Document de reporting

Projet : Proof of concept - **MedHead**

Auteur : Callait Emrys

Date : Janvier 2026

Version : 1.0

Table des matières

Document de reporting 1
Projet : Proof of concept - Meadhead 1
Auteur : Callait Emrys 1
Date : Janvier 2026 1
Version : 1.0 1
Contexte du projet 2
Objet du document 2
Aperçu du projet 3
Connexion et gestion d'erreurs 3
Dashboard : Sélection de la spécialité 3
Recherche de proximité (GPS ou Adresse) 3
Réservation d'un lit 3
Développement 4
Gestion des versions : GIT, Github Desktop & Github 4
IDE VS Code 4
Qualité du code et tests 6
Exécution du PoC 8
Pipeline CI/CD 8
Jenkins 8
Stage Compilation & tests : Jacoco (Test unitaire) 8
Stage Analyse : SonarQube 9

Stage Test API : Newman (Test d’intégration) 9
Stage Puppeteer (Tests E2E) 10
Documentation 11
Code as a documentation 11
Test as a documentation 12
Modèle de donnés 13
Optimisation du modèle de données 13
Justification du choix Front-end 14
Conformité RGPD & normes 14
Architecture & évolutivité 15
Réflexion "Chef de Projet" 16

Contexte du projet

MedHead est un groupement d'importantes institutions médicales opérant au sein du système de santé britannique et soumis aux directives du NHS. Actuellement, les organisations membres utilisent une grande variété de technologies et d'appareils différents.

Le consortium souhaite mettre en place une nouvelle plateforme unique pour unifier leurs pratiques. L'ambition finale est d'utiliser cet outil pour améliorer la qualité des soins prodigués aux patients

Objet du document

Ce document sert de synthèse officielle pour le développement du Proof of Concept (PoC) — ou prototype — du projet MedHead. Ses principaux rôles sont les suivants :

- Justification technique : Expliquer et motiver les choix technologiques effectués durant le développement.
- Synthèse des résultats : Présenter les résultats obtenus lors de la création de ce prototype.

Aperçu du projet

Lien de la vidéo : https://www.youtube.com/watch?v=7hYXP1wDDu4

Connexion et gestion d'erreurs

La vidéo s'ouvre sur l'interface connexion MedHead.

Identification : L'utilisateur saisit son adresse e-mail (utilisateur1@compte.co) et son mot de passe.

Gestion d'erreur : Une première tentative échoue, affichant un message d'erreur rouge "Authentification requise" illustrant la robustesse du système de sécurité. La connexion réussit ensuite lors de la saisie correcte des identifiants

Dashboard : Sélection de la spécialité

Une fois connecté, l'utilisateur accède au tableau de bord intitulé « Quelle est l'urgence ?».

L'interface propose une liste de groupes de spécialités (Medical Specialties, Mental Health, etc.)

L'utilisateur peut soit naviguer dans les listes déroulantes, soit utiliser la barre de recherche pour trouver une spécialité précise, comme la cardiologie ou la chirurgie.

Recherche de proximité (GPS ou Adresse)

Après avoir choisi une spécialité, le système redirige vers l'interface "Trouver un hôpital"

Par adresse : L'utilisateur saisit une adresse (ex: "Paris") dans le champ de texte et clique sur "Chercher.

Par GPS : Le bouton "GPS" est également disponible pour géolocaliser automatiquement l'utilisateur et proposer les unités de soins les plus proches.

Le système affiche alors une liste d'hôpitaux (ex: Hôpital Bichat, Lariboisière) avec le temps de trajet estimé et le nombre de lits disponibles.

Réservation d'un lit

La dernière étape montre la concrétisation du besoin médical.

L'utilisateur clique sur le bouton "Réserver ce lit" associé à l'hôpital choisi.

Une fenêtre contextuelle de confirmation apparaît instantanément avec le message : "Lit réservé avec succès !".

On peut observer que le compteur de lits disponibles diminue suite à cette action, validant la mise à jour en temps réel de la base de données

Développement

Gestion des versions : GIT, Github Desktop & Github

Git C'est le système de gestion de versions qui agit comme le moteur du projet en permettant de suivre l'historique de chaque modification du code.

GitHub est la plateforme d'hébergement web qui sert de "coffre-fort" centralisé pour stocker les différents répertoires (repositories) du projet en ligne. Pour MedHead, deux repositories distincts ont été créés : un pour le code technique (API et application web) et un autre pour les documents d'architecture et de reporting. Son utilisation est cruciale car elle permet non seulement de cloner le projet sur n'importe quel poste de travail, mais elle est également la source des versions dans le pipeline CI/CD.

GitHub Desktop est une application de bureau qui offre une interface graphique simplifiée qui permet de piloter Git sans avoir à taper des lignes de commande complexes dans un terminal. Elle a été choisie pour le projet MedHead afin de faciliter la gestion quotidienne des tâches

IDE VS Code

Visual Studio Code a été sélectionné comme IDE principal pour le développement du PoC. Sa polyvalence, renforcée par un catalogue d'extensions spécifique, permet de transformer cet éditeur léger en une station de travail robuste capable de gérer simultanément le Back-end (API) et le Front-end (Webapp) sans changer

Le socle de développement Java

Pour manipuler le langage Java utilisé dans tout le projet, plusieurs extensions essentielles ont été installées :

- **Language Support for Java (Red Hat) & Java** : Fournissent l'intelligence de code (autocomplétion, navigation) indispensable à la productivité.
- **Debugger & Test Runner for Java** : Permettent de détecter les anomalies et d'exécuter les tests unitaires directement depuis l'IDE, garantissant ainsi la fiabilité des fonctions d'insertion et de consultation.
- **Maven for Java & Gradle** : Assurent la gestion des dépendances et la compilation du projet, Maven étant l'outil de build principal retenu pour ce PoC.

L'écosystème technique

Le projet reposant sur le framework **Spring Boot** pour simplifier la configuration et accélérer le développement, une suite d'outils dédiée a été intégrée :

- **Spring Boot Extension Pack & Tools** : Offrent une assistance au codage spécifique aux annotations Spring.
- **Spring Initializr Java Support** : A été utilisé pour générer la structure de base de l'API et de l'application web avec les caractéristiques requises.

- **Lombok Annotations Support** : Indispensable pour exploiter la dépendance Lombok installée, permettant de réduire drastiquement le code (getters, setters) dans les modèles comme Hopital.java ou Utilisateur.java.

- **Indent-rainbow & XML Tools** : Améliorent la lisibilité immédiate du code et facilitent la gestion des fichiers de configuration complexes comme le pom.xml.

- **Vetur** : Installée pour supporter les composants Front-end et assurer une coloration syntaxique optimale des interfaces.

Qualité du code et tests

Afin de répondre aux standards de qualité élevés du milieu médical (NHS)

SonarQube IDE + CICD

Analyse le code en continu pour détecter les bugs potentiels et les vulnérabilités de sécurité avant même l'archivage sur GitHub.

JaCoCo (Java Code Coverage) IDE + CICD

Cet outil est indispensable pour valider la fiabilité des tests unitaires mentionnés dans ce rapport. Il mesure le pourcentage de code réellement exécuté lors des tests. Pour MedHead, cela permet de s'assurer que le PoC à une couverture de tests.

Postman - Newman : Tests fonctionnels et documentation

L'outil **Postman** a été utilisé pour tester et valider les points de terminaison (endpoints) de l'API tout au long du développement.

- **Documentation technique** (Test as a documentation): La collection Postman créée sert à tester les endpoint à la mains, mais aussi de documentation technique de référence, détaillant l'ensemble des fonctionnalités et des appels possibles vers l'API.
    - Dossier avec la collection : https://github.com/EmrysC/MedHead_PoC_systeme_d_intervention_d_urgence/tree/main/Poc/Poc/newman_tests

- **Intégration CI/CD** : Cette collection ne se limite pas à des tests manuels ; elle a permis l'élaboration de scripts de tests automatisés. Ces derniers sont intégrés dans le pipeline de déploiement continu (**CI/CD**) via GitHub Actions, permettant ainsi de vérifier automatiquement le bon fonctionnement de l'API à chaque nouvelle livraison.

Jmeter Métriques de performance & peuplement

Apache JMeter est une solution de référence *open-source* retenue pour réaliser les **tests de performance** du système. Il permet de simuler des groupes d'utilisateurs virtuels qui envoient des requêtes simultanées vers les API du projet (Spring Boot / MariaDB).

Résultat de l’analyse de Jmeter :

https://github.com/EmrysC/MedHead_PoC_systeme_d_intervention_d_urgence/blob/main/Poc/Poc/src/test/jmeter/Description%20des%20tests.pdf

L'utilisation de JMeter vise principalement à :

- **Identifier les goulots d’étranglement** pour garantir la disponibilité du système d'urgence.

- **Mesurer des indicateurs clés** : la latence (temps de réponse), le débit (nombre de réservations par seconde) et la robustesse face aux accès concurrents en base de données.

- **Tester le "chemin critique"** : le scénario nominal simule un utilisateur qui se connecte, recherche une spécialité et une unité de soins, puis effectue une réservation de lit.

- **Insérer de la donnée en masse** : Même si Jmeter n’est pas normalement fait pour faire ça, il a été détourné pour créer des utilisateurs en masse et ainsi générer le script de seeding.

Exécution du PoC

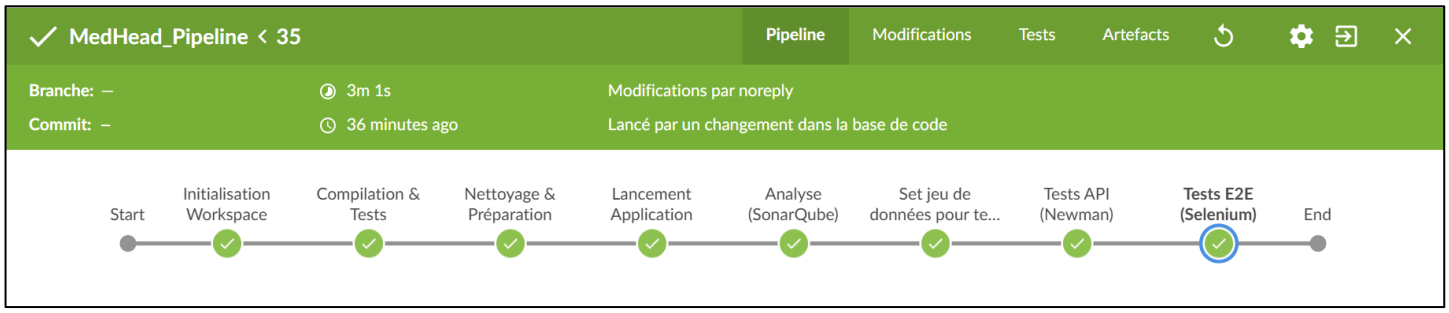
- **Docker** est une plateforme qui permet de "transformer" une application en un bloc standardisé et autonome, que l'on appelle un conteneur.

- **MariaDB** est le système de gestion de base de données relationnelle (SGBDR) *open source* choisie pour assurer la persistance des données du système MedHead. Il s'agit du "coffre-fort" où sont stockées de manière structurée toutes les informations critiques du PoC, notamment les utilisateurs, les hôpitaux et les référentiels de spécialités du NHS.

Pipeline CI/CD

Jenkins

Jenkins est un serveur d'automatisation *open-source* utilisé pour mettre en place ce qu'on appelle l'**Intégration Continue (CI)** et le **Déploiement Continu (CD)**.



Stage Compilation & tests : Jacoco (Test unitaire)

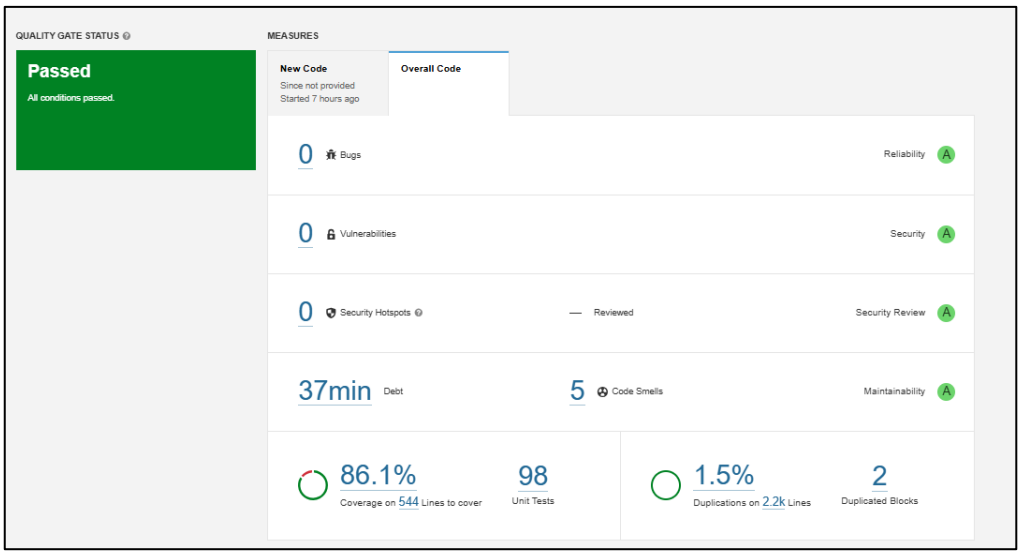
Cette étape du pipeline CI/CD constitue le premier rempart de qualité du projet MedHead, intervenant immédiatement après le push du code sur GitHub. Elle combine la validation technique (compilation) et la validation logique (tests unitaires).

- Le code source est exempt d'erreurs de syntaxe.

- Toutes les dépendances nécessaires (Spring Boot, MariaDB Driver, Lombok) sont correctement résolues.

- Les tests unitaires, situés dans les répertoires de test, sont exécutés automatiquement. Ces tests vérifient les fonctions des programmes.

Stage Analyse : SonarQube



L'étape d'analyse SonarQube dans ton pipeline GitHub Actions joue le rôle de **"Quality Gate"** (porte de qualité) :

- **Gardien (Quality Gate Status: Passed) :** Comme on le voit en haut à gauche, le statut est "Passed". Dans le pipeline, si ce statut était "Failed" (par exemple, si la couverture descendait sous 80%, faille de sécurité, …), le déploiement s'arrêterait immédiatement. Cela empêche de livrer du code défectueux.

- **Automatisation de la revue de code :** Au lieu d'attendre qu'un humain lise ton code, la CI/CD fait ce travail en quelques secondes à chaque *push*. C'est un gain de temps énorme pour l'équipe de développement.

- **Confiance pour la mise en conteneur :** Puisque SonarQube valide la qualité, l'image **Docker** générée juste après est basée sur un code sain et sécurisé.

Stage Test API : Newman (Test d'intégration)

L'automatisation via Newman permet de valider le comportement de l'API à chaque modification du code (Push ou Pull Request) sur la branche main:

- **Vérification des contrats** : On s'assure que les endpoints (comme la recherche d'hôpitaux ou la réservation de lit) répondent toujours selon le format attendu.

- **Non-régression** : Newman garantit qu'une nouvelle fonctionnalité n'a pas cassé un service existant.

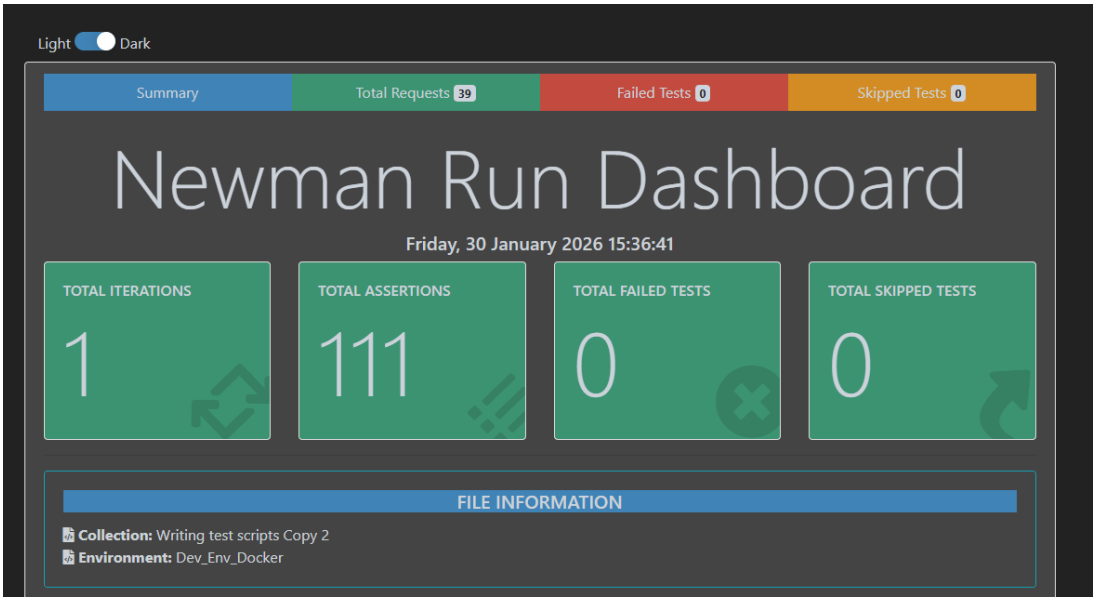
- **Validation du fonctionnement global** : Contrairement aux tests unitaires, Newman teste la communication réelle entre l'application et la base de données MariaDB.

De plus un **Newman Run Dashboard** (illustré ci-dessous) est associé, il transforme les logs techniques en une interface de reporting claire :

- **Visibilité immédiate** : Le dashboard montre instantanément le statut global (ici, **0 échec** sur **111 assertions**).

- **Preuve de conformité** : Pour un projet soumis aux normes du **NHS**, il fournit une preuve visuelle datée (30 janvier 2026) que le système est stable et opérationnel.

- **Analyse détaillée** : Il permet de voir précisément le nombre d'itérations et de requêtes (39 dans ce run), facilitant l'identification rapide d'un test spécifique qui aurait échoué.



Stage Puppeteer (Tests E2E)

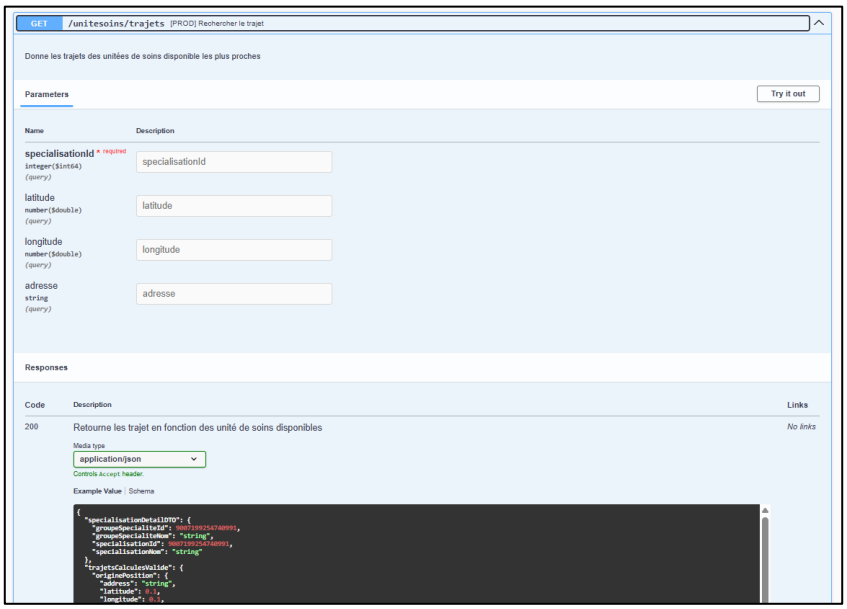
L'étape finale de validation consiste en des tests **End-to-End (E2E)** automatisés avec **Puppeteer**. Alors que les tests unitaires valident le code et Newman valide l'API, Puppeteer simule le comportement réel d'un utilisateur dans son navigateur pour tester le "chemin critique" du système.

- **Validation du parcours nominal** : Il vérifie que l'enchaînement Connexion -> Recherche -> Réservation fonctionne sans accroc dans l'interface graphique.
- **Optimisation CI/CD (Docker/Jenkins)** : Le script est configuré pour s'exécuter en mode "headless" (sans interface visible) avec des arguments spécifiques (--no-sandbox, --disable-dev-shm-usage) pour garantir sa stabilité dans un conteneur Docker.
- **Preuve par l'image** : En cas de succès ou d'échec, le système capture automatiquement des **screenshots**, permettant une preuve visuelle de l'état de l'application.

Dossier des test E2E : https://github.com/EmrysC/MedHead_PoC_systeme_d_intervention_d_urgence/tree/main/Poc/Poc/e2e_tests

Documentation

Code as a documentation



Dans le cadre de l'architecture Spring Boot de MedHead, le concept de "Code as Documentation" (ou documentation par le code) est implémenté directement au sein des Controllers. Grâce à l'intégration de l'outils Swagger, la documentation technique n'est plus un document statique séparé, mais elle est générée automatiquement à partir des annotations Java présentes dans le code source. Les avantages de cette approche sont :

- **Synchronisation parfaite** : Comme la documentation est extraite des contrôleurs, elle reflète toujours la réalité technique de l'API (paramètres requis, types de données, codes de retour).

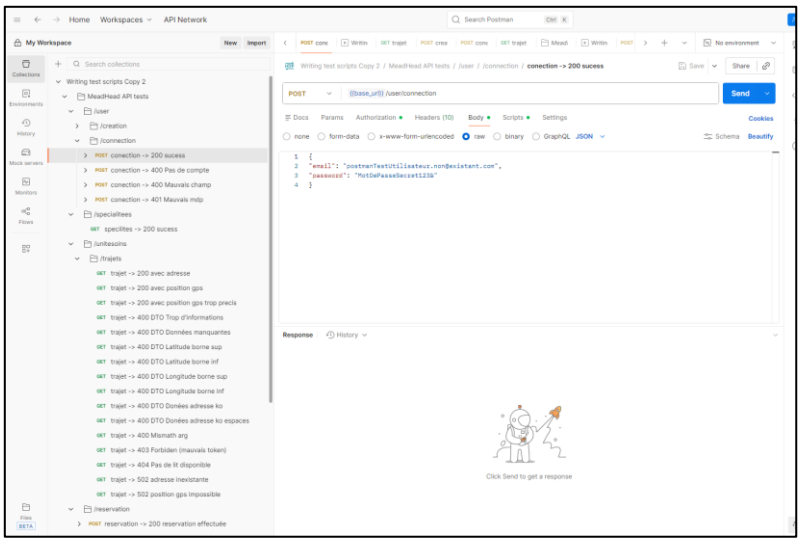
- **Interface interactive** : L'image de l'interface Swagger montre l'endpoint /unitesoins/trajets avec ses paramètres détaillés (specialisationId, latitude, longitude, adresse) et un bouton "Try it out" permettant aux développeurs de tester l'appel en temps réel.

- **Contrat d'interface clair** : Le schéma de réponse (JSON) est exposé de manière transparente, définissant précisément la structure des données que l'application web doit consommer.

- **Documentation vivante** : Cette méthode assure que la documentation évolue en même temps que le code, évitant ainsi l'obsolescence des informations techniques.

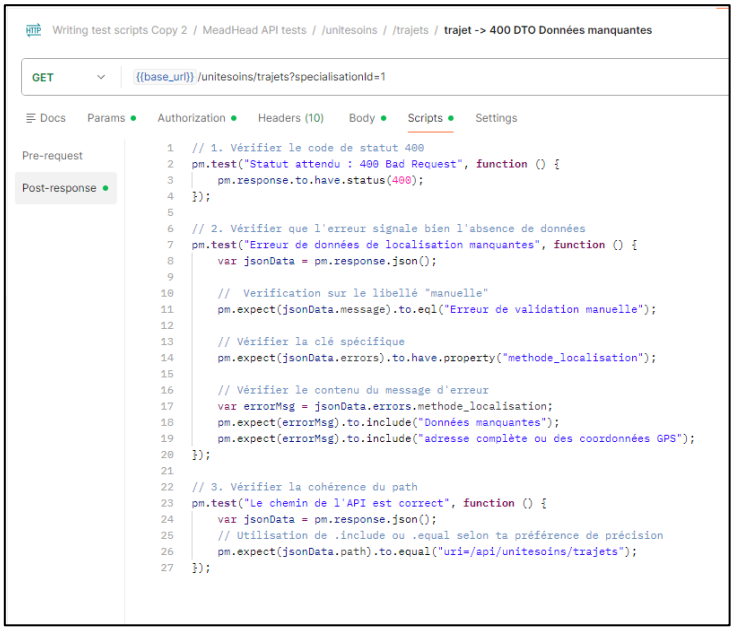
Test as a documentation

L'approche **"Test as a Documentation"** via Postman et Newman complète parfaitement Swagger en ajoutant une couche de validation comportementale. Si Swagger décrit ce que l'API *est* (sa structure), Postman décrit comment elle *doit se comporter* dans des situations réelles.

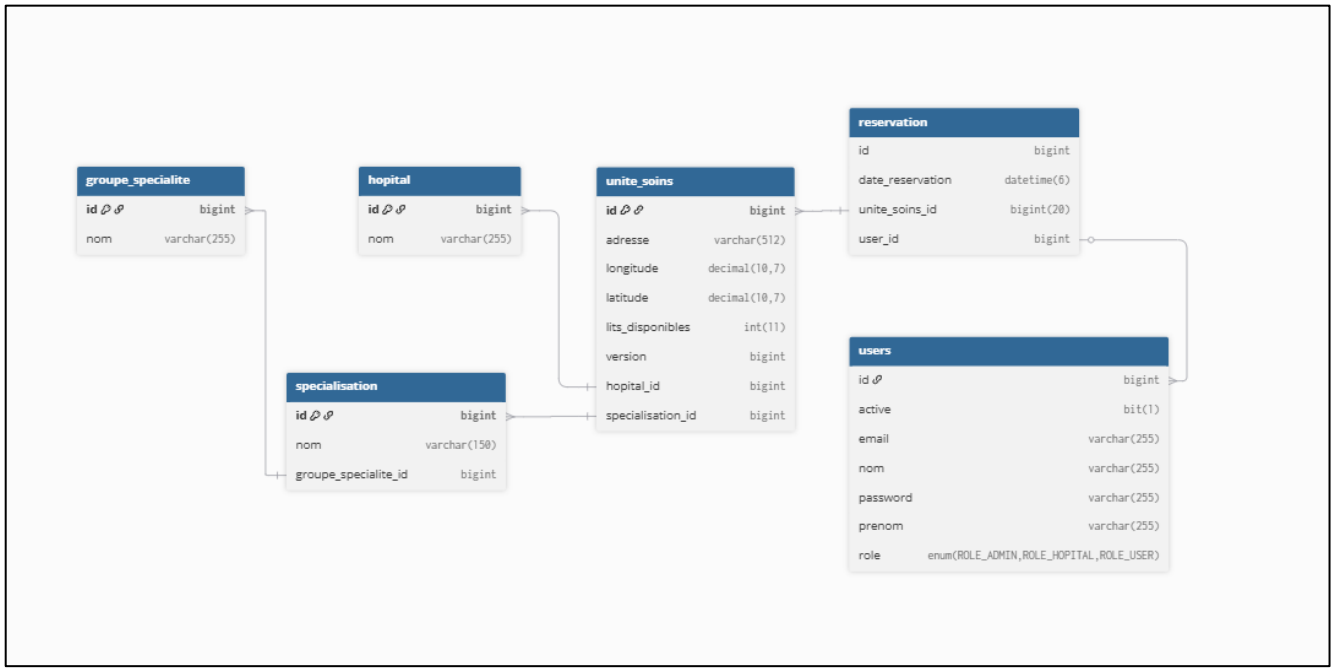


L'exécution de ces tests via Newman dans ton pipeline transforme la documentation en une source de vérité absolue.

- **Validation du contrat en temps réel** : Le dashboard Newman montre que sur 39 requêtes et 111 assertions, aucune n'a échoué. Cela prouve que la documentation (les tests) correspond exactement à l'état réel du code déployé dans l'environnement Docker.



Modèle de donnés



Optimisation du modèle de données

Le schéma de base de données du PoC a été affiné pour répondre aux exigences réelles de la médecine d'urgence. Contrairement aux spécifications initiales, la décision technique à été prise de déplacer les attributs adresse et nombre de lits de la table hopital vers la table unite_soins.

Cette architecture, visible dans le diagramme de classes, se justifie par deux facteurs critiques :

- **Précision géographique (Accès) :** Un hôpital est une infrastructure vaste qui peut disposer de plusieurs entrées ou pavillons éloignés. En rattachant l'adresse (et les coordonnées GPS latitude/longitude) directement à l'unité de soins, le système permet de guider les services d'urgence vers le point d'entrée le plus proche du service concerné, optimisant ainsi chaque seconde du trajet.

- **Spécialisation des ressources :** La gestion des capacités au niveau global d'un établissement manque de sens médical. Un lit disponible en maternité ne peut absolument pas être réaffecté à un patient de Mental Health. Isoler les lits_disponibles au sein de l'unité de soins garantit que le système ne propose que des ressources réellement adaptées à la pathologie du patient, évitant des erreurs de réservation qui pourraient compromettre la prise en charge médicale.

Cette structure garantit que chaque reservation est liée à une unité de soins précise, possédant sa propre capacité et sa propre localisation, rendant le prototype MedHead beaucoup plus robuste et proche des besoins du terrain.

Justification du choix Front-end

Le choix de Vue.js, couplé à Bootstrap, a été privilégié pour ce prototype en raison de sa grande simplicité de prise en main et de son efficacité.

L'objectif de l'interface MedHead étant de rester épurée et fonctionnelle (mobiles), Vue.js s'est imposé comme la solution idéale : c'est un framework léger qui ne nécessite pas une complexité technique excessive pour répondre aux besoins du projet.

Cette simplicité permet de se concentrer sur l'essentiel, la recherche d'hôpitaux et la réservation de lits sans alourdir le développement avec des fonctionnalités superflues.

L'utilisation de Bootstrap vient compléter cette approche en offrant une bibliothèque de composants prêts à l'emploi, garantissant une interface propre, professionnelle et "responsive". Pour un projet médical, la réactivité de Vue.js assure une navigation fluide, tandis que sa capacité à communiquer avec l'API REST Spring Boot permet une séparation claire entre la logique métier et l'affichage.

En résumé, ce duo technologique a été choisi pour maximiser la rapidité de développement tout en garantissant une expérience utilisateur stable et performante, parfaitement adaptée aux exigences d'un système d'urgence.

Conformité RGPD & normes

Le respect du RGPD et des normes de sécurité est assuré par une approche de Privacy by Design intégrée dès la genèse du prototype.

- La sécurité des accès est renforcée par l'utilisation de Spring Security au niveau de la Gateway, garantissant une authentification robuste pour chaque flux de données.

- Conformément au principe de minimisation, les données personnelles sont limitées au strict nécessaire : seuls l'e-mail et le mot de passe de l'utilisateur sont stockés, ce dernier étant systématiquement chiffré (BCrypt via Spring Security) pour en garantir la confidentialité

- Traçabilité sécurisée : Le système enregistre l'historique des réservations (qui a réservé quel lit/ressource), permettant un suivi précis tout en restant conforme aux normes de protection.

- Une séparation stricte est opérée en base de données entre les informations des utilisateurs et les données opérationnelles (hôpitaux et lits), évitant tout croisement non autorisé.

- L'outil SonarQube analyse en continu le code pour détecter les vulnérabilités et les risques de sécurité, assurant ainsi une protection proactive conforme aux exigences critiques du système de santé

Architecture & évolutivité

L'adoption d'une architecture en microservices est justifiée par le besoin de scalabilité et de résilience du système de santé britannique. En isolant les fonctionnalités (gestion des utilisateurs, recherche d'hôpitaux, réservations), chaque service peut être déployé, mis à jour et mis à l'échelle indépendamment via le pipeline CI/CD et Docker.

Cette approche évite qu'une panne sur le module de recherche n'affecte la capacité de réservation de lits déjà identifiés, assurant une haute disponibilité critique en situation d'urgence.

De plus, cela permet d'intégrer facilement le système dans un écosystème plus large via une architecture pilotée par les événements, facilitant la communication avec d'autres institutions du consortium.

Enfin, cette modularité technique reflète la réalité organisationnelle du projet, où différentes équipes pourraient travailler sur différents services sans conflits majeurs.

Réflexion "Chef de Projet"

Stratégie de pilotage et management d'équipe

La conduite du projet MedHead repose sur une méthodologie Agile (Scrum) afin d'assurer une livraison incrémentale et flexible du prototype. Le pilotage se concentre sur la levée des obstacles techniques et fonctionnels pour maintenir l'effort sur le chemin critique : le flux Connexion, Recherche de spécialité et Réservation de lit.

- Le soutien opérationnel est assuré par des rituels tels que le Daily Stand-up pour une identification rapide des blocages.
- Des revues de sprint permettent de valider les avancées techniques et fonctionnelles avec les parties prenantes du consortium.

Approche de l'intégration et du déploiement continu (CI/CD)

L'approche technique choisie n'est pas seulement un choix d'outils, mais une stratégie de réduction des risques pour un projet sensible soumis aux directives du NHS.

Le pipeline CI/CD via Jenkins et GitHub agit comme un "gardien de la qualité" automatisé, empêchant toute livraison de code défectueux en production.

Cet investissement en temps de configuration initial est justifié par le gain de stabilité et de confiance qu'il apporte lors de chaque mise en conteneur Docker. Cette automatisation permet à l'équipe de se concentrer sur l'innovation fonctionnelle plutôt que sur la maintenance corrective de dernière minute.

Gestion des risques et pyramide de tests

Pour assurer la conformité d'un système médical d'urgence, la stratégie de test repose sur le concept de la pyramide de tests, allant des tests unitaires JaCoCo aux tests E2E avec Puppeteer. Cette approche permet d'apporter de la valeur progressivement : les tests unitaires valident la logique, Newman valide les contrats d'API et les tests de performance JMeter garantissent que le système supportera une montée en charge réelle.

Soutien à l'équipe et évolution de la solution

LE soutien à l'équipe passerait par la mise à disposition d'un écosystème technique robuste (IDE VS Code optimisé, documentation Swagger automatisée) pour maximiser la productivité. Notament grace aux concepts de « Code as a Documentation » et do «Tests as a documentation » .