# 

# Reporting POC



Table des matières

[Reporting POC 1](#_Toc141047365)

[1. Contexte 3](#_Toc141047366)

[2. Périmètre de la POC 3](#_Toc141047367)

[3. Organisation du projet et des tests 4](#_Toc141047368)

[3.1. Organisation générale 4](#_Toc141047369)

[3.2. Le Backend 4](#_Toc141047370)

[3.2.1. Choix technologiques 4](#_Toc141047371)

[3.2.2. Méthode de développement 5](#_Toc141047372)

[3.2.3. Intégration Continue et Déploiement continu 6](#_Toc141047373)

[3.2.4. Test de la solution 7](#_Toc141047374)

[3.2.4.1. Test Unitaire 7](#_Toc141047375)

[3.2.4.2. Test de Stress 7](#_Toc141047376)

[3.2.5. Normes 8](#_Toc141047377)

[3.3. Le Frontend 8](#_Toc141047378)

[3.3.1. Choix technologiques 8](#_Toc141047379)

[3.3.2. Méthode de développement 8](#_Toc141047380)

[3.3.3. Intégration Continue et Déploiement continu 8](#_Toc141047381)

[3.3.4. Test de la solution 8](#_Toc141047382)

[3.3.5. Normes 8](#_Toc141047383)

# Contexte

MedHead est un regroupement de grandes institutions médicales œuvrant au sein du système de santé britannique et assujetti 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.

L’objectif de cette POC (Preuve de concept : abréviation de sa traduction anglaise « Proof Of Concept ») est de fournir l’hôpital le plus proche avec des lits disponibles par domaine de spécialisation (NHS : Service de santé National). Cette POC sera un argument pour convaincre le comité de validation de l’architecture cible retenue.

# Périmètre de la POC

Ce POC sera limité aux fonctionnalités présentes dans le tableau ci-joint.

|  |  |
| --- | --- |
| Fonctionnalité | Description |
| Lister les spécialisations | Retourne la liste des spécialisations stockées en base |
| Recherche d’un hôpital avec lits disponibles par spécialisation et position actuel GPS | Retourne un hôpital spécialisé dans un domaine au plus proche des points GPS fournies avec des lits disponible  « Fonctionnalités incluses : Vérification de lits disponible et Evènement lors d’un résultat trouvé » |

Cette POC devra être en partie ou en totalité être réutilisable pour la réalisation du projet cible.

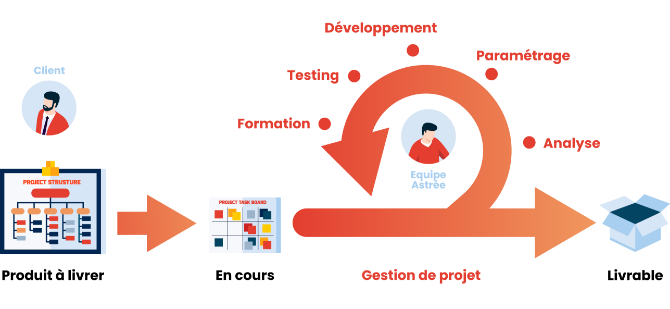
# Organisation du projet et des tests

## Organisation générale

Le projet sera découpé en deux parties :

* **Le Backend** : Le cœur de la solution qui fournira les API nécessaires qui seront consommés par l’interface web
* **Le Frontend** : l’interface Web qui sera fourni aux utilisateurs

Les deux parties seront détaillées ci-dessous avec les différentes phases d’un projet agile :



## Le Backend

## Choix technologiques

En raison d’une contrainte technique d’utilisation de java comme langage pour le backend, dans cette POC la version java 17 a été utilisée, car elle offre un support à long terme (LTS).



Coté Framework, nous utiliserons Spring avec son outil Spring Boot, qui accélère le développement en limitant la préparation de l’environnement serveur. Elle embarque tout le nécessaire à l’exécution de la solution et permet de se concentrer directement sur le code du projet.



Pour la présentation de cette POC, la technologie de la base de données utilisés est H2 Database Engine, un outil simple est rapide qui stocke les données dans la mémoire vive ou disque de la machine. A chaque redémarrage du serveur les données sont réinitialisées.



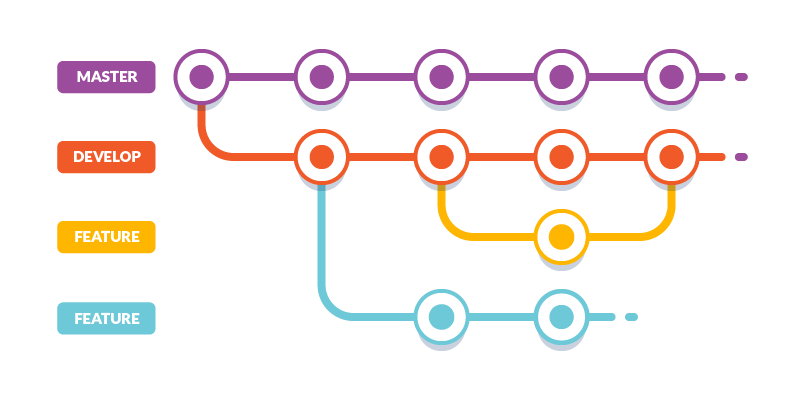
La partie API utilisera la technologie API REST, qui permet une flexibilité en termes de communication client et serveur.



Pour la partie fonctionnelle, nous avons utilisé Google Map API, un des leaders sur le marché en therme de calcul d’itinéraire

## Méthode de développement

Le code sera déposé sur GitLab principalement avec un miroir sur GitHub. Le développement suivra le modèle GitFlow :



* Une branche Release (Master)
* Une branche de développement (Develop)
* Des branches de fonctionnalités pour chaque fonctionnalité nécessaire. (FEATURE)

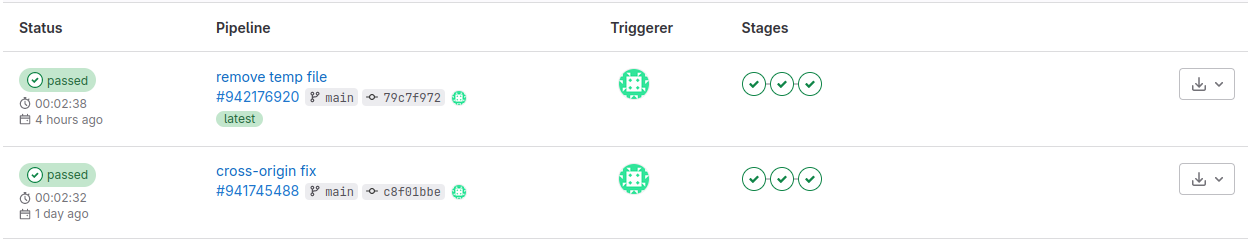
Lorsque la version en développement est stable et après validation hiérarchique, mettre à jour le master pour publie cette release. Une fois « Commit » sur le GitLab les derniers tests seront effectuées à avant la mise en production.

## Intégration Continue et Déploiement continu

Pour chaque publication d’une release, le processus d’intégration continu et de déploiement continu se fera de manière automatisée avec le GitLab CI/CD.

Un pipeline dédié sera exécuté( [gitlab-ci.yml](https://github.com/wilkill/P11_POC_Code/blob/main/doc/gitlab-ci.yml) )

Exemples d’exécutions :

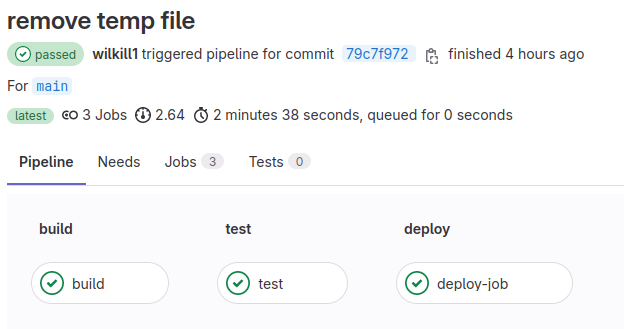


Dans chaque « Version » il exécutera différentes étapes « stage » :

* Build (Construction de l’exécutable)
* Test (Test Unitaire défini dans le code)
* Deploy (Deploiement sur le serveur destination)

Chaque étape sera exécuté dans un environnement qui respecte les prérequis du projet dans une image Docker (un container) de type : maven:3.8.4-openjdk-17

Ci-dessous un détail d’exécution



## Test de la solution

## Test Unitaire

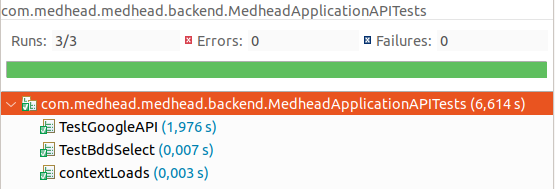
Pour les Test Unitaire nous allons utiliser Junit (un framework spécialisé dans les tests unitaires de code java) Il créé automatique des rapports d’exécution « [surefire-reports](https://github.com/wilkill/P11_POC_Code/tree/main/doc/Test_Application_Report/surefire-reports)».

Chaque fonctionnalité choisie pourra être testée de manière automatique.

Dans ce projet il y aura trois tests :

* + Chargement du contexte Spring
  + Test de la base de données
  + Test de l’API Google avec des données de test prédéfinies et fonctionnelles

Nous obtenons un rapport de ce type :

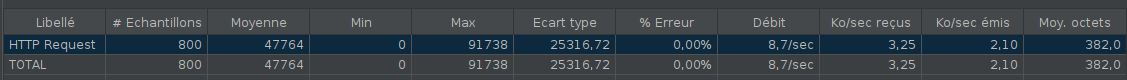


**Conclusion :** Tous concluant.

## Test de Stress

Pour le test de stress, nous avons utilisé la solution JMeter, elle permet de définir une monté en charge, elle simule un appel de x utilisateurs sur un laps de temps donné. Nous pouvons définir plusieurs indicateurs et fournisse chacun un rapport d’exécution. Voir [JMeter Rapport](https://github.com/wilkill/P11_POC_Code/tree/main/doc/Test_Application_Report/JMeter)

Dans cette POC, nous allons effectuer un test de 800 requêtes de l’API en une seconde. Nous obtenons le résultat suivant :



Les 800 requêtes ont été traiter en un temps de 1 minute 32 seconde soit 8,7 requêtes par seconde sans erreur.

Une requête exécutée prend environ 0.300 ms. (Log application)

**Conclusion :** Etant dans un environnement de test, le serveur d’application, la base de données et l’outil Jmeter sont exécutés en simultané, donc on triple la demande de ressource machine à un instant T, malgré cela le serveur traite la demande sans échec. Dans un environnement de production ou les rôles serveur sont dissocié sur des machines dédier et les machines sont redondées avec équilibreur de charge. Nous pouvons nous rapprocher de l’objectif de réponse attendue dans ce projet

D’autre solution de calcule de trajet entre deux points GPS sont disponible en hébergement local, qui permettrai un temps de réponse accrue à l’outil mais nécessiterais des ressources supplémentaires

## Normes

Dans le développement de cette application tous les échanges entre le client et le serveur seront chiffrés via le protocole TLS via un certificat Web HTTPS. Toutes données à caractère confidentiel seront protégées pas une authentification et une gestion de rôle d’utilisateur.

Nous respecterons les données clientes RGPD.

L’exécution de ce projet s’effectuera en suivant les différentes norme et standardisation actuel tel que TOGAF et méthodologie Agile

## Le Frontend

## Choix technologiques



Le framework utilisé pour la partie Frontend sera Angular, l’un des premiers framework dans ce domaine. Elle apporte un panel de fonctionnalités avancées directement. Il permet une modularité plus simple et un développement collaboratif plus facile.

## Méthode de développement

Le développement du frontend est plus simple et nécessité très peu de code métier. Grâce au plugin Git, il permet de partager rapidement le code aux autres développeurs.

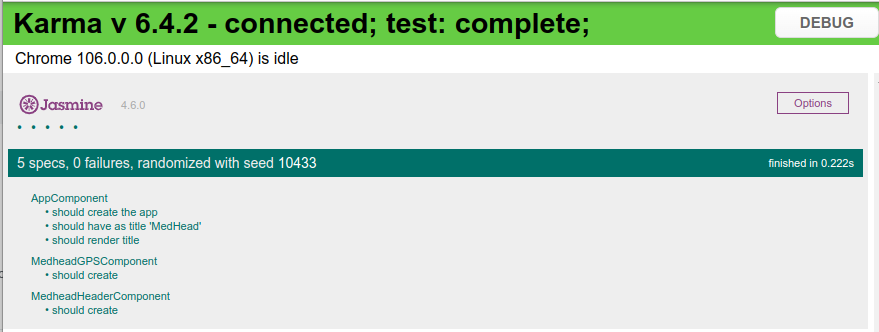
## Intégration Continue et Déploiement continu

## Test de la solution

## Test Unitaire :

Pour l’exécution des tests et vérifier que la solution est fonctionnellement correcte. Angular CLI propose son inclus ‘Karma’, un outil de test via la commande `ng test`. Il test le cœur de la solution et aussi es différent composant qu’il contient.

Nous avons une interface web qui nous montre les tests et le résultat de ces derniers :



## Test Complet :

## Normes