

# Document de Reporting

## Table des matières

1. Introduction.....	1
<b>1.1. Contexte du projet</b> .....	1
2. Justification de l'environnement technologique.....	2
<b>2.1. Gradle</b> .....	2
<b>2.2. Spring Boot</b> .....	2
<b>2.3. Java 17</b> .....	3
<b>2.4. Utilisation de RowMapper avec JdbcTemplate</b> .....	3
<b>2.5. Classes métiers</b> .....	3
<b>2.6. SGBDR SQLite</b> .....	4
<b>2.7. SpringDoc-OpenAPI</b> .....	4
<b>2.8. Frameworks JavaScript/TypeScript</b> .....	4
3. Respect des normes et principes.....	4
4. Résultats et enseignements de la PoC .....	6
5. Automatisation des builds avec GitHub Actions .....	7
6. Déploiement simplifié (utilisation de Docker).....	8
7. Conclusion .....	8

<b>Auteur</b>	BENTZ Maxime
<b>Version du document (Version.Révision)</b>	V1.6
<b>Date de dernière modification</b>	23/11/2023

## 1. Introduction

**Projet :** Preuve de concept pour le système d'intervention d'urgence en temps réel

**Equipe :** Architecture métier du consortium MedHead

### 1.1. Contexte du projet

MedHead est un regroupement de grandes institutions médicales œuvrant au sein du système de santé britannique et assujéti à la réglementation et aux directives locales (NHS). Les organisations membres du consortium utilisent actuellement une grande variété de



technologies et d'appareils. Ils souhaitent une nouvelle plateforme pour unifier leurs pratiques. La technologie Java est pour eux un socle technique fiable pour ce projet.

## 2. Justification de l'environnement technologique

### 2.1. *Gradle*

La décision d'utiliser Gradle comme outil de gestion de dépendances et de construction de projet est basée sur plusieurs avantages clés qu'il offre par rapport à des alternatives comme Maven :

- Flexibilité : Gradle est conçu pour être extrêmement adaptable, ce qui le rend idéal pour les projets qui ont des besoins de construction non conventionnels ou complexes.
- Performance : Grâce à la mise en cache et à l'incrémental build, Gradle (bien que disponible sous Maven avec l'ajout d'un Addon séparé) peut offrir des temps de construction plus rapides que Maven, ce qui peut être intéressant pour générer des exécutables plus rapidement.
- DSL Groovy/Kotlin : Gradle utilise Groovy ou Kotlin comme langage pour son DSL (Domain Specific Language), ce qui rend les fichiers de configuration beaucoup plus lisible que le XML volumineux utilisé par Maven.
- Extensibilité : Etant basé sur des plugins, il est simple d'ajouter des fonctionnalités à Gradle. La communauté active a créé une multitude de plugins qu'on pourra ajouter très simplement à notre projet. En tant que développeur .NET, cela se rapproche beaucoup du concept de packages Nuget qu'on retrouve dans cet environnement.

### 2.2. *Spring Boot*

Spring Boot a été choisi comme Framework de développement pour sa capacité à simplifier la configuration, faciliter le développement d'applications Web Java et offrir une large gamme de modules intégrés afin de répondre à des besoins spécifiques (sécurité, liaison avec un SGBDR, etc).

**Dans notre cas, nous utiliserons les dépendances Spring Boot suivantes :**

'org.springframework.boot:spring-boot-starter-web:3.0.4'
'org.springdoc:springdoc-openapi-starter-webmvc-ui:2.0.3'
'org.springframework.boot:spring-boot-starter-security:3.0.4'
'org.springframework.boot:spring-boot-starter-test:3.0.4'
'org.springframework.boot:spring-boot-starter-jdbc:3.0.4'

### 2.3. Java 17

Une version stable et récente de Java a été utilisée pour assurer une compatibilité optimale avec les nouvelles fonctionnalités propres au Framework Java et bénéficier des dernières améliorations en matière de performance et de sécurité. Nous pouvons interchanger entre OpenJDK OpenJ9 et OpenJDK Hotspot pour build et exécuter notre application.

### 2.4. Utilisation de RowMapper avec JdbcTemplate

Dans les différents services, JdbcTemplate est utilisé pour effectuer des opérations CRUD sur la base de données. L'une des caractéristiques essentielles de JdbcTemplate est l'utilisation de RowMapper pour mapper les lignes d'une requête SQL sur des objets.

Le RowMapper est utilisé pour convertir chaque ligne de la ResultSet SQL en un objet de l'entité Hopital. Cela nous est particulièrement utile lorsque nous avons plusieurs lignes dans le ResultSet et que nous souhaitons les convertir en une liste d'objets.

Exemple d'utilisation :

```
private static final RowMapper<Hopital> rowMapper = (rs, rowNum) ->
    new Hopital(rs.getInt("Id"), rs.getString("Nom"), rs.getString("Adresse"),
        rs.getString("Ville"), rs.getString("CodePostal"),
        rs.getInt("LitsDisponibles"));
```

### 2.5. Classes métiers

Les différentes données métiers sont représentés sous formes de classes afin de manipuler nos données plus efficacement dans notre code.

<b>Coordinate</b>	<p>Cette classe représente une coordonnée géographique avec une latitude (lat) et une longitude (lng).</p> <p>Elle est utilisée pour localiser les hôpitaux ou les patients sur une carte et permettre de renvoyer l'établissement le plus proche en fonction d'un besoin.</p>
<b>Hopital</b>	<p>Cette classe représente un hôpital avec des détails tels que son nom, son adresse, le nombre de lits disponibles, etc.</p> <p>Elle contient également une liste de spécialisations de manière à ce que chaque hôpital ai ses propres spécialités médicales.</p>
<b>Patient</b>	<p>La classe Patient contient des informations sur un patient, y compris son nom, sa date de naissance, son adresse, etc. Elle inclut également une propriété specialiteRequise, qui représente la spécialisation dont le patient a besoin.</p>



<b>Specialisation</b>	Cette classe représente donc une spécialisation médicale, qui pourrait être quelque chose comme la cardiologie, la neurologie, etc.  Elle contient un id pour identifier de manière unique chaque spécialisation et un libelle pour décrire la spécialisation.
-----------------------	--

## 2.6. *SGBDR SQLite*

Utilisé comme base de données légère et intégrée, ce qui facilite le déploiement de la preuve de concept.

## 2.7. *SpringDoc-OpenAPI*

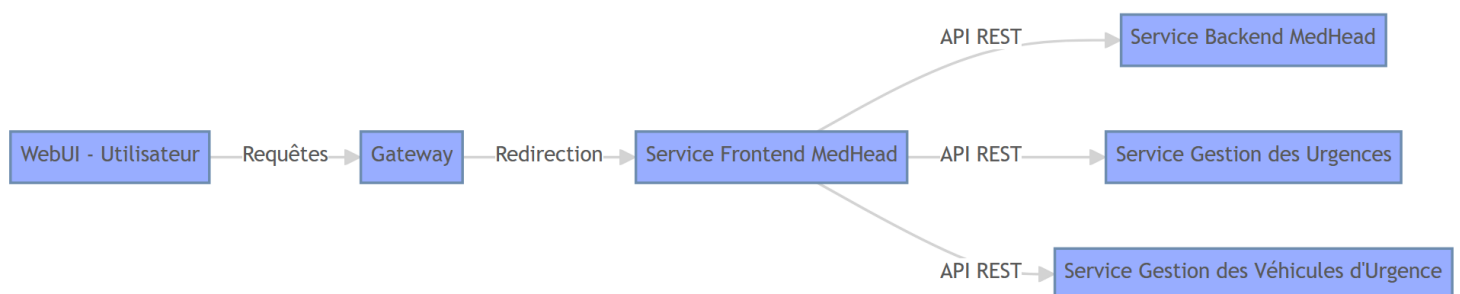
Nous utiliserons SpringDoc OpenAPI pour fournir une documentation interactive Swagger de l'API, permettant une meilleure visualisation et des tests d'intégration faciles

## 2.8. *Frameworks JavaScript/TypeScript*

Angular, React ou VueJS sont les options recommandées pour l'interface utilisateur car ils sont populaires, bien soutenus et flexibles. Dans notre cas, nous utiliserons VueJS (Vue 3.X).

## 3. Respect des normes et principes

**Architecture Microservice** : En utilisant un backend Spring Boot dissocié encapsulable dans un conteneur Docker, nous avons toutes les bases pour développer une architecture microservice dans le futur. De plus, l'application est modulaire, ce qui signifie qu'elle peut être décomposée en petits services autonomes à l'avenir.



*Exemple d'architecture Microservice pouvant résulter de notre développement, il s'agit d'une suggestion d'implémentation parmi de nombreuses possibilités, il est également possible d'ajouter un orchestrateur pour répliquer les services et gérer notamment la montée en charge.*



**RGPD** : Les données du patient sont protégées conformément à la norme RGPD. Le module spring-boot-starter-security est intégré pour assurer la sécurité des données et des transactions.

**Sécurité** : Il est totalement possible d'utiliser un [Cerfbot](#) (Let's Encrypt) pour chiffrer les échanges de données à tout niveau de notre application que ce soit entre l'API et le Frontend ou le Frontend et les clients. En effet, il est totalement possible de développer un petit script (bash ou cmd pour automatiser le renouvellement du certificat SSL). Pour des raisons de simplicité, sécurité et car il est nécessaire d'avoir un nom de domaine valide, aucun certificat SSL n'est délivré avec les applications sur le Git.

**Pyramide des tests** : L'intégration de tests à différents niveaux est essentielle pour garantir la qualité du logiciel. Un projet de tests unitaire a été conçu et vise à valider le bon fonctionnement du backend. En parallèle, un stress test visant à vérifier les performances et le support de montée en charge d'une instance de l'API a également été implémenté **côté Frontend** et n'a pas été inclus dans les tests unitaires. En effet, pour avoir une idée réaliste des performances de l'API, il est nécessaire de se mettre dans une situation concrète où les requêtes sont effectuées depuis un micro-service indépendant du Backend. Concernant les tests unitaires, ceux-ci sont implémentés dans le workflow CI/CD et sont nécessaires pour obtenir build de notre application.

Une page d'état a également été conçue sur notre Frontend afin de pouvoir suivre le bon fonctionnement de notre API en temps réel. Cette dernière vise à confirmer que l'application fonctionne bien mais ne valide en aucun cas chaque méthode de notre API.

## État des Endpoints

Statut	URL	Temps de Réponse Moyen
Online	http://localhost:8080/api/patient	33.50 ms
Online	http://localhost:8080/api/hospital	422.60 ms
Online	http://localhost:8080/api/spécialisation	9.50 ms
Online	http://localhost:8080/api/emergency/getdisponibility/1	12.20 ms

## Test de Stress de l'API

Tester la réactivité de l'API avec 800 requêtes simultanées.

Lancer le Test de Stress

### Résultats du Test

Temps de réponse moyen : 2.67 ms

#### Console de Test

```
[21:44:10] Début de l'exécution des tests...
[21:44:10] Requête pour patient ID 1, temps de réponse: 125.80 ms
[21:44:10] Requête pour patient ID 1, temps de réponse: 128.30 ms
[21:44:10] Requête pour patient ID 1, temps de réponse: 146.30 ms
[21:44:10] Requête pour patient ID 1, temps de réponse: 148.60 ms
[21:44:10] Requête pour patient ID 1, temps de réponse: 149.00 ms
```

**Intégration et livraison continue (CI/CD)** : Les pipelines CI/CD sont prévus pour automatiser le déploiement et les tests.




## 4. Résultats et enseignements de la PoC

**Performance et Efficacité** : L'application a été pensée pour être rapide et efficace et ainsi propose des temps de réponse moindres afin de pouvoir répondre aux besoins des urgentistes. En effet, en dehors du choix technologique, les méthodes permettant à l'API de catégoriser les hôpitaux par distances (méthode `getDistance` dans la classe `MapEngine`) propose une mise en cache des coordonnées afin de réduire le nombre d'appels à des services extérieurs.

**API RESTful** : Une API REST est fournie, comme défini dans le cahier des charges, avec des endpoints spécifiques pour obtenir des informations sur les hôpitaux et leurs disponibilités, les spécialisations, les patients, le matching patient / établissement de soin, etc. De plus, toutes les méthodes de CRUD ont été implémentées afin de pouvoir manipuler les données à notre guise.

**Interaction avec des services externes** : La méthode `getCoordinates` de la classe `MapEngine` interagit avec une API externe pour obtenir des coordonnées à partir d'une adresse, démontrant la capacité du système à intégrer des services tiers.

**Documentation Swagger** : La documentation Swagger intégrée offre une interface interactive pour tester l'API. Swagger permet notamment de générer une documentation basée sur nos méthodes et nous permet de tester simplement ces dernières au travers d'une page complète et détaillée.



Explorer

## API de Gestion des Interventions d'Urgence 1.0 SV3

[/medhead-api-docs](#)

Cette API expose des points de terminaison pour gérer les interventions d'urgence.

[Conditions d'utilisation](#)  
[Consortium MedHead - Site Web](#)  
[Envoyer un e-mail au Consortium MedHead](#)  
[Licence spécifique](#)

**Les serveurs**

### Hôpital Endpoint de gestion des hôpitaux.

- OBTENIR** `/api /hôpital /{id}` Récupère un hôpital par ID
- METTRE** `/api /hôpital /{id}` Mise à jour d'un hôpital par ID
- SUPPRIMER** `/api /hôpital /{id}` Supprimer un hôpital par ID
- OBTENIR** `/api /hôpital` Récupère la liste des hôpitaux
- POSTER** `/api /hôpital` Ajout d'un nouvel hôpital
- OBTENIR** `/api /hôpital /getHospitalByDistance /{distance}` Récupère les hôpitaux dans un rayon spécifique

### Patient Endpoint de gestion des patients

- OBTENIR** `/api /patient /{id}` Récupère un patient à l'aide de son identifiant
- METTRE** `/api /patient /{id}` Met à jour un patient
- SUPPRIMER** `/api /patient /{id}` Supprime un patient

**Gestion des erreurs :** Les réponses HTTP appropriées sont gérées dans le contrôleur, par exemple en renvoyant une erreur 404 lorsque les données requises ne sont pas trouvées.

## 5. Automatisation des builds avec GitHub Actions

L'automatisation des builds est un aspect indispensable du workflow CI/CD pour garantir que chaque modification soit vérifiée et validée de manière continue. Dans cette optique, nous avons adopté GitHub Actions comme plateforme d'intégration continue (CI) ce afin de centraliser au maximum le projet au travers d'un seul outil.

**Intégration Continue :** Chaque fois qu'une nouvelle modification est poussée sur notre dépôt GitHub, GitHub Actions déclenche automatiquement un build, assurant ainsi que le code est toujours fonctionnel.

**Tests Automatisés :** Au-delà de la simple compilation, notre pipeline CI exécute également l'ensemble de nos tests, garantissant que les nouvelles modifications n'introduisent pas de régressions.

**Notifications :** En cas d'échec du build ou des tests, GitHub envoie une notification aux contributeurs du repos qui sont donc immédiatement informés, permettant une réaction rapide pour résoudre les problèmes.



**Flexibilité** : GitHub Actions offre une grande flexibilité pour définir des workflows complexes, répondant à tous nos besoins spécifiques d'automatisation.

L'utilisation de GitHub Actions simplifie notre processus de développement (même par rapport à d'autres plateformes CI), et garantit une livraison continue et stable.

## 6. Déploiement simplifié (utilisation de Docker)

Dans le cadre de notre projet PoC, Docker fait partie intégrante de notre écosystème de développement et de déploiement, ce pour plusieurs raisons :

**Environnement homogène** : Docker garantit que l'application fonctionne de la même manière dans tous les environnements, qu'il s'agisse d'un ordinateur de développeur, d'un environnement de test ou d'un serveur de production.

**Isolation** : Notre application MedHead et ses dépendances sont emballées dans un conteneur Docker isolé, afin d'éviter tous conflits avec un écosystème applicatif existant et garantir une sécurité renforcée.

**Portabilité** : Les conteneurs Docker peuvent être facilement déplacés entre les environnements, ce qui facilite les tests et le déploiement.

**Intégration avec des outils modernes** : Docker s'intègre également très bien avec des outils tels que Kubernetes, ce qui nous permet une orchestration et une mise à l'échelle faciles de notre application MedHead.

**Optimisation des ressources** : Contrairement aux machines virtuelles, les conteneurs Docker partagent le même OS kernel, réduisant ainsi la consommation de ressources.

L'adoption de Docker pour notre projet nous permet de développer, tester et déployer notre application de manière plus efficace, rapide et sécurisée.

## 7. Conclusion

L'environnement technique du projet PoC pour le système d'intervention d'urgence répond aux exigences attendues. L'application est réactive, le déploiement et la mise en place peut-être scalable, et fournit une API efficace et documentée pour s'intégrer avec d'autres Front comme une application mobile.