Définition d'architecture

- Proof Of Concept Medhead -

Par Wilfrid Boudia, Architecte logiciel

Le 20/03/2023

Table des matières

A. Objet du document

Ce document présente l'analyse de base et l'objectif pour la validation de la Proof Of Concept (POC) de la future application de Medhead.

Il contient un rappel des besoins et des attentes des différents acteurs qui y seront listés travaillant sur ce projet.

Il vise à démontrer la faisabilité et l'intérêt d'un nouvel outils au sein de l'organisation.

Il explique aussi l'architecture mise en place par la POC et l'architecture conseillée pour l'application finale.

Il décrit également les scénarios d'utilisation, l'environnement de fonctionnement, la méthodologie, les risques et opportunités, les axes d'amélioration et également les livrables fournit.

B. Objectif

Ce document donne un descriptif générale de la POC et décrit les techniques utilisées pour créer l'application.

C. Description et contraintes du projet

Voir le document « **Hypothèse de validation** »

D. Principes d'architecture

Voir le document « **Principe de l'architecture** »

E. Parties prenantes

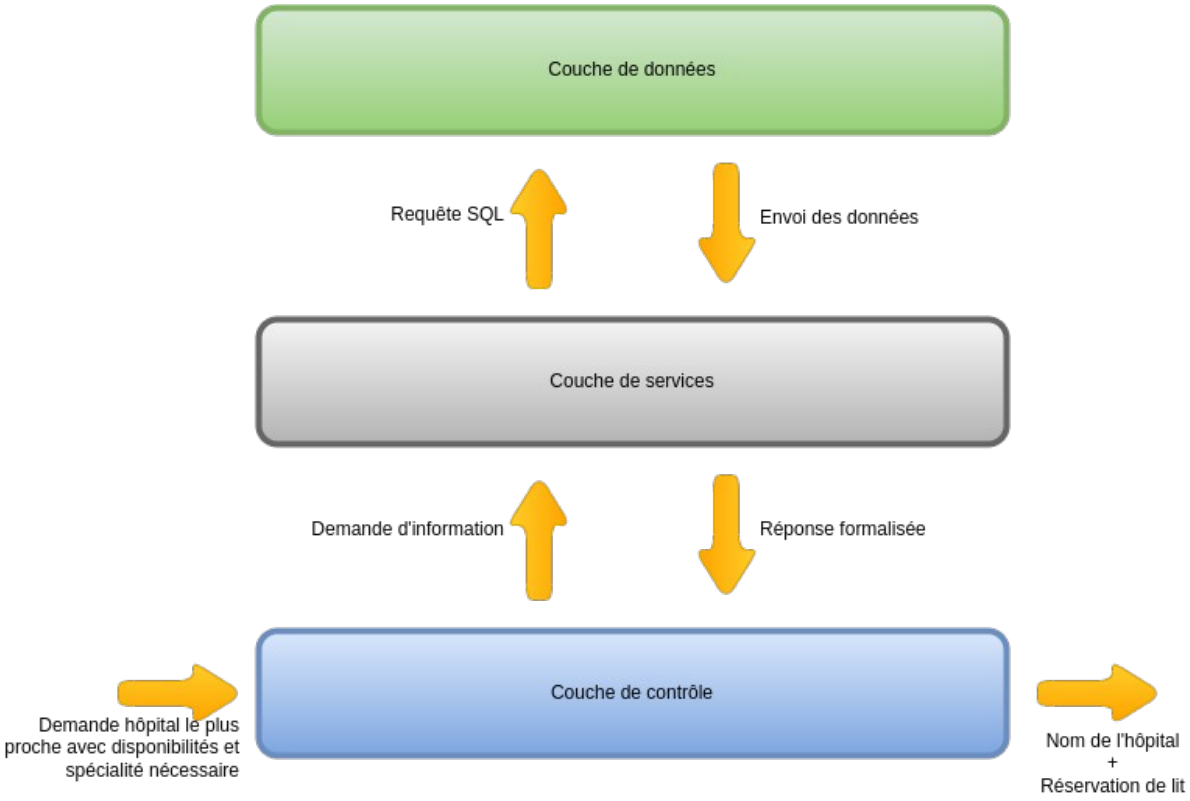
Ci-dessous la liste des parties prenantes au projet :

Organisation	Domaine spécialisé	Motivations métier	Remarques
Ursa Major Health	Planification de rendez-vous pour la médecine générale (GP)	Amélioration des processus de travail dans le cadre de l'intégration manuelle avec d'autres systèmes.	Consolidation des rendez-vous et réduction des problèmes d'intégration avec les systèmes de prise de rendez-vous hospitaliers.
Jupiter Scheduling	Prise de rendez-vous à l'hôpital	Amélioration des processus de travail dans le cadre de l'intégration manuelle avec d'autres systèmes.	Synergies avec d'autres systèmes de prise de rendez-vous et suppression des intégrations manuelles avec les systèmes de planification GP.
Emergency Expert Systems	Recommand ations de l'hôpital en temps réel en fonction de la disponibilité des lits	Intégration et validation des données requises pour améliorer les décisions.	Réduire le taux de mauvaises recommandations en pondérant les décisions en fonction des antécédents médicaux connus des patients.
Schedule Shack (désormais Schedule Shed)	Liste du personnel médical et planification	Amélioration des processus de travail.	Dépend actuellement de mises à jour manuelles peu fiables des systèmes de prise de rendez-vous lorsque le personnel n'est pas disponible.

F. Architectures

1. Architecture de la POC

La POC a été créé avec une architecture de micro-service qui vont interpréter les demandes des utilisateurs pour aller questionner la base de données et renvoyer les données dans un format compréhensible par l'utilisateur. Ci-dessous un schéma récapitulatif :



2. Architecture de données

Cette couche stocke et indexe les données de l'application nécessaire au bon fonctionnement de l'application. Pour la POC, cette base de données est stockée dans une instance PostgreSQL temporaire qui fonctionne dans un container Docker, créé spécifiquement pour alimenter la POC. Cette instance est remplie par un script SQL lancé à l'initialisation du container Docker. Lors de ce lancement, une instance de base de données Postgres SQL est créé et rempli avec le script puis peut être questionné par la POC.

3. Architecture d'applications

La POC est constitué de plusieurs couches de traitement :

- la couche « model » représente le modèle de données et permet également de constituer les objets utilisables par les autres couches.
- La couche «repository» qui gère les échanges entre la couche « model » et la couche « service » qui utilisera ces données.
- La couche « service » qui appelle les méthodes « util » pour effectuer les traitements sur les données.
- La couche « controller » qui permet les échanges avec l'utilisateur via des adresses URL personnalisées.

4. Architecture CI/CD

Afin d'assurer un déploiement plus rapide et controllé, le code de la POC a été publié sur GitHub pour faciliter le partage et la vérification du code. En effet, le projet étant développé en Test Driven Development (TDD), l'écriture des tests, correspondant à un besoin, sont définies en premières. Ces tests permettent de vérifier que toutes modifications seront validées et ne bloqueront pas l'application. Ces tests sont également lancées dans GitHub, dans la partie Action, afin de vérifier que le code est toujours fonctionnel après la dernière modification.

G. Scénario d'utilisation

Vois le document « README.txt » contenu dans le projet

H. Environnement

1. Matériel

La POC n'a été testé que dans 2 environnements pour le moment :

- l'ordinateur du concepteur qui possède un intel-i7 3517U avec 6Go de RAM tournant sous Linux
- le dépôt GitHub qui compile et lance les tests de l'application

2. Modèles d'architecture technologique

Dans cette section, nous verrons les différents outils technologiques préconisé pour la conception des futures vidéo interactives :

Langage

Le langage Java a été choisi pour créer cette POC. Cela permettra de trouver facilement des développeurs capables de reprendre et faire évoluer le code.

De plus, de nombreuses bibliothèques du langage Java ont été utilisés pour compléter et clarifier le code java, comme :

- Java Persistence API pour simplifier la conception de l'API
- Junit pour créer des tests unitaires
- Lombok pour automatiser la génération de certaines parties du code
- Driver PostgreSQL pour ls connexions avec la base de données
- Jacoco pour évaluer le taux de couverture de code

Framework

Le framework Spring Boot a été utilisé pour concevoir cette POC. Il offre plusieurs avantages :

- Architecture avec bibliothèque facilement implémentable grâce à Spring initializer
- Compatibilité large
- Robustesse éprouvée
- Large documentation et source d'information

- ***Outils diverses***

Enfin, de nombreux autres outils ont été utilisés pour concevoir l'application la plus évolutive et complète possible. De plus, ils permettent de tester l'application et avoir plus d'informations sur son comportement. En voici la liste :

- Maven pour compiler avec toutes les dépendances nécessaires
- JMeter pour effectuer des tests de charge et stress test
- Postman pour vérifier les réponses de l'API
- Docker pour créer des images de l'application et notamment de la base de données de test
- GitHub pour déposer le code et le tester

I. Risques et opportunités

1. Risques :

Les risques sont listés dans le document « **Analyse des risques** ».

2. SWOT

Ci-dessous le tableau SWOT regroupant les principaux paramètres de l'environnement :

	Avantages	Inconvénients
Facteurs internes	• Améliorer la vitesse des interventions et des retours des ambulances • Garantir un lit au malade dans l'hôpital cible • Mise en place d'un suivi et historique	• Mise en place longue • Changement d'habitude pour les salariés utilisateurs
	Opportunités	**Menaces**
Facteurs externes	• Echanges simplifiés entre les acteurs	• Normes à créer avec les partenaires

3. Gap Analysis

Le gap analysis est consultable dans le document « **Document de definition de l'architecture** ».

J. Axes d'améliorations :

1. Vitesse de réponses :

La contrainte d'une vitesse de réponse inférieure à 200 millisecondes n'a pas été respecté. La configuration modeste de la machine de développement ne permet pas de garantir les meilleures performances au bon fonctionnement de l'application. Ces tests devront être effectués sur un serveur de test avec une configuration plus fidèle à ce qu'un serveur peut fournir comme environnement de fonctionnement. Si à l'issue de ces tests le temps de réponse n'est toujours pas satisfaisant, plusieurs axes d'approche devront être mis en place :

- Faire un métrique des parties de code pour localiser les zones plus lentes et les améliorer

- Créer une instance de PostgreSQL en local et non dans une machine Docker

2. Modifier le GitHub actions pour récupérer les détails de réponses de tests

Dans cette configuration, l'application est livrée en 2 containers Docker contenant :

- une base de données PostgreSQL avec les données de test
- une image de l'application

Ces deux images sont administrées par un docker-compose qui peut lancer les images indépendamment. Ce fichier lance les test de l'application avant de la compiler et la lancer dans un container. Avec ne mode de fonctionnement, nous ne pouvons voir le résultat des tests en détail. Nous savons qu'ils ont réussi pour pouvoir lancer mais l'image mais GitHub n'a a pas accès aux logs.

Pour gagner en clarté dans les livraisons, il devrait être possible de ne lancer en container que la base de données et faire compiler le code par GitHub.

Sinon, on pourrait imaginer créer une base de données embarquée, créée et lancée par Maven au moment de la compilation.

1. Livrables

Pour cette POC, voici les livrables fournis en plus de ce document :

- Une hypothèse de validation de principe
- Un document de stratégie de test
- Un document sur les principes de l'architecture
- Un document listant les building blocks de la POC