Exigences de la PoC

Projet allocation de lits d'hôpital pour les urgences

Table des matières

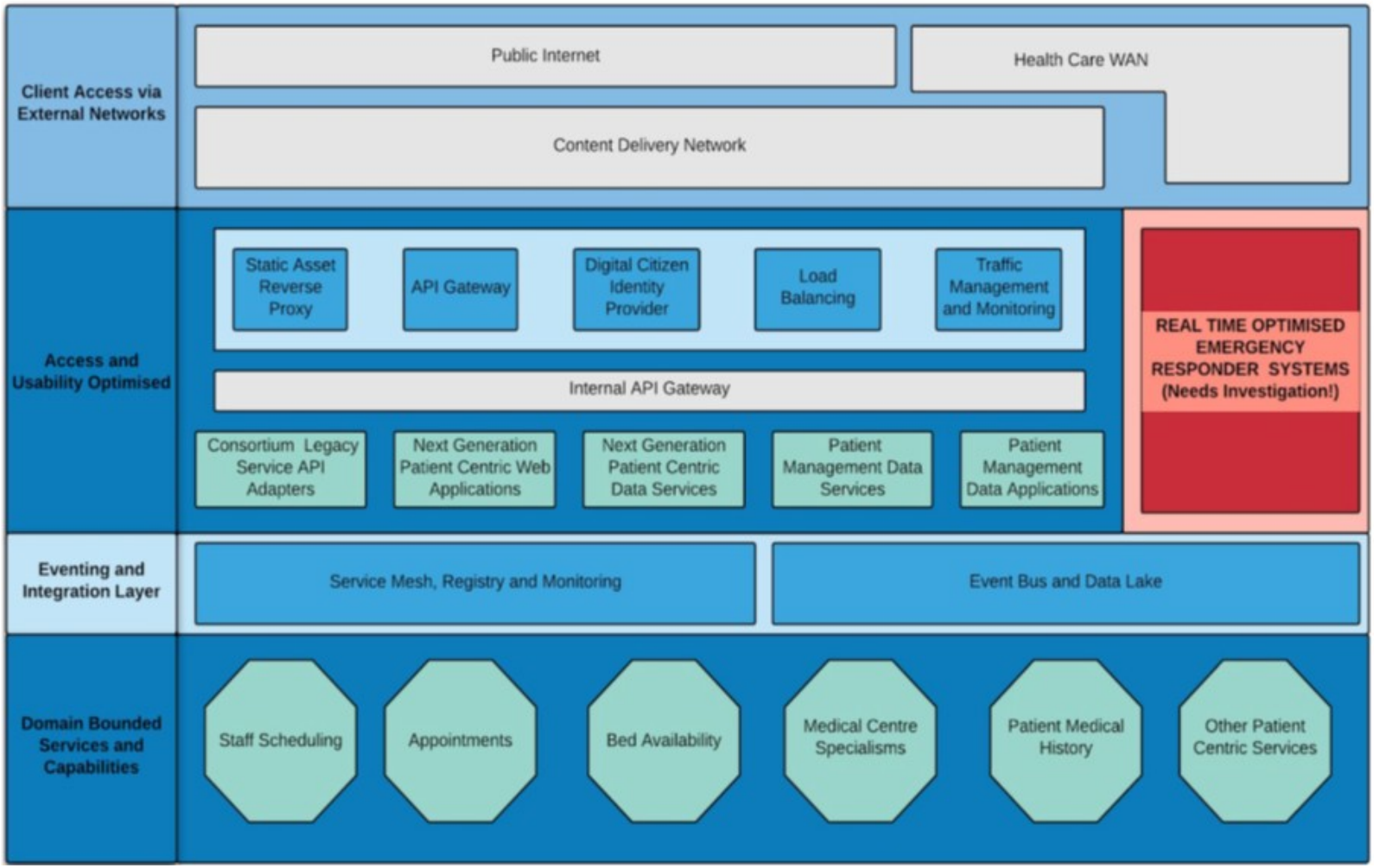
Introduction

Ce document a pour objectif de rappeler les différentes exigences convenue de la PoC à mettre en place.

De plus, il décrira les différents composants techniques utilisés tout au long de la PoC

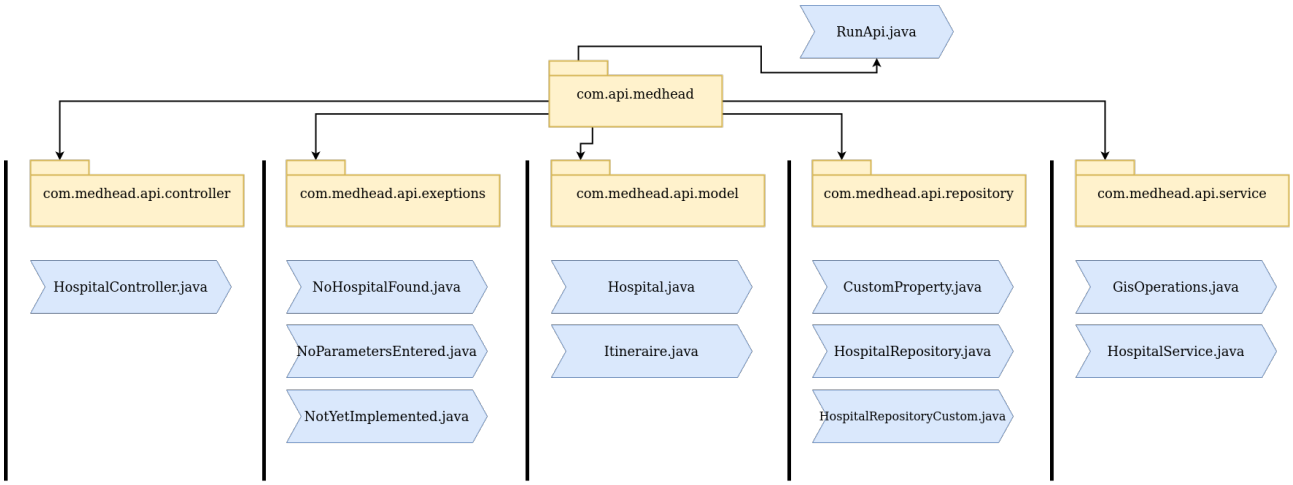
Les exigences convenues de la PoC peuvent être trouvées dans le document « *Hypothèse de développement* » présent dans le dépôt d’architecture.

Architecture cible



Architecture de la PoC

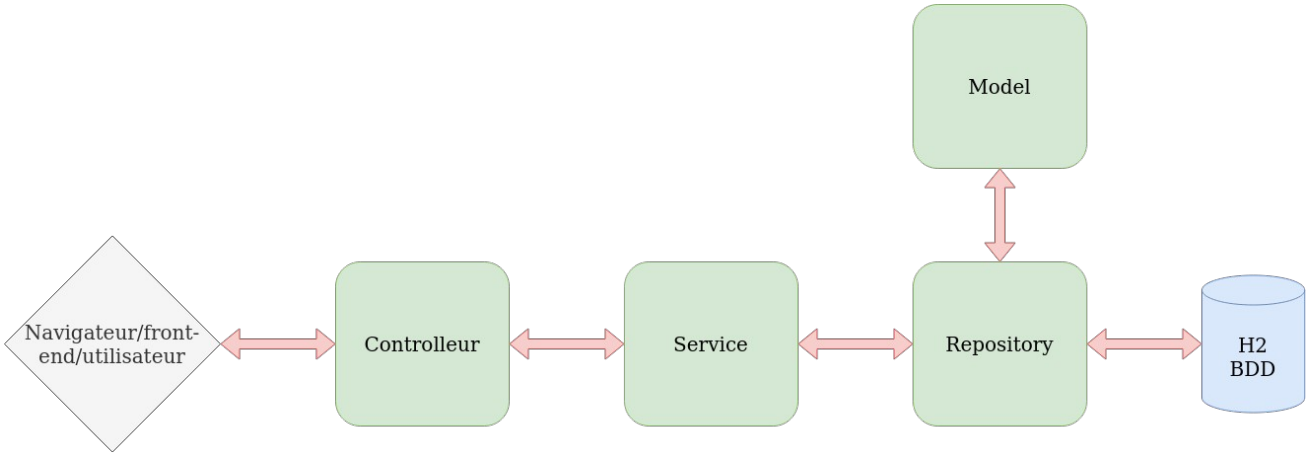
Dans le cadre du POC, différentes hypothèses de développement ont été énoncées et devront être respecté pour le sous-système d'intervention d'urgence en temps réel.



L'application est développée selon une architecture micro-service.

Elle respecte des bonnes pratiques de développement, à savoir le modèle MVC dans notre cas, avec les classes suivantes :

- *HospitalController.java*
- *Hospital.java*
- *HospitalRepository*
- *HospitalService*



Chaque classe est placée dans son package, les packages étant inclus dans un package « parent ».

À ces classes s’ajoute des classes de gestion des différentes exceptions, chacune d’elle pouvant recevoir un message personnalisé.

- *NoHospitalFound.java*
- *NoParametersEntered.java*
- *NotYetImplemented.java*

Exigences convenues

API RESTful

« *Une API REST, ou RESTful, est une interface de programmation d'application (API ou API web) qui respecte les contraintes du style d'architecture REST (Representational State Transfer) et permet d'interagir avec les services web RESTful.* » - Redhat.com

Une API RESTful doit respecter un ensemble de contraintes architecturales. RESTful n'est pas une norme ou un protocole particulier.

Une API RESTful doit remplir les critères suivants :

- Une architecture client-serveur, possédants des clients, des serveurs, des ressources à accéder, et les communications doivent être faites par des requêtes HTTP
- Être « Stateless ». C'est-à-dire que chaque processus est indépendant. Le processus ne stocke pas de données et n'a pas de référence avec les transactions passées
- Des données peuvent être mises en cache
- Une interface entre les composants permet un transfert des informations

Sécurisation des données

Les données du patient ne sont pas utilisés dans le cadre de la PoC. De ce fait, ce point précis n'a pas été géré, la PoC se focalisant sur le traitement de l'allocation des lits d'hôpitaux et la réponse au besoin de gain de vitesse.

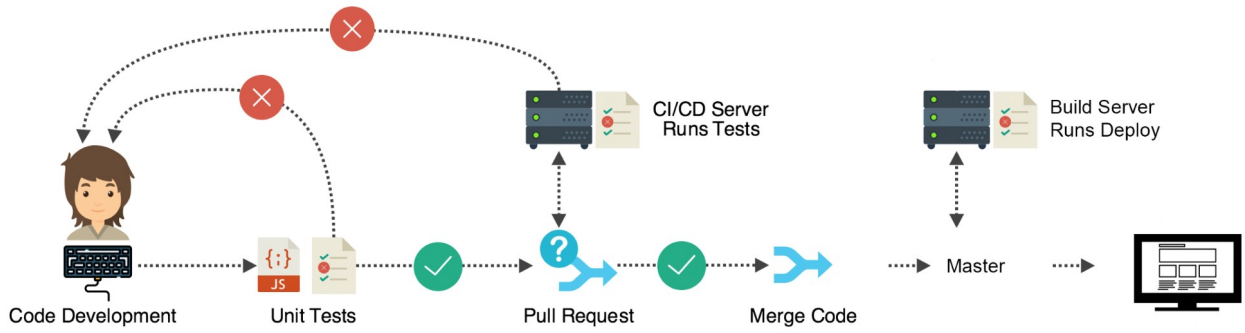
Cependant, l'usage de technologies comme Auth2 ou des tokens JWT fourni par Spring Boot Secure sont à prévoir pour permettre d'améliorer la sécurité de l'authentification et l'utilisation systématique de protocoles sécurisés (dont HTTPS) pour le transfert des données pour éviter tout risque de vol de données par écoute active ou passive du réseau, ou attaque de type Man In the Middle.

Tests

Se référer au document « *Document de stratégie de test* » présent dans le dépôt d'architecture.

Intégration

En plus de la fonctionnalité demandée, la PoC démontre une capacité d'intégration pour les développements futurs. En effet, la présence du code sur un dépôt accessible par tous les développeurs du projet permet un travail collaboratif.



De plus, l'utilisation de pipeline d'intégration et de déploiement continue (CI/CD), ainsi qu'une stratégie de tests documentée permet une intégration simplifiée et plus complète dans les développements futurs.

Modularité

L'usage de SpringBoot dans une architecture en micros-services permet une première couche de modularité.

Par définition, une architecture en micros-services service permet de déployer des applications constituées de services aux fonctionnalités complémentaires.

Chaque micros-services possède sa propre base de donnée et est indépendant des autres micros-services. Le micro service peut être vu comme un bloc de construction s'incluant dans une application au côté d'autres blocs.

De plus, la modularité est aussi assurée dans l'absence d'obligation d'utiliser un langage en particulier pour les micros-services. De ce fait, le projet est développé actuellement en Java, avec

l'utilisation du framework SpringBoot, mais pourrait à l'avenir faire l'objet d'un développement dans un autre langage.

Enfin, la structure du projet, avec une séparation des modèles, des services, des contrôleurs, repository ou encore la gestion des exceptions permet une modularité par sa conception.

Validation des principes d'architecture

Se référer au document « *validation des principes d'architecture* »

Intégration et déploiement continue

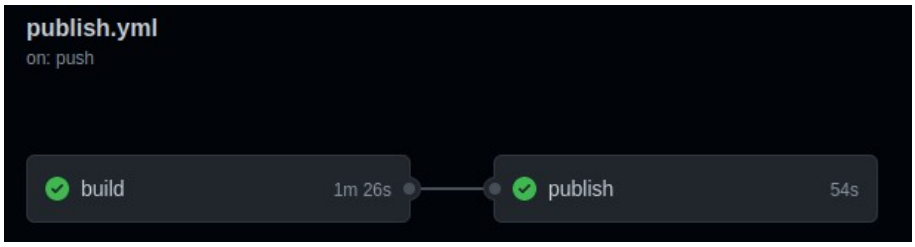
Un process d'intégration et de déploiement continu est mis en place pour cette PoC.
Il a été choisi de fonctionner, pour cette preuve de concept, avec l'outil de pipeline CI/CD intégré à Github : Github Actions.

Au niveau du dépôt de code, chaque pipeline est représentée par un fichier yaml présent dans un dossier *.github/workflows/*

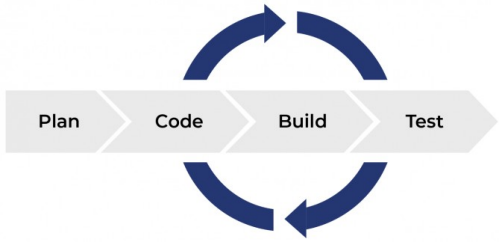
Le fichier *publish.yml* y décrit, dans notre PoC, les actions d'intégration et de déploiement continue, chacun faisant l'objet d'un job :

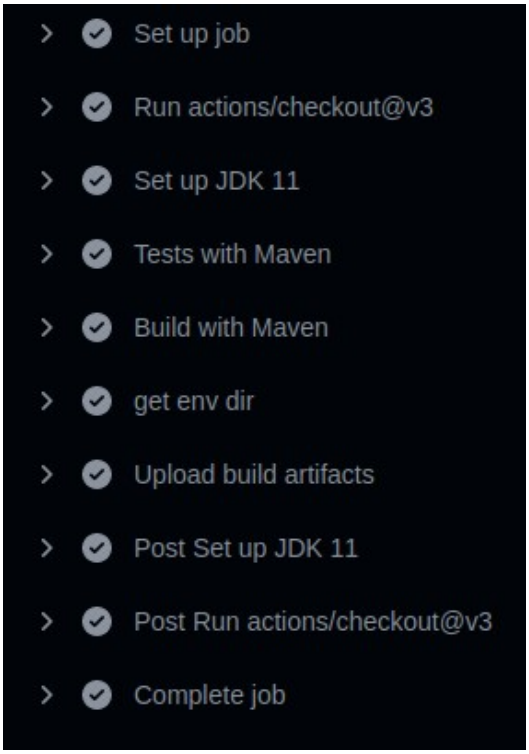
- Intégration Continue : Job « Build »
- Déploiement Continue : Job « Publish »

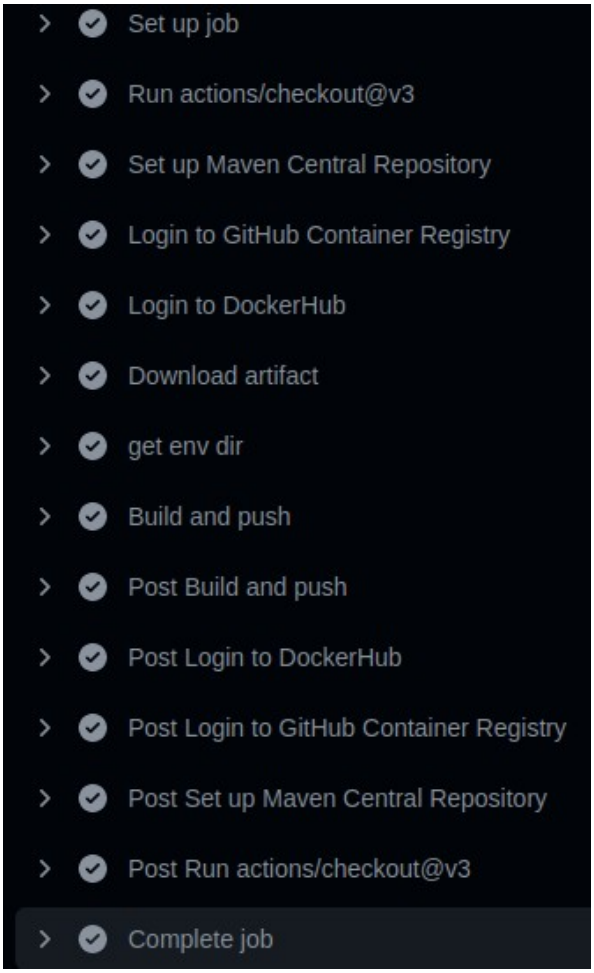
Ces jobs sont exécutés au moment du *git push*, dans le cas de notre PoC, mais pourrait être configuré pour une exécution au moment des pull-requests.



Chaque job est constitué d'étapes (steps) décrivant le déroulement des actions.







Dans le cas de la PoC, il a été fait le choix d'un fonctionnement simple, avec l'utilisation le plus possible d'actions préexistante :

<createur>/<actionname>@<version>	Usage
actions/checkout@v3	• Permet l'accès au depôt GitHub pour le workflow
actions/setup-java@v3	• Téléchargement et configuration d'une version de java • Configuration du runner pour la publication à l'aide d'apache Maven • Mise en cache des dépendances pour Maven
actions/upload-artifact@v3	• Permet de partager les données entre les jobs et de sauvegarder l'artefact quand le workflow est terminé
docker/login-action@v2	• Connexion à un registry docker
actions/download-artifact@v2	• Télécharge les artefacts du build
docker/build-push-action@v2	• Créé une image docker • Upload de l'image dans le registry privé

Les actions GitHub peuvent être trouvées sur le marketplace : https://github.com/marketplace

Un versionning est mis en place dans le nom de l'artefact via la nomenclature suivante :

pocApiMedHead-0.0.1-SNAPSHOT-v${{github.run_number}}.jar

Et pour l'image docker :

etcomment/ms1:v${{github.run_number}}

L'utilisation du numéro de run (variable *github.run_number*) pour la génération de l'artefact permet de bien séparer les items pour chaque build et faciliter des retours en arrière, des tests de non régression ou encore des tests comparatifs.

Nous pourrions très bien utiliser la variable env.GITHUB_REF_NAME par exemple, qui nous ferait avoir un artefact avec le nom de la branche qui execure le workflow.