Plateforme MedHead

Application POC pour le traitement des recommandations de lits dans les situations d'intervention d'urgence

ADD - Couche applicative

Environnement technique et structure du code

Version 1.0

Auteur : Hervé Prevost - Architecte Logiciel - MedHead

Date : 14/09/2023

Version : 1.0

SOMMAIRE

SOMMAIRE 2
OBJET ET CONTEXTE 3
OBJECTIF ET PÉRIMÈTRE DE LA POC 3
ENSEIGNEMENTS DE LA POC 3
ENVIRONNEMENT TECHNIQUE 4
Projet Java 4
H2 Database 4
STRUCTURE DU CODE 5
Couche Modèle 5
Couche Contrôleur 6
Couche Repository 7
Couche Service 7
Couche Sécurité 8

OBJET ET CONTEXTE

Ce document constitue un ajout au document ADD du dépôt GITHUB consacré à l'architecture du projet de plateforme MedHead

https://github.com/HPrevost/Software-Architect-P11/tree/master/artefacts/architecture/architecture-definition-document

Il permet de décrire plus en détail l'environnement technique et la structure du code de l'application de Proof Of Concept pour le traitement des recommandations de lits disponibles en situation d'urgence.

OBJECTIF ET PÉRIMÈTRE DE LA POC

Le code de la POC a consisté en la création d'une API minimale permettant l'interrogation du lit disponible le plus proche en fonction de la localisation géographique du patient et la spécialité médicale requise pour l'intervention d'urgence.

L'objectif de la POC était de préciser le cadre technique et confirmer la faisabilité d'un micro-service API permettant la consultation des lits disponibles en urgence sur le réseau d'établissements de santé Medhead.

ENSEIGNEMENTS DE LA POC

La POC n'a pas permis de valider l'ensemble des exigences et les métriques établis dans l'hypothèse de succès du fait de son exécution dans un contexte local et non sur un environnement de production. Elle a néanmoins prouvé sa réelle faisabilité technique sur le principe d'un micro-service en Backend de consultation de lits en urgence aisément partageable à l'ensemble du réseau Medhead et indépendant du dispositif de consultation. Elle a permis également de présenter les outils et méthodes utilisables pour le développement des futurs composants de la plateforme à travers la mise en place d'un pipeline CI/CD fonctionnel.

ENVIRONNEMENT TECHNIQUE

Projet Java

Le code de la POC a été réalisé en **Java** avec **Apache Maven** et **Spring Boot** dont voici les dépendances installées :

- **Spring Web** : permettant de créer une API RESTful et d'exposer des endpoints.
- **Lombok** : librairie pour optimiser certaines classes.
- **H2 Database** : la base de données
- **Spring Data JPA** : pour gérer la persistance des données avec la base de données. Les données du jeu d'essai sont stockées dans une base H2.

Le projet a été initialisé avec **Spring Initializer**.

L'IDE utilisé est **Eclipse**.

Les dépendances sont référencées dans le fichier **pom.xml** à la racine du projet.

src/main/resources/application.properties répertorie les variables de configuration du projet.

Lors du lancement de l'application, le serveur web **tomcat** intégré à spring est lancé en local sur le port 9000, permettant l'accès à l'API via l'URI **http://localhost:9000/patient/urgence**.

Le service est interrogé par un client **postman** installé en local utilisé avec les paramètres par défaut.

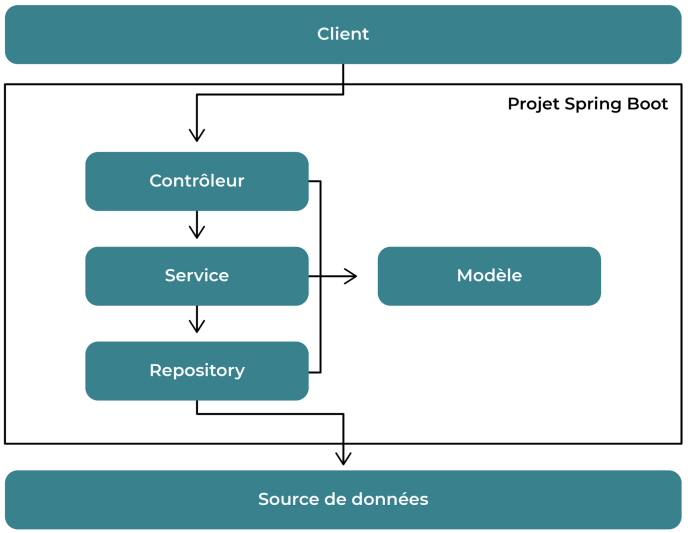
H2 Database

Les données du jeu d'essai créées sur la base H2 sont accessibles via la console :
http://localhost:9000/h2-console/login.jsp?jsessionid=635a15ac37e3bdf33286495b9bdaea55

L'id de session est dynamique et la configuration pour le test permet l'accès à la console sans authentification.

STRUCTURE DU CODE

La structure du code suit les recommandations du modèle en couches dont voici une illustration.



L'architecture du code est découpée en micro services. Pour faciliter les tests seul le micro-service d'interrogation d'urgence a été réalisé.

Le dossier de l'application de POC intitulé **p11api**. Le code est composé des fichiers suivants :

P11apiApplication.java : classe principale permettant d'éxécuter l'application

Couche Modèle

Les fichiers de classe suivants définissent les propriétés et méthodes nécessaires à la consultation et la mise à jour des différentes tables du modèle de données.

- **Etablissement.java** : pour l'entité établissement de santé (hôpital, clinique...). Nom, adresse et localisation GPS.
- **Specialite.java** : pour une spécialité de santé. Il est à noter qu'une spécialité est nécessairement rattachée à un groupe de spécialité même si celle-ci n'est pas utilisée dans le cadre de la POC.
- **Lit.java** : pour les lits appartenant à un établissement. Un lit est décrit par sa référence interne dans l'établissement, l'id de son établissement de rattachement et enfin l'id de la spécialité à laquelle

il est dédié. Il est à noter en effet que dans l'organisation des services de santé un lit est nécessairement rattaché à un service de soin et donc une spécialité.
- **Patient.java** : pour l'entité patient, regroupe ses informations d'identification (dont le no SS) et de contact. Cette table n'est pas utilisée dans le cadre spécifique de la POC, celle-ci ne gérant pour l'instant que la consultation des disponibilités et non la réservation.

Les autres classes ont été placées pour correspondre à la structure minimale requise pour la gestion de l'occupation des lits mais elles ne sont pas utilisées dans le cadre de la POC proprement dit. Il s'agit de :

- **GroupeSpecia.java** : classe définissant les propriétés et méthodes d'un groupe de spécialités. Selon la nomenclature NHS, les spécialités sont regroupées par groupe de spécialité, une recherche élargie aurait permis en cas d'indisponibilité de lit pour une spécialité donnée de trouver un lit dans le groupe auquel elle correspond.
- **Occupation.java** : classe regroupant les propriétés et méthodes des périodes d'occupation des lits au sein d'un établissement. Dans le cadre de la POC cette classe est inutilisée, c'est la table des lits qui remplira cette fonction. La table des lits ne contient que les lits disponibles à l'instant t. L'application centralisée MedHead devant permettre à chaque établissement d'insérer ou supprimer automatiquement un enregistrement lit dans cette table selon sa disponibilité.

Couche Contrôleur

Seuls deux contrôleurs ont été définis dans le projet :

- **EtablissementControleur.java** : Ce contrôleur gère l'ensemble des opérations CRUD nécessaires aux opérations de consultation et mise à jour d'un établissement. Il est donné à titre d'exemple mais n'est pas utilisé dans le cadre de la POC.
- **PatientControleur.java** : Ce contrôleur ne contient que les méthodes nécessaires au rapatriement des données utiles dans le cadre de la POC. C'est dans ce contrôleur qu'il a été choisi de placer la fonction principale faisant l'objet du test, à savoir la fonction **findLit** permettant la restitution des données du lit disponible le plus proche pour un patient en état d'urgence. Le contrôleur contient

également les getters correspondants aux champs spécifiques retournés par la requête (Réf du lit dans l'établissement, Nom et Coordonnées GPS de l'établissement). Il contient également le getter du message d'erreur éventuel.

Couche Repository

Seuls deux contrôleurs ont été définis dans le projet :

- **EtablissementRepository.java** : Ce repository est un simple interface implémentant le Crudrepository de l'établissement tel qu'automatisé par spring.
- **PatientRepository.java** : Ce repository est un interface de type JPARepository abritant la requête SQL utilisée pour la fonction findLit. Cette requête est un select comportant une jointure entre la table lit et etablissement afin de récupérer les trois champs décrits ci-dessus. La requête est filtrée sur l'id de la spécialité passée en paramètre et triée selon par ordre croissant des distances entre les coordonnées GPS de l'établissement et celles du patient. Par souci de simplification, les coordonnées GPS sont exprimées sous forme de simples entiers et le calcul de la proximité s'effectue par une soustraction entre ces deux valeurs (en valeur absolue). La requête ne retourne qu'un enregistrement au maximum.

Couche Service

Deux services et un interface ont été implémentés pour les besoins du POC :

- **EtablissementService.java** : gère la récupération de tous les établissements ou d'un établissement selon son id passé en paramètre, la suppression d'un établissement dont l'id est passé en paramètre et la mise à jour d'un établissement qu'il s'agisse de sa création ou sa modification. Ce service n'est pas utilisé dans le cadre de la POC.
- **FindLitResult.java** : interface spring généré pour récupérer les champs individuellement de la requête contenue dans findLit qui figure dans le fichier PatientService.java. Un getter pour le message d'erreur éventuel successif à l'exécution de la requête a été ajouté à cette interface.
- **patientService.java** : Retourne la fonction findLit définie dans le repository patient avec l'interface FindLitResult décrit ci-dessus.

Couche Sécurité

Une classe concentre les éléments de configuration http :

- **SecurityConfiguration.java** : Cette classe filtre les requêtes en n'autorisant l'accès qu'à certains répertoires de l'application à savoir h2-console/ et api/ pour la console H2 et le service REST patient/urgence/ pour postman.