Document de reporting – PoC MedHead

Table des matières

1. Contexte et enjeux du projet MedHead

1.1 Présentation du consortium MedHead

Le consortium MedHead regroupe plusieurs grandes institutions médicales opérant au sein du système de santé britannique (National Health Service – NHS). Ces établissements évoluent dans un environnement fortement réglementé et critique, où la qualité, la fiabilité et la rapidité des systèmes d'information ont un impact direct sur la prise en charge des patients.

Historiquement, les organisations membres du consortium utilisent des systèmes d'information hétérogènes, développés à différentes périodes et reposant sur des technologies variées. Cette hétérogénéité complexifie l'interopérabilité des systèmes, limite la visibilité globale sur les capacités hospitalières et rend difficile la mise en œuvre de services transverses à l'échelle du consortium.

Dans ce contexte, MedHead souhaite initier la conception d'une plateforme unifiée permettant d'harmoniser les pratiques et de soutenir les processus critiques liés à la gestion des situations d'urgence médicale.

1.2 Problématique métier liée aux interventions d'urgence

Lors d'une situation d'urgence, la rapidité de décision est un facteur déterminant pour la qualité de la prise en charge des patients. L'un des enjeux majeurs identifiés par le consortium concerne la capacité à recommander rapidement un établissement hospitalier adapté à une spécialité médicale donnée et disposant de lits disponibles.

En l'absence d'un système centralisé et fiable, les services d'urgence peuvent être confrontés à plusieurs difficultés :

- manque de visibilité en temps réel sur la disponibilité des lits ;
- orientation vers des établissements inadaptés à la spécialité requise ;
- saturation de certains hôpitaux, au détriment d'autres établissements disposant de capacités disponibles ;
- perte de temps critique lors des phases de décision.

La problématique ne se limite pas à la simple disponibilité des lits. Elle inclut également la notion de **distance et de temps de trajet**, éléments essentiels dans un contexte d'intervention d'urgence où chaque minute compte.

1.3 Inquiétudes et attentes des parties prenantes

Lors des échanges avec le comité d'architecture et les partenaires institutionnels du consortium, plusieurs inquiétudes majeures ont été exprimées concernant la future plateforme MedHead.

Les parties prenantes souhaitent notamment s'assurer que :

- le système reste fonctionnel et performant lors des périodes de forte activité (pics d'appels d'urgence) ;

- l’architecture retenue soit robuste, évolutive et conforme aux principes définis par le consortium ;
- les choix technologiques reposent sur des solutions éprouvées, adaptées à un environnement critique comme celui du NHS ;
- la plateforme puisse s’inscrire à terme dans une architecture plus large, potentiellement orientée événements.

Ces inquiétudes ont conduit à la décision de prioriser la réalisation d’une preuve de concept (PoC) afin de valider les choix d’architecture et de réduire les risques techniques avant d’envisager un déploiement à plus grande échelle.

1.4 Objectifs de la preuve de concept

La preuve de concept réalisée dans le cadre du projet MedHead poursuit plusieurs objectifs complémentaires.

D’un point de vue fonctionnel, elle vise à démontrer la capacité du système à recommander un établissement hospitalier en tenant compte :

- de la spécialité médicale requise ;
- de la disponibilité des lits ;
- de la distance et du temps de trajet estimés.

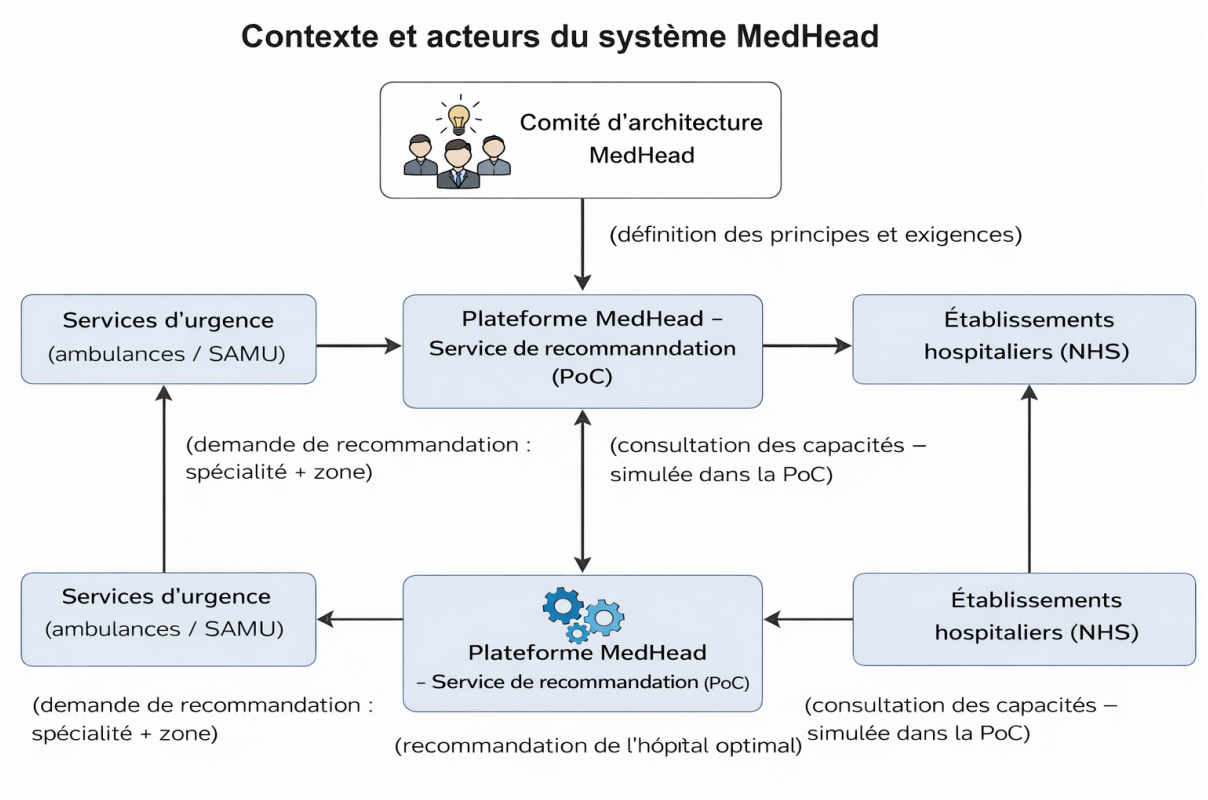
D’un point de vue technique, la PoC a pour objectif de :

- valider l’utilisation de la technologie Java, imposée comme socle technique par le consortium ;
- démontrer l’intégration d’un backend exposant des API REST dans une architecture de type microservices ;
- mettre en œuvre un frontend léger permettant la démonstration fonctionnelle du service ;
- garantir la qualité du code grâce à des tests automatisés ;
- évaluer la capacité du backend à supporter une montée en charge à l’aide de tests de performance.

Enfin, la PoC a également pour vocation de produire des enseignements utiles pour la suite du projet, tant sur les choix technologiques que sur l’organisation du développement et des pratiques d’intégration continue.

Le périmètre de cette preuve de concept est volontairement limité afin de se concentrer sur les risques majeurs identifiés, tout en restant aligné avec l’architecture cible définie par le consortium MedHead.

Figure 1 – Contexte et acteurs du système MedHead



2. Analyse des exigences et périmètre de la preuve de concept

2.1 Analyse des exigences fonctionnelles

Les exigences fonctionnelles de la preuve de concept ont été définies à partir du document des exigences fourni par le consortium MedHead. Elles visent à couvrir les cas d'usage critiques liés à la gestion des interventions d'urgence, tout en restant dans un périmètre volontairement limité.

Les exigences fonctionnelles retenues pour la PoC sont les suivantes :

Référence	Exigence fonctionnelle	Description
EF-01	Sélection par spécialité	Le système doit permettre de sélectionner une spécialité médicale issue des référentiels NHS.
EF-02	Disponibilité des lits	Le système doit prendre en compte uniquement les établissements disposant de lits disponibles (> 0).
EF-03	Recommandation d'hôpital	Le système doit recommander l'hôpital le plus pertinent en fonction des critères définis.

EF-04	Prise en compte de la distance	La recommandation doit intégrer une notion de distance et de durée de trajet.
EF-05	Réservation d'un lit	Le système doit permettre la réservation d'un lit et la mise à jour de la disponibilité.
EF-06	Interface de démonstration	Une interface web simple doit permettre de démontrer le fonctionnement de la PoC.

Ces exigences ont été implémentées de bout en bout, depuis l'interface utilisateur jusqu'au backend exposant les API REST.

2.2 Analyse des exigences non fonctionnelles

Au-delà des fonctionnalités, la mission met fortement l'accent sur les exigences non fonctionnelles, en particulier la robustesse, la performance et la qualité du code.

Les exigences non fonctionnelles retenues sont les suivantes :

Référence	Exigence non fonctionnelle	Description
ENF-01	Performance	Le service doit répondre rapidement, y compris en période de forte charge.
ENF-02	Scalabilité	L'architecture doit pouvoir s'inscrire dans une approche microservices.
ENF-03	Testabilité	Le code doit être couvert par des tests automatisés.
ENF-04	Fiabilité	Les erreurs doivent être maîtrisées et identifiées.
ENF-05	Traçabilité	Les modifications doivent être versionnées et intégrées via un pipeline CI.

Ces exigences ont guidé les choix techniques et l'organisation du développement de la PoC.

2.3 Correspondance exigences ↔ réponses apportées par la PoC

Le tableau ci-dessous synthétise la manière dont la preuve de concept répond aux exigences identifiées.

Exigence	Réponse apportée par la PoC
EF-01	Utilisation d'un référentiel de spécialités NHS chargé côté backend (exposé via API).
EF-02	Filtrage des établissements disposant de lits disponibles dans le backend.
EF-03	Algorithme de recommandation basé sur la spécialité, la disponibilité et la durée de trajet.
EF-04	Intégration d'un service de routage réel **OpenRouteService (ORS)** pour calculer distance et durée à partir des coordonnées (lon/lat).
EF-05	Endpoint de réservation décrémentant le nombre de lits disponibles (mise à jour en base).
EF-06	Frontend React permettant la sélection, l'affichage de la recommandation et la réservation.
ENF-01	Tests de charge réalisés avec JMeter (résultats interprétés en tenant compte de la dépendance ORS).
ENF-02	Backend conçu comme un service autonome compatible microservices.
ENF-03	Tests unitaires et d'intégration exécutés via Maven.
ENF-04	Gestion explicite des cas d'erreur côté backend, avec codes HTTP adaptés sur réservation (**404/409/200**).
ENF-05	Intégration continue via GitHub Actions.

Ce tableau constitue un élément central pour démontrer la conformité de la PoC aux attentes du comité d'architecture.

2.4 Définition du périmètre de la preuve de concept

Éléments inclus dans le périmètre

- recommandation d'un hôpital à partir de critères fonctionnels simples ;
- **calcul de distance et durée via ORS (simulation routière exacte)** ;
- réservation de lits avec mise à jour de l'état ;
- persistance des données (hôpitaux, lits, spécialités) en **base PostgreSQL** ;
- tests automatisés du backend ;
- tests de montée en charge ;
- pipeline d'intégration continue.

Éléments exclus du périmètre

- intégration avec des systèmes hospitaliers réels ;
- gestion de données patients réelles ;
- **industrialisation des données** (audit, historisation, gestion multi-sources, gouvernance des référentiels) ;
- déploiement en environnement cloud ou production ;
- implémentation complète d'une architecture orientée événements.

Ces exclusions sont assumées et cohérentes avec l'objectif d'une preuve de concept, dont le rôle est de valider des choix d'architecture sans engager des développements lourds ou prématurés.

2.5 Justification du périmètre retenu

Le périmètre de la PoC a été défini de manière à maximiser la valeur apportée au comité d'architecture, tout en limitant les risques et les coûts de développement.

Cette approche permet :

- de tester rapidement les hypothèses techniques clés ;
- de produire des métriques objectives sur les performances ;
- de fournir une base solide pour décider des orientations futures du projet MedHead.

La PoC joue ainsi pleinement son rôle d'outil d'aide à la décision pour les parties prenantes.

3. Architecture et implémentation de la preuve de concept

3.1 Vue d'ensemble de l'architecture de la PoC

La preuve de concept MedHead repose sur une architecture applicative volontairement simple, mais alignée avec les principes d'une architecture microservices. L'objectif n'est pas de livrer une solution complète, mais de démontrer la faisabilité technique et la pertinence des choix d'architecture retenus.

L'architecture de la PoC est composée des éléments suivants :

- un frontend web permettant la démonstration fonctionnelle ;
- un backend exposant des API REST, implémentant la logique métier ;
- une base de données **PostgreSQL** pour la persistance des entités de la PoC ;
- un service externe **OpenRouteService (ORS)** pour le calcul de distance et de durée de trajet.

Cette séparation claire des responsabilités permet de garantir le découplage entre les couches, de faciliter les tests et de préparer une évolution future vers une architecture distribuée plus complexe.

3.2 Architecture backend

3.2.1 Choix technologiques

Le backend de la PoC est développé en Java à l'aide du framework Spring Boot, conformément aux exigences imposées par le consortium MedHead. Ce choix repose sur plusieurs critères :

- maturité et robustesse de l'écosystème Java ;
- forte adoption dans les environnements critiques et réglementés ;
- facilité de création d'API REST ;
- intégration native avec les outils de test et d'intégration continue.

La PoC s'appuie également sur :

- **PostgreSQL** pour la persistance ;
- **Spring Data JPA / Hibernate** pour l'accès aux données ;
- l'intégration d'un service tiers **OpenRouteService (ORS)** pour la simulation routière exacte (distance/durée).

Spring Boot permet de réduire la complexité de configuration et d'accélérer la mise en œuvre, tout en respectant de bonnes pratiques d'architecture logicielle.

3.2.2 Organisation du code backend

Le code backend est structuré de manière claire et cohérente, en suivant une séparation logique des responsabilités :

- Controllers : exposition des endpoints REST ;
- Services : implémentation de la logique métier ;
- Repositories : accès aux données via JPA ;
- Models / DTO : représentation des données échangées ;
- Configuration et utilitaires : gestion des paramètres techniques.

Cette organisation favorise la lisibilité du code, la maintenabilité et la testabilité des différents composants.

3.2.3 Endpoints exposés

Le backend expose plusieurs endpoints REST permettant de couvrir les fonctionnalités de la PoC :

Endpoint de recommandation (`POST /recommendations`)
Permet de recommander un établissement hospitalier à partir d'une spécialité et d'une zone d'origine. La logique de recommandation repose sur :

- filtrage des établissements par spécialité ;
- prise en compte de la disponibilité des lits ;

- appel ORS pour calculer distance et durée de trajet (route réelle) ;
- sélection de l'établissement minimisant la durée (critère principal).

Endpoint de réservation (`POST /reservations`)
Permet de réserver un lit dans un établissement donné et de mettre à jour la disponibilité en base PostgreSQL.
Les codes HTTP sont utilisés pour rendre l'API explicite :

- **200 OK** : réservation confirmée ;
- **409 Conflict** : aucun lit disponible (réservation refusée) ;
- **404 Not Found** : hôpital inconnu.

Endpoints de consultation (`GET /zones`, `GET /specialities`)
Exposent les zones d'origine disponibles et le référentiel de spécialités utilisé dans la PoC.

Endpoint de supervision (`GET /health`)
Endpoint de type health check permettant de vérifier rapidement l'état de fonctionnement du backend.

3.2.4 Gestion de la distance et intégration ORS

Le calcul de distance et de durée de trajet repose sur **OpenRouteService (ORS)** via l'API "directions". Ce choix répond à l'exigence de simulation routière exacte (distance réellement parcourue) dans un contexte d'urgence.

Cette approche implique :

- une dépendance à un service tiers (latence réseau, quotas, disponibilité) ;
- la nécessité, pour une industrialisation, de prévoir des mécanismes de **timeouts**, de **cache** et de **stratégie de repli** (fallback) en cas d'indisponibilité ;
- une gestion maîtrisée des erreurs (le backend ignore un candidat si ORS échoue pour lui, et renvoie un message explicite si aucun itinéraire ne peut être calculé).

3.3 Architecture frontend

3.3.1 Choix technologiques

Le frontend de la PoC est développé à l'aide du framework React, avec l'outil de build Vite et la bibliothèque Bootstrap pour la mise en forme.

Ce choix est motivé par :

- la conformité aux exigences du projet (framework JavaScript moderne) ;
- la rapidité de mise en œuvre ;
- la simplicité de création d'interfaces réactives ;
- la séparation claire entre logique métier et présentation.

3.3.2 Rôle du frontend dans la PoC

Le frontend a pour rôle principal de démontrer le fonctionnement de la PoC, et non de constituer une interface utilisateur finalisée.

Il permet notamment :

- la sélection d'une spécialité médicale ;
- la sélection d'une zone d'origine ;
- l'affichage de la recommandation proposée par le backend ;
- la réservation d'un lit avec confirmation visuelle et mise à jour des disponibilités.

Le design est volontairement simple afin de se concentrer sur la validation fonctionnelle et technique.

3.3.3 Organisation du code frontend

Le code frontend est structuré de manière modulaire :

- composants réutilisables pour l'interface ;
- factorisation de la logique via un hook personnalisé ;
- séparation des appels API dans des services dédiés (fetch zones, spécialités, recommandations, réservations).

Cette organisation améliore la lisibilité du code et facilite son évolution, tout en respectant les bonnes pratiques de développement frontend.

3.4 Alignement avec une architecture microservices

Bien que la PoC n'implémente qu'un nombre limité de services, son architecture est pensée pour s'inscrire dans une approche microservices :

- backend autonome et découplé ;
- communication via API REST ;
- possibilité d'externaliser certains composants (ex. service de recommandation, service de réservation, service de référentiel) ;
- compatibilité avec une architecture orientée événements à terme.

3.5 Bénéfices de l'architecture retenue

L'architecture mise en œuvre dans la PoC présente plusieurs avantages :

- clarté et simplicité, adaptées à une preuve de concept ;
- séparation des responsabilités ;
- testabilité élevée ;
- évolutivité vers une architecture cible plus complète.

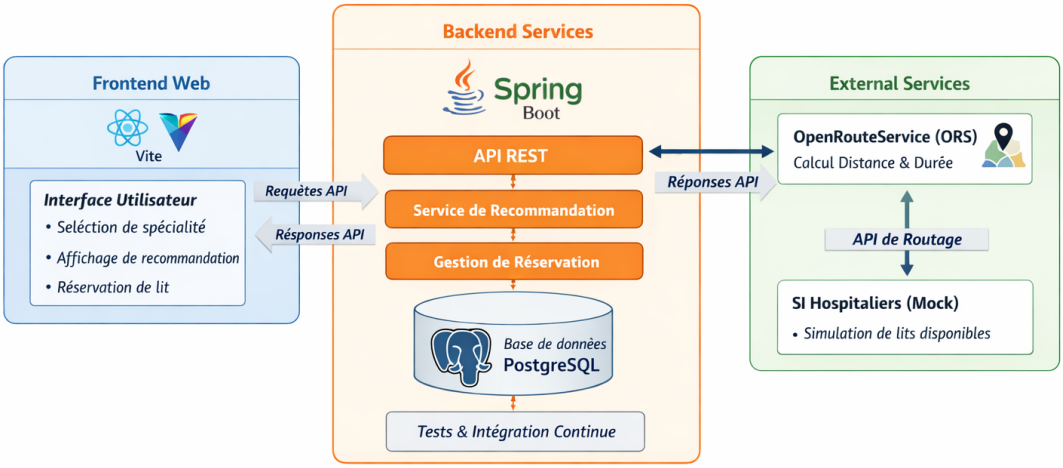


Figure 2 – Architecture applicative de la preuve de concept MedHead

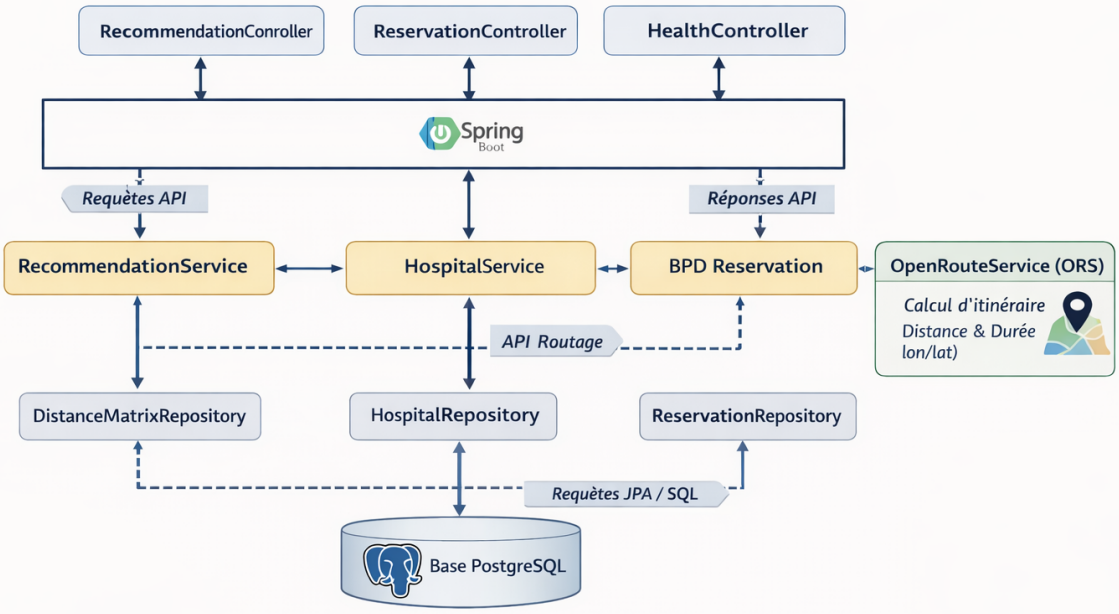


Figure 3 – Diagramme de composants de la preuve de concept MedHead

4. Qualité, tests, performance et intégration continue

4.1 Stratégie globale de qualité

La qualité de la preuve de concept MedHead a été abordée comme un élément central du projet, en cohérence avec la criticité du domaine médical et les attentes exprimées par le comité d'architecture.

La stratégie retenue repose sur plusieurs piliers complémentaires :

- automatisation des tests afin de détecter rapidement les régressions ;
- validation fonctionnelle et technique des principaux cas d'usage ;
- évaluation des performances du backend sous charge ;
- mise en place d'une intégration continue garantissant la traçabilité et la reproductibilité des builds.

4.2 Tests automatisés du backend

4.2.1 Types de tests mis en œuvre

Le backend Spring Boot de la preuve de concept MedHead est couvert par plusieurs catégories de tests automatisés, visant à valider à la fois le comportement technique et fonctionnel du système.

Les tests mis en œuvre comprennent :

• **Tests de démarrage de l'application**
Vérification du bon chargement du contexte Spring Boot, de la configuration JPA et de la connexion à la base de données PostgreSQL.

• **Tests de services métier**
Validation de la logique applicative, notamment :

- le filtrage des établissements par spécialité médicale ;
- la prise en compte de la disponibilité des lits ;
- l'intégration du service de routage OpenRouteService pour l'évaluation des distances et durées.

• **Tests de contrôleurs REST (tests d'intégration)**
Réalisation de scénarios complets à l'aide de MockMvc, couvrant :

- l'endpoint de recommandation d'hôpital ;
- les cas d'erreur métier (zone inconnue, absence de lits disponibles) ;
- l'endpoint de réservation avec mise à jour en base PostgreSQL ;
- la gestion des codes HTTP (200 OK, 404 Not Found, 409 Conflict).

Afin de garantir la reproductibilité et l'indépendance des tests vis-à-vis des dépendances externes, le service de routage ORS peut être isolé via des mécanismes de mocking lors des campagnes de tests automatisés.

L'ensemble des tests est exécuté automatiquement à l'aide de Maven et du framework JUnit, assurant une couverture cohérente des composants critiques de la PoC.

4.2.2 Exécution et résultats des tests backend

Les tests backend sont exécutés via la commande suivante :

```
./mvnw test
```

L'exécution complète de la suite de tests a produit les résultats suivants :

• ensemble des tests exécutés avec succès ;
• 0 échec ;
• 0 erreur ;
• temps d'exécution global compatible avec une utilisation en intégration continue.

Ces résultats confirment la stabilité fonctionnelle du backend MedHead et la robustesse de son architecture applicative, y compris dans un contexte intégrant une base de données relationnelle et des services externes.

Ces tests automatisés constituent un socle essentiel pour sécuriser les évolutions futures du système et accompagner une montée en charge ou une industrialisation progressive de la plateforme MedHead.

4.3 Tests de performance et montée en charge

4.3.1 Objectifs des tests de charge

Les tests de performance ont pour objectif principal de répondre aux inquiétudes des parties prenantes concernant la capacité du système à fonctionner correctement lors de pics d'activité, typiques des situations d'urgence médicale.

Ils visent notamment à :

- mesurer les temps de réponse du backend ;
- vérifier l'absence d'erreurs sous charge ;
- évaluer la capacité de traitement du service de recommandation ;
- observer l'impact de l'intégration de services externes (routage ORS) et de la persistance en base de données PostgreSQL.

4.3.2 Outil et scénario de test

Les tests de charge ont été réalisés à l'aide de l'outil Apache JMeter.

Le scénario de test met en œuvre :

- un groupe d'utilisateurs simulés ;
- des requêtes POST répétées vers l'endpoint de recommandation ;
- un volume total de 1000 requêtes afin de représenter une montée en charge significative.

Contrairement à la première version de la PoC utilisant un service de routage simulé, cette campagne de tests a été réalisée avec :

- l'intégration réelle de l'API OpenRouteService (ORS) pour le calcul des distances et durées ;
- l'utilisation d'une base de données PostgreSQL pour la persistance des établissements hospitaliers.

Cette approche permet d'obtenir des métriques plus réalistes, tout en introduisant une variabilité liée à la latence d'un service externe.

Dans une perspective d'industrialisation, plusieurs stratégies pourraient être envisagées pour stabiliser les performances :

- mocker ORS pour les tests purement backend ;
- mettre en place un mécanisme de cache des résultats de routage ;
- intégrer des mécanismes de résilience (timeouts, circuit breakers).

4.3.3 Résultats des tests de charge

Les résultats obtenus lors des tests de charge sont les suivants :

- nombre total de requêtes : 1000 ;
- taux d'erreur : 0 % ;
- temps de réponse moyen : environ 39 ms ;
- temps de réponse minimum : 27 ms ;
- temps de réponse maximum observé : 380 ms ;
- débit moyen : environ 33 requêtes par seconde.

Ces résultats montrent que le backend MedHead conserve de bonnes performances même avec l'intégration d'un service externe de routage et une base de données relationnelle.

La variabilité observée sur le temps de réponse maximal s'explique principalement par la dépendance à l'API ORS, ce qui est attendu dans une architecture s'appuyant sur des services tiers.

Dans l'ensemble, la PoC démontre la capacité du système à supporter une montée en charge réaliste tout en conservant une stabilité et une fiabilité élevées.

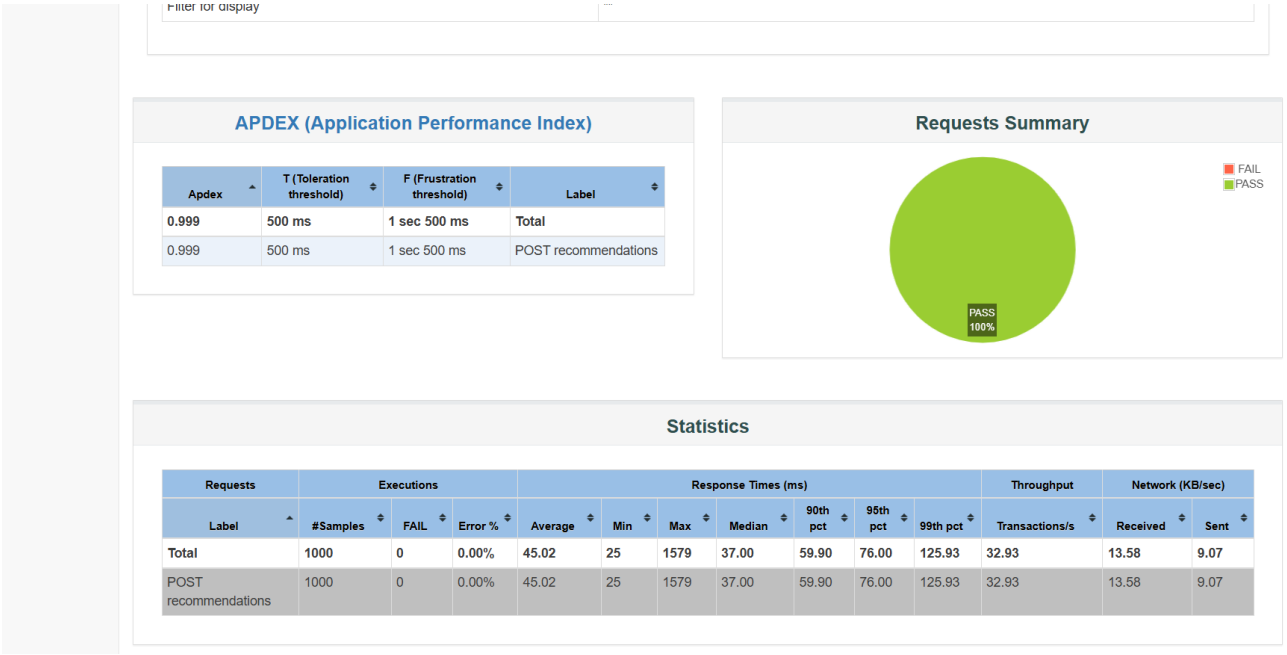


Figure 4 – Résultats des tests de charge réalisés avec Apache JMeter sur l'endpoint de recommandation, intégrant OpenRouteService réel pour le calcul de distance et persistance PostgreSQL pour les données hospitalières.

4.4 Intégration continue (CI)

4.4.1 Mise en place du pipeline CI

Une chaîne d'intégration continue a été mise en place à l'aide de GitHub Actions. Elle est déclenchée automatiquement à chaque push sur le dépôt de code.

Le pipeline comprend notamment :

- récupération du code source ;
- installation des dépendances ;
- compilation du projet ;
- exécution des tests automatisés.

4.4.2 Bénéfices de l'intégration continue

La mise en place de l'intégration continue apporte plusieurs bénéfices majeurs :

- détection rapide des erreurs et régressions ;
- amélioration de la qualité globale du code ;
- traçabilité des changements ;
- reproductibilité des builds.

Bien que la PoC ne soit pas destinée à être déployée en production, cette démarche prépare le projet à une industrialisation future.

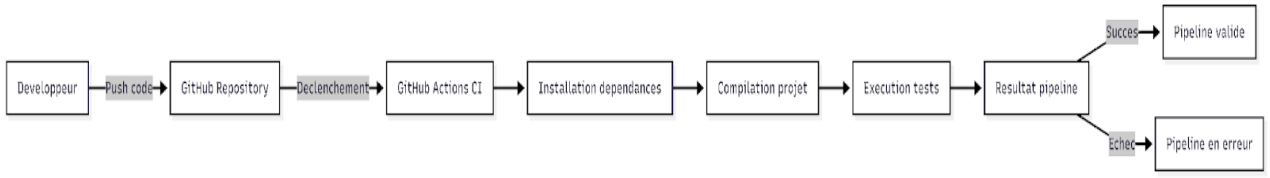
4.5 Limites et perspectives

Les tests réalisés valident les hypothèses techniques majeures, mais présentent certaines limites inhérentes au périmètre :

- tests de charge réalisés sur un environnement local ;
- dépendance ORS non testée en résilience (pannes, timeouts, quotas) ;
- absence de supervision avancée.

Ces limites constituent des axes d'amélioration clairement identifiés pour les phases ultérieures du projet.

Figure 5 – Pipeline d'intégration continue de la PoC MedHead



5. Enseignements, limites et recommandations

5.1 Enseignements tirés de la preuve de concept

La réalisation de la preuve de concept MedHead a permis de valider plusieurs hypothèses majeures formulées en amont du projet.

Sur le plan fonctionnel, la PoC démontre qu'il est possible de fournir une recommandation d'établissement hospitalier pertinente à partir de critères tels que la spécialité médicale, la disponibilité des lits et une distance/ durée calculées de manière réaliste via ORS. Le parcours de bout en bout, depuis la demande émise par les services d'urgence jusqu'à la réservation d'un lit, a pu être implémenté et démontré avec succès.

Sur le plan technique, l'architecture retenue confirme la pertinence d'une séparation claire entre le frontend et le backend. L'utilisation d'une API REST centralisée permet une bonne lisibilité des échanges et facilite les évolutions futures. La structuration du code côté backend (contrôleurs, services, repositories, modèles) et côté frontend (composants, hooks, services) contribue à la maintenabilité et à la clarté de la solution.

Enfin, la mise en place de tests automatisés, de tests de charge et d'une intégration continue apporte une première garantie de qualité et de robustesse.

5.2 Limites identifiées de la PoC

Malgré les résultats positifs obtenus, plusieurs limites ont été identifiées, principalement liées au périmètre volontairement restreint.

Même si les données sont persistées en base PostgreSQL dans la PoC, elles restent **des données de démonstration** : la disponibilité des lits n'est pas alimentée par des systèmes hospitaliers réels et ne reflète pas un "temps réel" industriel.

La dépendance à ORS apporte une simulation routière exacte, mais introduit aussi :

- une latence variable (réseau) ;
- des contraintes de quotas ;
- la nécessité de mécanismes de résilience pour une mise en production.

Par ailleurs, la PoC ne couvre pas les aspects liés à la sécurité, à l'authentification, à la gestion des droits ou à la conformité réglementaire (notamment RGPD), indispensables pour une mise en production.

5.3 Recommandations pour une mise à l'échelle industrielle

Dans une perspective de passage à l'échelle, plusieurs recommandations peuvent être formulées :

- intégrer des sources de données réelles (SI hospitaliers) pour alimenter lits/spécialités ;
- prévoir cache + timeouts + circuit breaker pour la dépendance ORS ;
- faire évoluer l'architecture vers une approche plus distribuée (scalabilité horizontale, résilience, supervision) ;
- enrichir la CI avec contrôles qualité, tests perf automatisés et déploiements vers environnements de test ;
- mettre en place une gouvernance claire pilotée par le comité d'architecture MedHead.

Tableau 1 – Évolution de la preuve de concept vers une solution de production

Axe	Preuve de concept (PoC)	Solution de production
Objectif	Valider la faisabilité technique et architecturale	Fournir un service fiable, sécurisé et scalable
Données	Données de démonstration persistées en PostgreSQL	Données réelles issues des SI hospitaliers avec gouvernance
Recommandation	Algorithme simple (spécialité + lits + durée ORS)	Algorithme enrichi (priorités métier, règles avancées, SLA)
Performance	Tests JMeter locaux avec variabilité liée à ORS	Tests à grande échelle, résilience et gestion des quotas
Scalabilité	Exécution sur une instance unique	Architecture distribuée avec montée en charge horizontale
Résilience	Limitée (fallback simple en cas d'échec ORS)	Timeouts, circuit breakers, cache, reprise automatique
Sécurité	Non implémentée	Authentification, autorisation,

Axe	Preuve de concept (PoC)	Solution de production
		chiffrement des échanges
Traçabilité	Logs basiques	Observabilité complète et audit
CI/CD	Intégration continue (build + tests)	CI/CD complet avec déploiements automatisés
Supervision	Absente	Monitoring, alerting et tableaux de bord
Conformité	Hors périmètre	RGPD et exigences réglementaires du secteur santé

5.4 Conclusion générale

La preuve de concept MedHead remplit pleinement ses objectifs initiaux en démontrant la faisabilité technique et architecturale d’un système de recommandation hospitalière.

Les choix réalisés (Spring Boot, PostgreSQL, React, intégration ORS, gestion d’erreurs REST, tests et CI) constituent une base solide pour une éventuelle industrialisation. Les enseignements tirés et les limites identifiées offrent une vision claire des travaux à mener pour transformer cette PoC en une solution opérationnelle à grande échelle.

Dans ce contexte, la PoC peut être considérée comme un succès et un point de départ crédible pour les phases futures du projet MedHead.