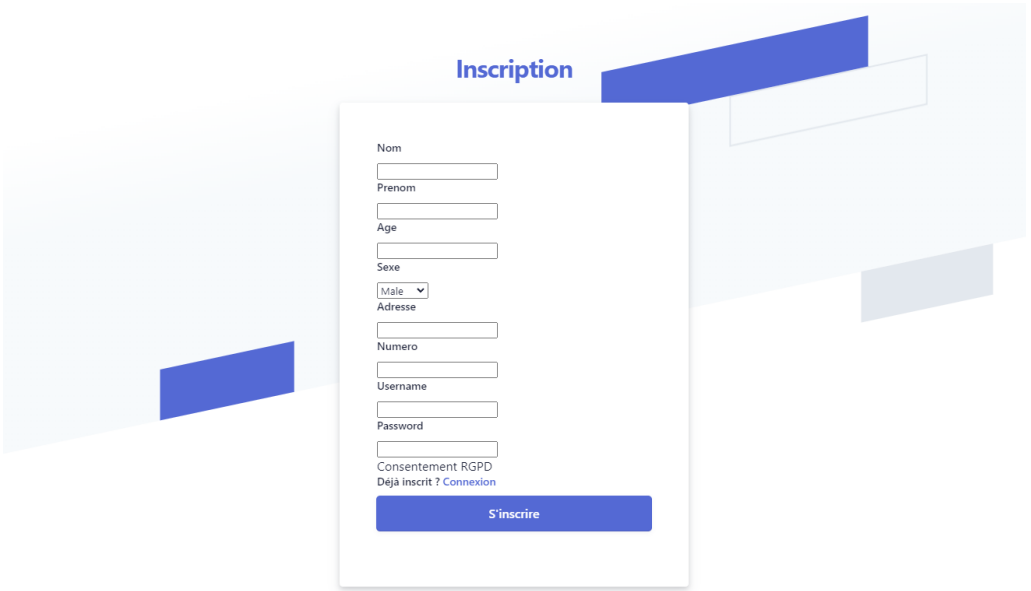
A. Interfaces

La POC est composée de 3 pages principales, une page de connexion, d'inscription et une page où l'on peut saisir son adresse actuelle afin que l'hôpital le plus proche soit proposé.

Page de connexion :



Page d’inscription :



Page des Hôpitaux :

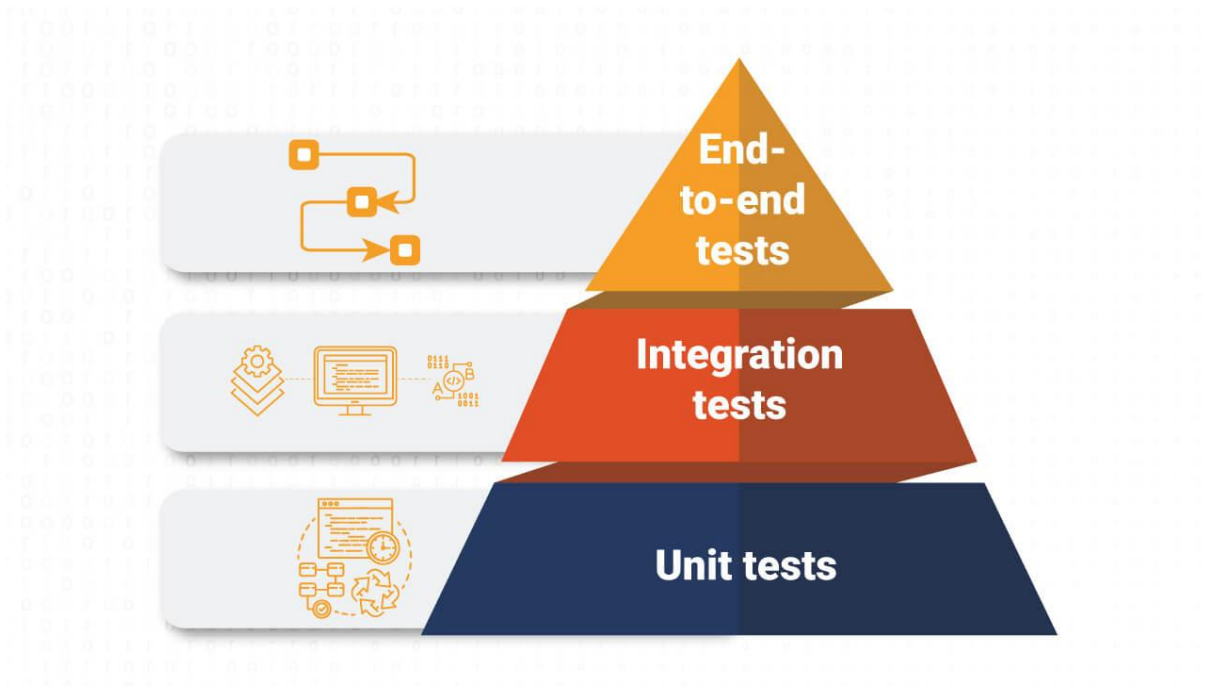
B. Pyramide de Tests

Le projet suit une pyramide de tests classique avec :

- Des tests unitaires pour valider chaque composant de manière isolée. Ils sont réalisés avec JUnit et Mockito pour les microservices Java.

- Des tests d'intégration pour valider l'intégration entre les microservices via leurs API REST. Spring Boot Test est utilisé pour tester les contrôleurs des microservices Spring.

- Des tests end-to-end (e2e) avec Cypress pour valider le parcours utilisateur de bout en bout dans l'application. Ils lancent l'application frontend et simulent des scénarios réels en interagissant avec l'interface comme un utilisateur.



Cette pyramide de tests permet de valider le fonctionnement de l'application à différents niveaux et d'éviter les régressions.

C. Analyse des résultats

Pour une charge de travail allant jusqu’à 800 requêtes par seconde pour récupérer l’hôpital le plus proche, la moyenne du temps de réponse est de 2131 millisecondes, ce qui est au dessus des 200 millisecondes attendus, cela est notamment dû à l’utilisation de l’API de Google qui est externe.

D. CI/CD

Le projet utilise GitHub Actions pour automatiser le pipeline CI/CD.

Le workflow est déclenché à la suite d'un push sur la branche principale main.

Dans un premier temps un job de test est réalisé afin d'exécuter les tests de la partie backend pour chaque microservice, Hôpital, Patient et Réservation.

Le job de build récupère le code source qui a passé les tests et le transforme en format prêt à être déployé pour chaque microservices.

Le job de frontend s'occupe de préparer l'environnement, réaliser les tests E2E et préparer pour le déploiement.

V. Conclusion

La preuve de concept répond aux exigences des parties prenantes qui sont les suivantes :

- Une API REST qui s'inscrit dans une architecture microservice avec Spring Boot.
- Une interface graphique qui consomme l'API avec une page de connexion, d'inscription, et de recherche des hôpitaux avec le framework VueJS.
- Les données sont protégées grâce à SpringSecurity.
- La POC est valide grâce aux nombreux et divers tests réalisés, unitaires, intégration, E2E et performance.
- Le pipeline d'intégration est pleinement fonctionnel.

Cependant, le temps de réponse de 200 millisecondes pour 800 requêtes par seconde ne peut pas être atteint.