

# Reporting POC



## Table des matières

<b>REPORTING POC</b>	<b>1</b>
<b>1. CONTEXTE</b>	<b>3</b>
<b>2. PERIMETRE DE LA POC</b>	<b>3</b>
<b>3. ORGANISATION DU PROJET ET DES TESTS</b>	<b>4</b>
3.1. ORGANISATION GENERALE	4
3.2. LE BACKEND	4
3.2.1. CHOIX TECHNOLOGIQUES	4
3.2.2. METHODE DE DEVELOPPEMENT	5
3.2.3. INTEGRATION CONTINUE ET DEPLOIEMENT CONTINU	6
3.2.4. TEST DE LA SOLUTION	7
3.2.4.1. TEST UNITAIRE	7
3.2.4.2. TEST DE STRESS	7
3.2.5. NORMES	8
3.3. LE FRONTEND	8
3.3.1. CHOIX TECHNOLOGIQUES	8
3.3.2. METHODE DE DEVELOPPEMENT	9
3.3.3. INTEGRATION CONTINUE ET DEPLOIEMENT CONTINU	9
3.3.4. TEST DE LA SOLUTION	10
3.3.4.1. TEST UNITAIRE :	10
3.3.4.2. TEST COMPLET :	10
3.3.5. NORMES	11

## 1. 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.

## 2. Périmètre de la POC

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

Fonctionnalité	Description
<b>Lister les spécialisations</b>	Retourne la liste des spécialisations stockées en base
<b>Recherche d'un hôpital avec lits disponibles par spécialisation et position actuel GPS</b>	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.

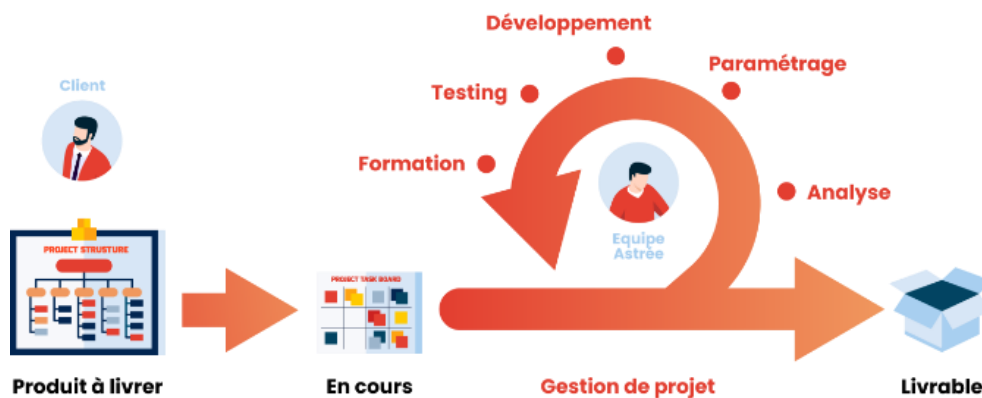
### 3. Organisation du projet et des tests

#### 3.1. 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 :



#### 3.2. Le Backend

##### 3.2.1. 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.

## H2

Pour la présentation de cette POC, la technologie de la base de données utilisés est H2 Database Engine, un outil simple et 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 terme de calcul d'itinéraire

### 3.2.2. 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 publier cette release. Une fois « Commit » sur le GitLab les derniers tests seront effectués avant la mise en production.

### 3.2.3. 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](#) )

Exemples d'exécutions :

Status	Pipeline	Triggerer	Stages	
 passed 00:02:38 4 hours ago	<a href="#">remove temp file</a> #942176920  main  		  	 
 passed 00:02:32 1 day ago	<a href="#">cross-origin fix</a> #941745488  main 		  	 



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

- Build (Construction de l'exécutable)
- Test (Test Unitaire défini dans le code)
- Deploy (Déploiement 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




#### remove temp file

 passed wilkill1 triggered pipeline for commit [79c7f972](#)  finished 4 hours ago

For [main](#)

 latest  3 Jobs  2.64  2 minutes 38 seconds, queued for 0 seconds

Pipeline Needs Jobs 3 Tests 0

build	test	deploy
 build	 test	 deploy-job

### 3.2.4. Test de la solution

#### 3.2.4.1. Test Unitaire

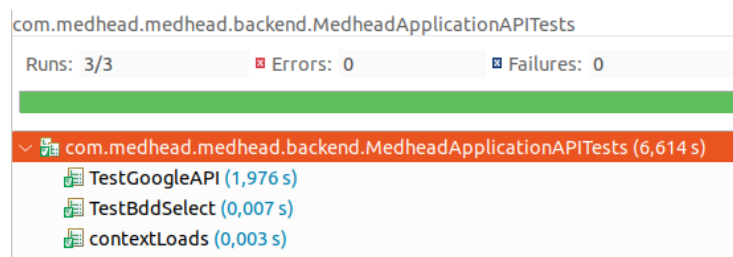
Pour les Test Unitaire nous allons utiliser Junit (un framework spécialisé dans les tests unitaires de code java) Il crée automatique des rapports d'exécution « [surefire-reports](#) ».

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.

#### 3.2.4.2. Test de Stress

Pour le test de stress, nous avons utilisé la solution JMeter, elle permet de définir une montée en charge, elle simule un appel de x utilisateurs sur un laps de temps donné. Nous pouvons définir plusieurs indicateurs et fournir chacun un rapport d'exécution. Voir [JMeter Rapport](#)

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

Libellé	# Echantillons	Moyenne	Min	Max	Ecart type	% Erreur	Débit	Ko/sec reçus	Ko/sec émis	Moy. octets
HTTP Request	800	47764	0	91738	25316,72	0,00%	8,7/sec	3,25	2,10	382,0
TOTAL	800	47764	0	91738	25316,72	0,00%	8,7/sec	3,25	2,10	382,0

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

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

A chaque retour positif nous obtenons une ligne de log suivante :

```
[MedheadApplicationAPI] - [GPSTools] - L'hopital le plus proche est le : [
Hospital [id=5, name=GHER ST BENOIT, specializations=[17, 45], latitude=-
21.05858, longitude=55.710549, bedsAvailable=22]] à [12.122] km - line:104
```

Demain, une réservation sera effectué avec une notification à l'hôpital trouvé avec le décompte des lits disponibles en temps réel.

**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 dédié 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 calcul de trajet entre deux points GPS sont disponible en hébergement local, qui permettra un temps de réponse accrue à l'outil mais nécessiterais des ressources supplémentaires

### 3.2.5. 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 suivant la réglementation RGPD.

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

## 3.3. Le Frontend

### 3.3.1. 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.



### 3.3.2. Méthode de développement

Le développement du frontend est plus simple et nécessite très peu de code métier. Grâce au plugin Git, il permet de partager rapidement le code aux autres développeurs. Nous resterons dans une logique de branche Gitflow :



### 3.3.3. Intégration Continue et Déploiement continu

Comme pour le backend, nous utiliserons GitLab CI/CD pour l'intégration et le déploiement continu avec les 3 phases (stages) de manière automatique à chaque commit d'une release:

- Install (Mise en place de l'environnement)
- Test (Test Unitaire défini dans le code)
- Deploy (Déploiement sur le serveur destination)

Chaque étape sera exécutée dans un environnement qui respecte les prérequis du projet dans une image Docker (un container) de type : node :16-bullseye

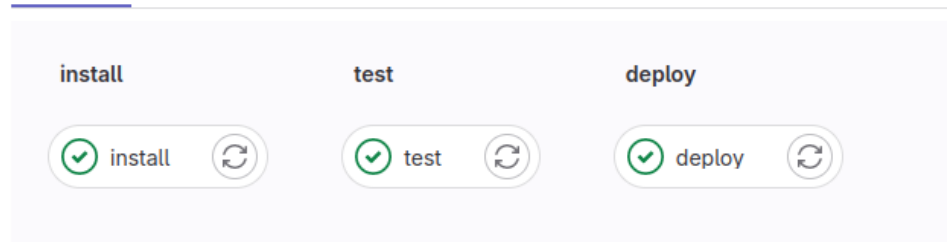
#### last gitlabci

✓ passed wilkill1 triggered pipeline for commit 0923601c finished 29 minutes ago

For main

latest 3 Jobs 2.99 2 minutes 59 seconds, queued for 0 seconds

Pipeline Needs Jobs 3 Tests 0

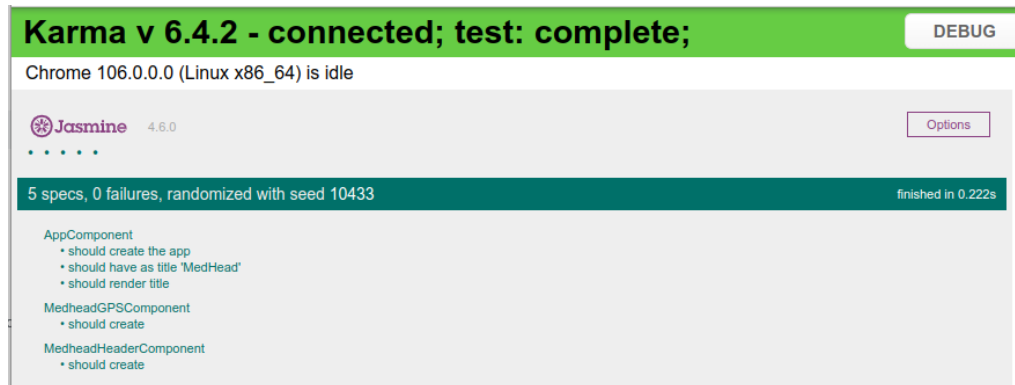


### 3.3.4. Test de la solution

#### 3.3.4.1. 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 :



#### 3.3.4.2. Test Complet :

Le test complet s'effectue aujourd'hui de manière manuelle en vérifiant le bon fonctionnement de l'outil conjointement avec ou sans l'application backend. En cas d'erreur de communication entre le front end et le backend une erreur système sera levée.

Dans un test complet, suivant la recherche nous obtiendrons l'hôpital le plus proche avec ces coordonnées GPS.

Dans une solution plus complète nous pourrions mettre en place des test e2e avec des scénarios complets de test.

Voici aujourd'hui l'interface avec une requête résolue :

**MedHead**

**Module de recherche d'hôpital le plus proche**  
Appel via API la méthode backend de recherche avec Google API

Spécialisation

Latitude

Longitude

Médecine d'urgence

-21.114302

55.654626

Rechercher

Hôpital :

Latitude :

Longitude :

GHER ST BENOIT

-21.05858

55.710549

### 3.3.5. Normes

Le frontend sera développé sous la méthode Agile avec des données factices pour les tests.

Il sera accessible sur le web sous forme internet ou extranet avec le respect des derniers systèmes de sécurisation. Tels que le protocole TLS 1.3 pour la partie certificat. De protection par authentification forte avec une gestion de rôle. Les données seront disponibles seulement si elles sont nécessaires et pour l'utilisation spécifique de l'utilisateur.

Les données stockées respecteront les différentes réglementation RGPD. Tout en précisant les différents traitements associés aux clients.