

# **Plan de tests**

## **Développement d'une Preuve de Concept**

MedHead+

## Auteur(s) et contributeur(s)

Nom & Coordonnées	Qualité & Rôle	Société
Gérald ATTARD	Consultant en Architecture logicielle	XXXXXXXXXX

## Historique des modifications et des révisions

N° version	Date	Description et circonstance de la modification	Auteur
1.0	18/01/2023	Création du document	Gérald ATTARD

## Validation

N° version	Nom & Qualité	Date & Signature	Commentaires & Réserves
1.0	Kara Trace CIO, Ursa Major Health		
	Anika Hansen, PDG, Jupiter Scheduling Inc.		
	Chris Pike Architecte d'entreprise principal, Schedule Shed		

## Tableau des abréviations

Abr.	Sémantique
BPMN	Business Process Model and Notation (trad. <i>modèle de processus métier et notation</i> )
CI/CD	Continuous Integration / Continuous Delivery / Continuous Deployment (trad. <i>intégration continue / livraison continue / déploiement continu</i> )
BDD	Business Driven Development (trad. <i>développement axé sur le métier</i> ) / Behavior Driven Development (trad. <i>développement dirigé par le comportement</i> )
BPM	Business Process Management (trad. <i>gestion de processus métier</i> )
CD	Continuous Deployment (trad. <i>déploiement continue</i> )
CI	Continuous Integration (trad. <i>intégration continue</i> )
DDA	Document de Définition d'Architecture
Dev Team	Development Team (trad. <i>équipe de développement</i> )
ETP	Équivalent Temps Plein
IDE	Integrated Development Environment (trad. <i>environnement de développement intégré</i> )
KPI	Key Performance Indicator (trad. <i>indicateur de performance clé</i> )
MDA	Model Driven Architecture (trad. <i>architecture dirigé par modèle</i> )
PoC	Proof Of Concept (trad. <i>preuve de concept</i> )
QA	Quality Assurance (trad. <i>assurance qualité</i> )
ROI	Return Of Investment (trad. <i>retour sur investissement</i> )
SAW	Statement of Architecture Work (trad. <i>énoncé des travaux d'architecture</i> )
SOA	Services Oriented Architecture (trad. <i>architecture orientée services</i> )
TDD	Test Driven Development (trad. <i>développement dirigé par le test</i> )
UML	Unified Modeling Language (trad. <i>langage de modélisation universel</i> )

## Table des matières

I. Introduction.....	6
II. Rappels.....	7
II.A. Architecture de base.....	7
II.B. Architecture de données et de l'information.....	8
II.C. Notions de tests.....	9
II.C.1. Test unitaire.....	10
II.C.2. Test fonctionnel.....	12
III. Ressources.....	14
III.A. Pipeline CI/CD.....	15
III.B. Structure de l'application de la PoC.....	16
IV. Politique de tests.....	17
IV.A. Jeu de données.....	19
IV.A.1. Copie de la production.....	19
IV.A.2. Génération de données.....	19
IV.A.3. Création d'un sous-ensemble à partir de la production.....	20
IV.B. Listes de contrôle.....	21
V. Fonctionnalités testées.....	27
V.A. Objectifs et critères d'évaluation.....	27
V.A.1. Objectif : Amélioration de la qualité des traitements d'urgence.....	27
V.A.2. Objectif : Amélioration de la confiance des utilisateurs.....	29
V.B. Composants testés.....	30
V.C. Objectifs et composants.....	31
V.C.1. Fonctionnel.....	31
V.C.2. Technique.....	33
VI. Campagne et scénario de tests.....	36
VI.A. Fonctionnel.....	36
VI.A.1. Synthèse.....	39
VI.B. Technique.....	40
VI.B.1. Synthèse.....	44
VII. Scénario et cas de tests.....	45
VIII. Infrastructure préconisée.....	55
VIII.A. Le pipeline DevOps.....	57
VIII.B. L'intégration continue.....	58
VIII.B.1. Développement.....	58
VIII.B.1.a. Phase d'analyse et de conception.....	58
VIII.B.1.b. Méthode d'analyse et de conception.....	58
VIII.B.1.b.i. Outil d'analyse.....	60
VIII.B.1.c. Outil de développement.....	60
VIII.B.2. Suivi de version.....	61
VIII.B.2.a. Les branches de développement.....	61
VIII.B.2.a.i. Un mouvement en deux temps.....	61
VIII.B.2.a.ii. Choisir son tempo.....	61
VIII.B.2.b. Le branchement par <i>release</i> .....	62
VIII.B.2.c. Outil.....	63
VIII.B.3. Package.....	64
IX. Hypothèses et risques.....	65

IX.A. Hypothèse en Ressources Humaines.....	65
IX.B. Hypothèse temporelle.....	65
IX.C. Risques.....	66
IX.C.1. Cartographie des risques.....	68



# I. Introduction

Le présent document complète le SAW et le DDA associés à ce projet de PoC pour le compte du Consortium *MedHead*.

En ce sens, la méthode qui sera utilisée rédiger cette documentation a été le BDD (*Business Driven Development*). Cette méthode de développement axée sur l'entreprise est un paradigme qui se concentre sur les besoins globaux d'une entreprise, et qui permet de développer des applications spécifiques destinées aux consommateurs.

Or, dans le contexte de *MedHead*, le besoin et la causalité de ce projet répondent aux critères d'utilisation de cette méthode.

Ainsi, au sein de ce document, cette étude utilisera le BDD pour réaliser l'écriture des tests nécessaires à la vérification de la sémantique de chaque fonctionnalité nécessaire à la réalisation de la PoC.

En appliquant une politique de modernisation continue au sein de *MedHead*, il s'avérera que l'un des problèmes inhérents au processus de développement d'une solution logicielle, pour le compte du Consortium, sera l'incapacité de suivre le rythme auquel elle devra migrer les différents SI existants au sein des différentes entreprises le constituant, en réponse aux besoins émergents.

Pour que les différents systèmes informatiques de *MedHead* migrent dans de bonne condition, le Consortium devra s'aligner sur les demandes de soins médicaux émergentes, tout en appliquant des technologies éprouvées répondant ainsi à sa propre politique technique.

Cette politique devra alors savoir faire la différence entre la conception de solutions, résolvant un ou plusieurs problèmes de processus métier, et la création de solutions centrées sur l'informatique, voire sur la technologie médicale elle-même.

Dans le cadre de ce projet, il est question de développer une PoC permettant de gérer les interventions d'urgence, et qui soit capable de communiquer, d'interopérer et d'interagir avec les systèmes existants.

En outre, au fur et à mesure que *MedHead* progressera face aux améliorations de ses processus métiers, les applications existantes inflexibles pourraient ne plus être capables d'honorer les changements nécessaires. Dans un tel scénario, le besoin d'un nouveau mécanisme alignant efforts informatiques et médicaux des exigences et de la stratégie d'entreprise apparaîtra forcément.

C'est dans ce contexte que l'utilisation du BDD facilitera cette transition, au travers d'un cadre compris, standardisé et pouvant être réalisé de manière efficace et répétée.

La première étape consistera à créer un modèle de processus métier (BPM) et à le mesurer à l'aide d'indicateurs de performance clés (KPI), de retour sur investissement (ROI) ou d'autres mesures. Enfin, le Consortium pourra utiliser ce BPM comme un mécanisme crucial pour communiquer les exigences résultantes au domaine informatique et médical.

C'est donc, en suivant cette méthode au sein de ce contexte que ce plan de test sera rédigé, AVANT la rédaction de spécifications techniques, et sera mis en œuvre APRÈS la finalisation de chaque brique fonctionnelle développée.

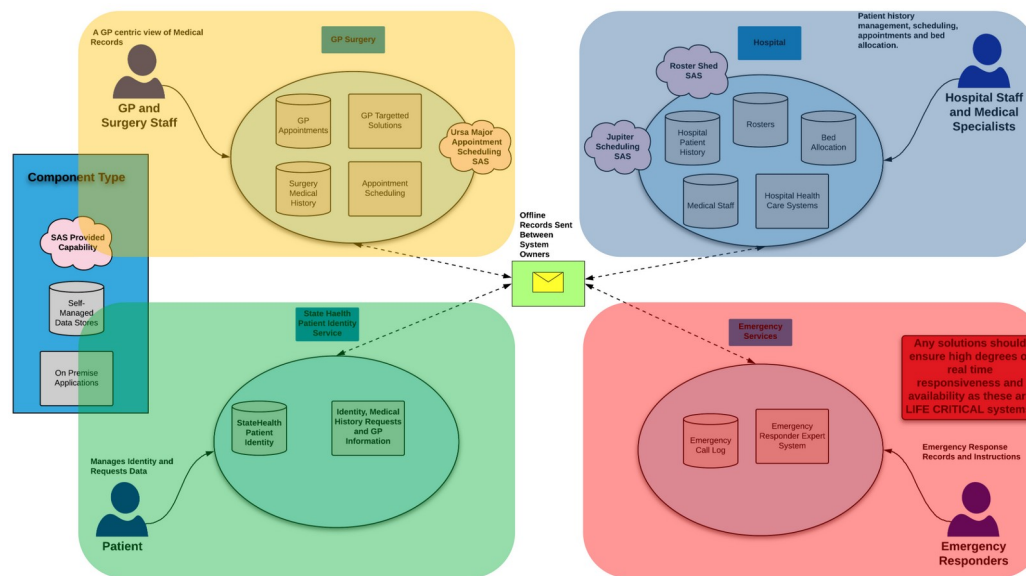
## II. Rappels

### II.A. Architecture de base

Tel qu'il a été présenté dans le DDA, l'architecture de base, celle en place actuellement, est un agrégat de quatre contextes professionnels complémentaires :

- un **contexte chirurgical**, comprenant notamment le domaine de médecine générale ;
- un **contexte hospitalier** dénombrant les infrastructures médicales d'accueil et leurs ressources matérielles ;
- un **contexte d'intervention d'urgence** pour lequel cette notion d'urgence est LE critère surclassant toutes les autres priorités intervenant dans n'importe quel autre contexte et/ou processus ;
- un **contexte d'identité médicale** permettant d'identifier pertinemment et sans équivoque chaque personne intervenant, directement ou indirectement, dans un processus médical quelconque.

Les contextes énumérés ci-dessus peuvent être décomposés et représentés graphiquement comme suit :

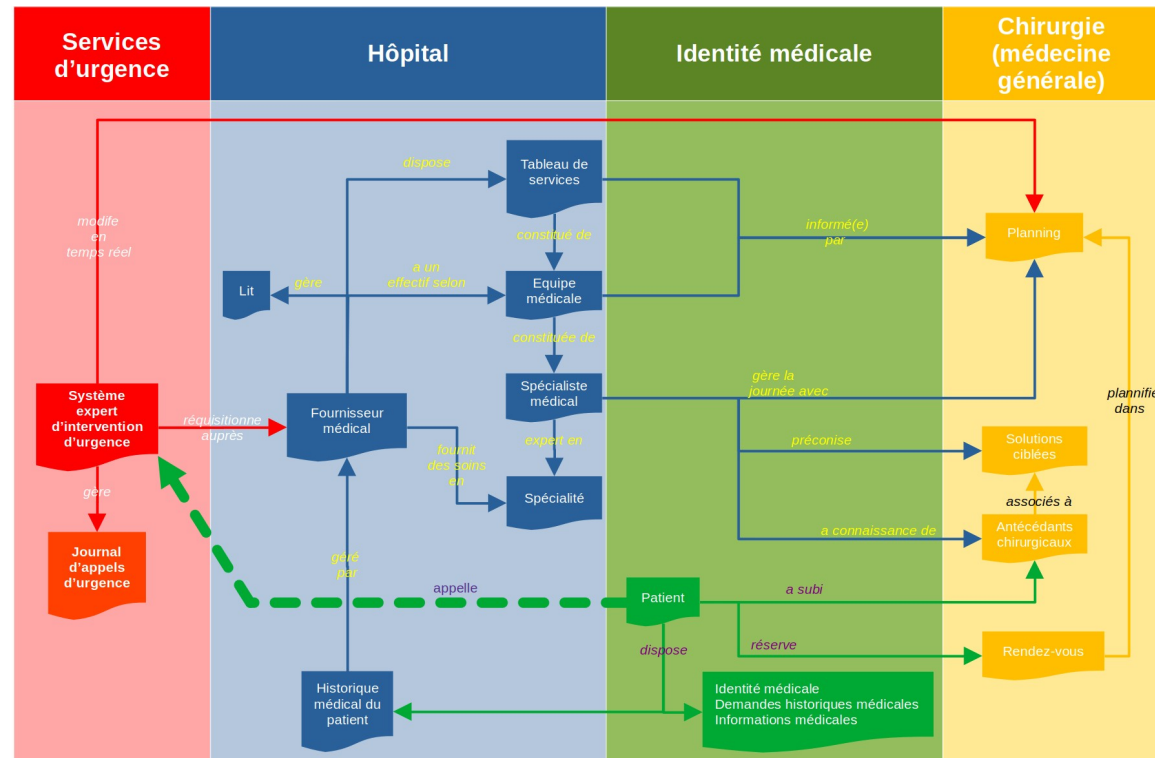


## II.B. Architecture de données et de l'information

Relativement aux quatre contextes présentés dans le paragraphe précédent, la réalisation de la PoC se concentrera exclusivement sur le domaine ci-dessous :

<b>Intervention d'urgence</b>	<ul style="list-style-type: none"> <li>Gestion du registre d'appels d'urgence</li> <li>Système expert d'intervention d'urgence</li> </ul>
-------------------------------	---

Ainsi, dans le diagramme de communication des données présentait dans le SAW, seule la colonne en **ROUGE** sera concernée par la réalisation de la PoC, telle que représenté graphiquement comme suit :





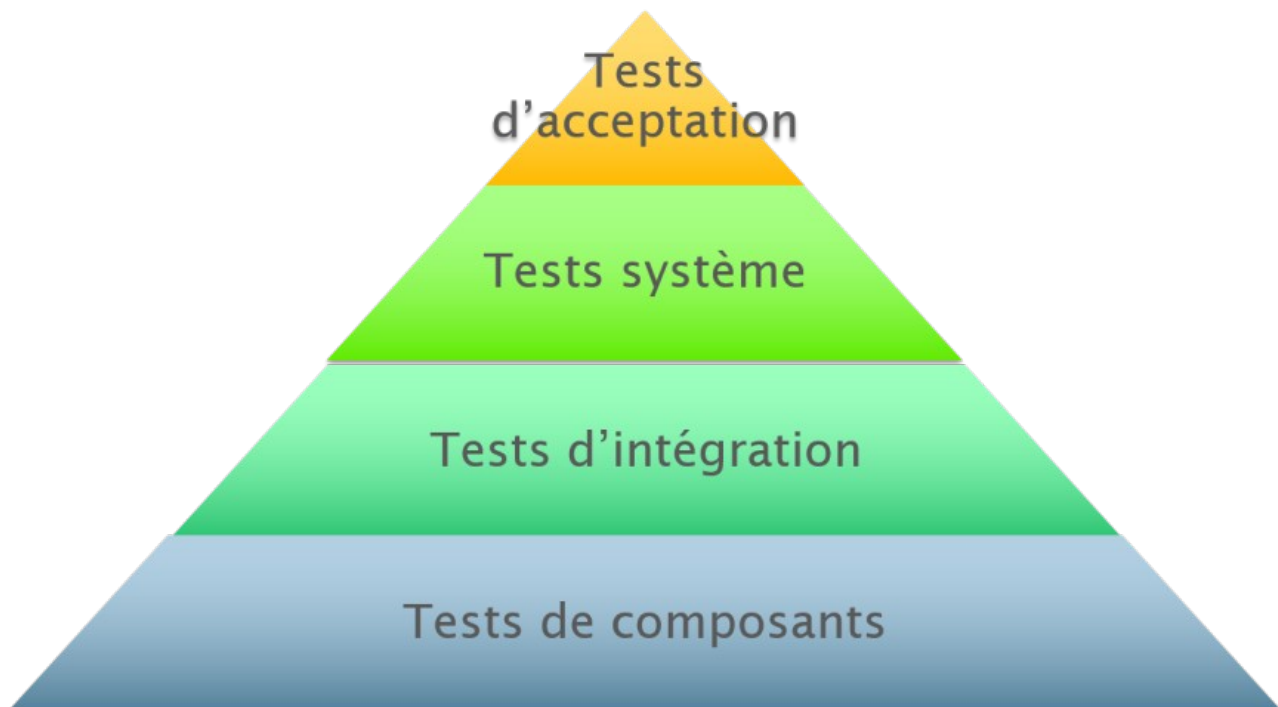
## II.C. Notions de tests

Il existe de nombreux types de tests aux objectifs et stratégies spécifiques, dont **la validation des exigences de base est critique**, tels que :

- **les tests d'acceptation** permettant de vérifier si l'ensemble du système fonctionne comme prévu ;
- **les tests d'intégration** permettant de s'assurer que les composants logiciels ou les fonctions fonctionnent ensemble ;
- **les tests unitaires** permettant de valider que chaque unité logicielle, représentant le plus petit composant testable d'une application, fonctionne comme prévu ;
- **les tests fonctionnels** permettant de vérifier les fonctions en émulant des scénarios métiers, en fonction des besoins fonctionnels - les tests en boîte noire sont un moyen courant de vérifier les fonctions ;
- **les tests de performance** permettant d'évaluer les performances du logiciel sous différentes charges de travail, tels que, par exemple, les tests de charge utilisés pour évaluer les performances dans des conditions de charge réelles ;
- **les tests de non-régression** permettant de vérifier si de nouvelles fonctionnalités brisent ou dégradent la fonctionnalité. En outre, les **tests d'intégrité** peuvent être utilisés pour vérifier les menus, les fonctions et les commandes au niveau de la surface, lorsqu'il n'y a pas suffisamment de temps pour effectuer un test de régression complet ;
- **les tests de charge** permettant de tester la quantité de contraintes que le système peut supporter avant qu'il ne tombe en panne – ces tests sont considérés comme un type de test non fonctionnel.
- **les tests d'utilisation** permettant de valider la capacité d'un client à utiliser un système, telle qu'une application Web, pour effectuer une tâche spécifique.

De plus, tout aussi important, **les tests exploratoires aident un testeur** ou une équipe de tests à **découvrir des scénarios et des situations difficiles à prévoir** pouvant entraîner des erreurs logicielles.

Ainsi, l'ensemble de ces tests peut se représenter selon la pyramide ci-dessous :



Même une application « simple » peut faire l'objet d'un grand nombre et d'une variété de tests. Un plan de gestion de tests permet de hiérarchiser les types de tests qui apportent le plus de valeur, compte tenu du temps et des ressources disponibles. L'efficacité des tests est optimisée en exécutant le moins de tests possible pour trouver le plus grand nombre de défauts.

Bien que cette étude ait vocation à se concentrer sur les tests fonctionnels, elle va néanmoins aborder les tests unitaires qui seront à prévoir lors du développement technique de la PoC.

### **II.C.1. Test unitaire**

Le test unitaire consiste à isoler une partie du code et à vérifier qu'il fonctionne parfaitement.

Il s'agit de petits tests qui valident l'attitude d'un objet et la logique du code.


Les tests unitaires sont effectués pendant la phase de développement des applications logicielles.

Ces tests sont effectués par les développeurs ; ils peuvent également être effectués par les responsables en assurance Qualité (QA) afin de s'assurer particulièrement sur telle ou partie du code.

En termes de bonne pratique Qualité, cette étude préconisera à *MedHead* de favoriser fortement l'utilisation du TDD et/ou du BDD. Néanmoins, ce document privilégiera le BDD et fera abstraction du TDD. En effet, ce dernier est une pratique AGILE orienté vers l'opérationnel et le technique, alors que ce document se veut être un ensemble de recommandations fonctionnelles.

Néanmoins pour décrire quelques peu l'utilisation du TDD, cette étude énoncera les raisons suivantes :

- le test unitaire révèle si la logique derrière le code est appropriée et si elle fonctionnera dans tous les cas ;
- le test unitaire améliore la lisibilité du code et aide les développeurs à comprendre le code de base, ce qui facilite grandement la mise en œuvre des modifications et cela, plus rapidement ;
- des tests unitaires bien conduits sont également de bons outils pour la documentation du projet ;
- en termes de performance, les tests sont effectués en un peu plus de quelques millisecondes, ce qui vous permet d'en réaliser des centaines en très peu de temps ;
- le test unitaire permet au développeur de remanier le code ultérieurement et de s'assurer que le module continue à fonctionner correctement. Des cas de test sont écrits à cet effet pour toutes les fonctions et méthodes afin que les erreurs puissent être rapidement identifiées et réparées, chaque fois que l'une d'elles est créée par l'introduction d'un changement dans le code ;
- la qualité finale du code s'améliorera parce qu'il s'agira en fin de compte d'un code propre et de haute qualité grâce à ces essais continus ;
- puisque le test unitaire divise le code en petits fragments, il est possible de tester différentes parties du projet sans avoir à attendre que d'autres parties soient terminées. Ici l'utilisation de *Mock* prend tout son sens et sa dimension.

En termes de framework d'utilisation, sur ce projet à dominante POO dont le langage de programmation préconisée est le JAVA, il sera conseillé de travailler avec *JUnit* , dont le framework *TestNG*, basé justement sur *JUnit*.

## II.C.2. Test fonctionnel

Les tests fonctionnels sont les tests définis par l'ISO-25010. Ils couvrent les tests d'exactitude, de complétude et d'aptitude à l'usage.

Ils n'ont aucun lien particulier avec les niveaux de tests et peuvent être effectués par les développeurs, les parties prenantes « métiers », les testeurs et tous les intervenants étant amenés à faire du test.

Cependant, une constante domine : les tests fonctionnels sont les tests réalisés par des testeurs...même si cela peut paraître évident à première vue, le métier de QA est un métier à part entière ; cette définition réalise alors une liaison avec les niveaux de test en assimilant les tests fonctionnels aux différents tests d'intégration ou systèmes.

Néanmoins, en fonction des contextes de test, cette définition peut faire apparaître certains contre-arguments à son utilisation, dont notamment :

- le fonctionnel ne peut pas se faire au niveau unitaire (pas de fonctionnel pour les développeurs) ;
- les testeurs ne font que du test fonctionnel alors qu'il serait également possible de leur demander des tests de performances, d'utilisabilité, de portabilité ou tout autre type de test...

Ainsi, les deux axiomes précédents ne peuvent être considérés comme tels dans certains contextes d'entreprise. Il sera donc nécessaire d'analyser le contexte du Consortium *MedHead* pour décider « qui teste quoi ? ».

Une autre facette à prendre en considération est que les tests fonctionnels sont les cas *passants*, les autres sont du « *Negative testing* ». Cette façon de percevoir les tests offre une autre vision tendant à définir le fonctionnel comme les cas d'usages principaux. Or, les cas non désirés ou d'erreurs sont souvent des usages fonctionnels qu'un utilisateur détourne volontairement ou involontairement, et dont il est important de tenir compte lors du développement.

En outre, les tests fonctionnels sont les tests effectués pendant l'exécution de l'application, c'est à dire que les tests fonctionnels seraient des tests dynamiques et que les tests statiques (comme les revues) ne pourraient pas proposer du test fonctionnel. Cette définition est un contre-sens du point de vue de la norme ISO-25010. Prenons l'exemple d'une revue d'une *User Story* en Agile qui met en avant un chemin alternatif manquant. Cette revue pourrait alors être un test statique et contribuerait à l'enrichissement fonctionnel de l'application AVEC l'ajout du chemin alternatif manquant.

De plus, il est bon de préciser que les tests fonctionnels ne sont pas que des tests d'IHM. Cette définition existe également et est réductrice ; elle s'apparente un peu à celle des tests faits par les testeurs. Néanmoins, cette définition peut également englober le niveau des tests d'acceptation et certains types de tests comme les tests d'utilisabilité. Par définition de l'ISO-25010, les tests d'utilisabilité sont du non-fonctionnel et il serait réducteur de considérer la testabilité de l'IHM qu'à travers le prisme du fonctionnel...

Dans le contexte de *MedHead*, il ne sera également pas souhaitable de considérer les tests fonctionnels comme des tests de bout en bout (*End to End*). De par la complexité de l'architecture préconisée pour *MedHead* au sein du DDA, les logiciels utilisés au sein de cette architecture complexe, où les tests composants pourraient être des tests sur des logiciels tiers, chaque composant testé sera à considérer comme un système embarqué ou tout simplement un composant parmi de nombreux partenaires. Cependant, il sera nécessaire de garder à l'esprit qu'un test fonctionnel ne saurait se voir assimiler à un test d'intégration.

Enfin, cette étude insistera sur le fait qu'un test fonctionnel n'est pas forcément un test manuel. Les personnels responsables de la QA pourront faire appel à des outils de tests automatisés pour les effectuer de manière périodique et pérenne. Cette automatisation prendra tout son sens lors des tests fonctionnels de non-régression permettant aux responsables QA de garantir que le système ne tombe pas en panne lors d'une mise à jour logicielle, même si celle-ci est mineure.

### III. Ressources

Les ressources associées à ce plan de test découleront directement des fonctionnalités définies dans le SAW et de l'infrastructure cible décrite au sein du DDA.

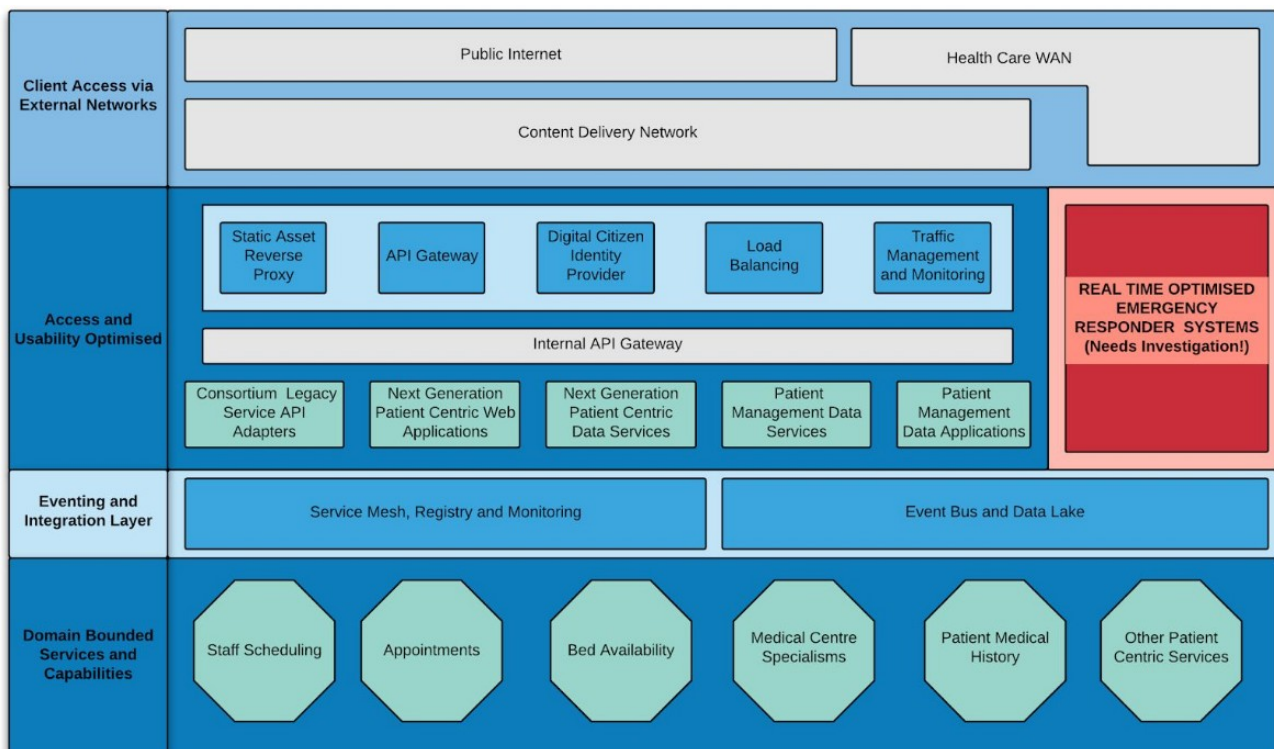
De plus, ce paragraphe n'assumera que les ressources nécessaires à ce projet de PoC. Les différents éléments abstraits, tels que des événements contextuels ou des choix procéduraux, devront être traités dans un document à part.

En outre, en considérant les éléments décrits dans le DDA, il ne sera pas nécessaire de prendre en compte le contexte initial de chaque système utilisé par chaque entreprise composant le Consortium, puisque cela correspond à une organisation et une infrastructure actuelle éprouvées par chacune d'entre elles.

Cette étude prendra donc cet état de fait comme hypothèse de départ sur laquelle les différentes phases de tests viendront s'appuyer.

En outre, les ressources intellectuelles nécessaires à ce projet de PoC, seront l'ensemble des processus métier médicaux, des événements et des choix possibles pour améliorer les soins prodigués à un patient.

Ainsi, en considérant les axiomes précédents, les éléments de ressource identifiées dans ce paragraphe seront principalement associées à des ressources logiciels ; les ressources matérielles étant prévues au sein du composant « *Access and Usability Optimised* » du diagramme initial du SAW ci-dessous :



De plus, cette PoC sera structurée en deux parties complémentaires :

- le pipeline CI/CD : composant indispensable au développement de la PoC ;
- la structure même de la PoC réalisé à l'aide du pipeline CI/CD.

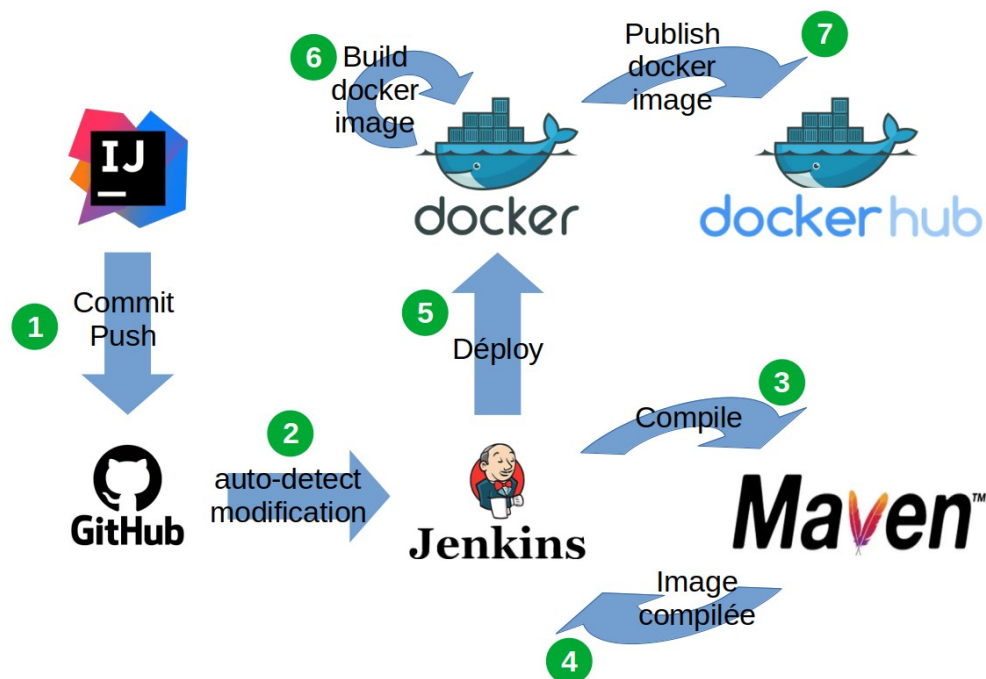
Les deux éléments ci-dessus seront développés dans les paragraphes qui suivent.

### III.A. Pipeline CI/CD

En ce qui concerne spécifiquement le pipeline CI/CD de réalisation de la PoC, celle-ci sera développée à partir d'un pipeline CI/CD constitué de :

Composant	Produit
Framework de programmation	Java – OpenJDK 17
Environnement de développement intégré	IntelliJ IDEA 2022.3.2
Contrôle de version	GitHub 2023
Automatisation de la production logicielle	Apache Maven 3.9.0
Gestion de la production logicielle	Jenkins 2.375.3
Plateforme de conteneurs logiciels	Docker 23.0.1
Stockage et partage des conteneurs logiciels	DockerHub 2023

L'ensemble des composants/produits présentés dans le tableau ci-dessus se verra séquencer (numéro d'étape en vert) selon le diagramme ci-dessous :



Ainsi, le diagramme ci-dessus définit le processus de développement, d'intégration et de déploiement de la PoC en sept étapes qui seront définies dans la SBB intitulée *Bloc de construction du pipeline CI/CD*.

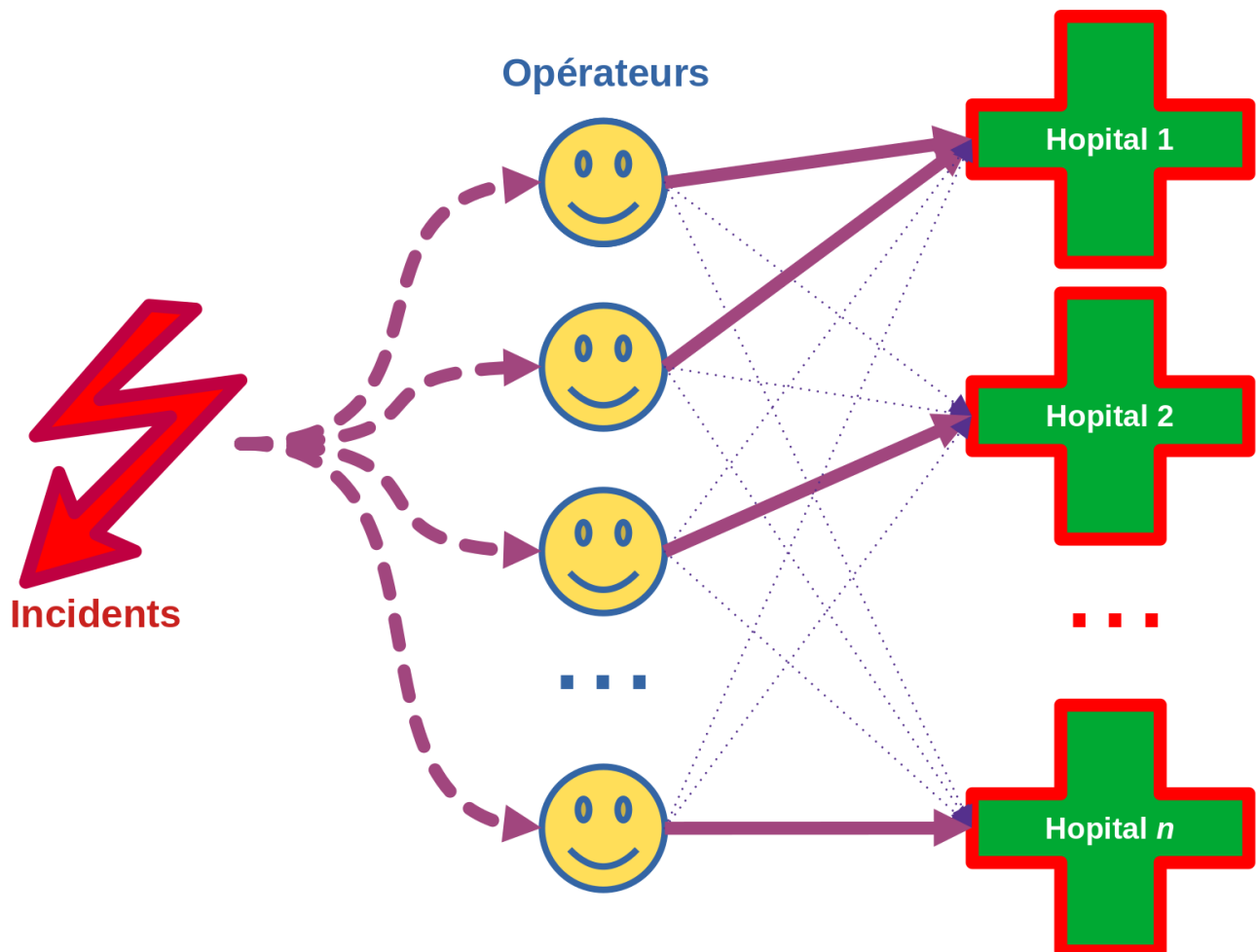
### III.B. Structure de l'application de la PoC

La structure de l'application de PoC aura, comme base de développement, les outils précédemment installés pour le pipeline CI/CD .

Cette structure sera constitué de 3 composants primordiaux représentant :

- les incidents survenus ;
- les hopitaux devant prendre en charge les incidents ;
- les opérateurs chargés de répondre aux messages d'urgence et de les répartir dans les différents hôpitaux.

Ainsi, les trois composants ci-dessus interagiront tel que :



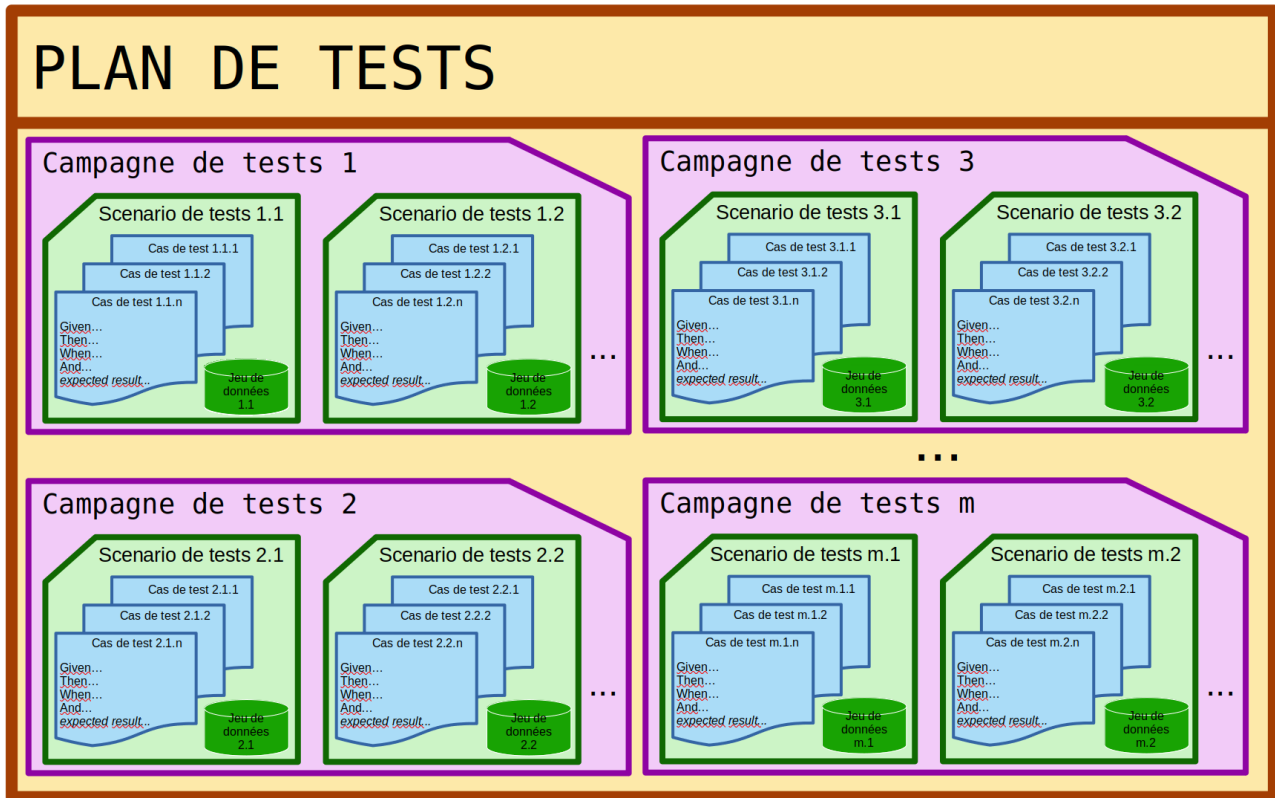
Ainsi, le diagramme ci-dessus définit le processus d'interaction entre chaque ressource, les étapes successives nécessaires à ces interactions seront définies dans la SBB intitulée *Bloc de construction de l'application de la PoC*.





## IV. Politique de tests

La conception du plan de tests associé à ce projet de PoC sera organisée selon le schéma ci-dessous :



Comme vous pourrez en prendre connaissance dans le schéma ci-dessus, le plan de tests sera constitué de :

- **plusieurs campagnes de tests**, dont chacune correspondra à une fonctionnalité spécifique ;
- **chaque campagne de tests sera découpée en un ou plusieurs scénarios de tests**, dont chacun représentera une mise en situation pour la campagne considérée ;
- **chaque scénario sera découpé en un ou plusieurs cas de tests**, dont chacun représentera un ou plusieurs cas d'usage précis pour le scénario considéré ;
- **chaque cas de test correspondra à un et un seul jeu de données** associé au cas de test considéré, lui-même associée au scénario de tests considéré, lui-même.

En suivant le découpage proposé ci-dessus, il sera également primordial de garder à l'esprit que la réalisation d'un test n'est pas une finalité : la réalisation d'un test correspond à la vérification qu'une fonction, un processus ou n'importe quelle entité conceptualisée, répond au besoin pour lequel il ou elle a été développé(e) de façon certaine, répétable et reproductible.

Néanmoins, avant d'arriver à ce résultat, l'origine du test débute toujours par la rédaction d'un plan de test. Ce dernier déterminera une base fixe du déroulement du test en question. Ainsi, il facilitera et organisera le déroulement et l'exécution des scénarios de tests lors d'une campagne ; tout comme la construction d'une maison nécessite un plan d'architecte dans le BTP...

Pour concevoir ce plan, il sera alors nécessaire d'étudier, d'analyser, de poser des limites et des impacts, de déterminer des outils et une ou des technologies adaptées...tout en définissant le temps alloué. En d'autres termes plus succincts, il sera nécessaire d'adopter une méthodologie ET une méthode.

En ce qui concerne ce projet, en se basant sur les analyses réalisées au sein du SAW et du DDA, il s'agira ici de définir les étapes du processus de test et d'identifier les risques afin de détecter le maximum d'anomalies possibles.

Suite, à ces différentes analyses, il s'agira alors de commencer à dessiner le plan en établissant un ordre dans lequel chaque composant sera complété, testé individuellement et, enfin, intégré avec les autres composants du système.

En ce sens, **ce plan de test ne sera pas informatif ; ce plan de test aura une valeur DÉCLARATIVE.**

Ce plan de test permettra alors de définir ce qui sera testé, pourquoi le tester, comment ces tests s'effectueront, quand et qui testera. Il sera alors possible d'appréhender un tel document, comme le menu d'un restaurant ou le sommaire d'un livre.

De plus, le contenu d'un tel plan dépendra alors de la complexité de la PoC, et du niveau à partir duquel les tests seront initiés : des tests de fonctionnalités hautes pourront se baser sur une méthode *Business Driven Development*, des tests fonctionnels de vérification d'implémentation sur une méthode de *Behavior Driven Development* et des tests de programmation sur le *Test Driven Development*.

En ce qui concerne le contexte de cette étude commanditée par *MedHead*, le présent document s'alignera sur la méthode utilisée au sein du cahier des charges et du DDA, à savoir le *Business Driven Development*.

En suivant cette méthodologie et sa méthode associée, le présent document s'efforcera d'être générique, fonctionnel et automatisable dans son implémentation, tout en proposant un plan de navigation, un niveau d'exigences et d'importance, permettant une compréhension globale de ses phases de réalisation.

En outre, la structure des différents éléments constituant ce plan sera indépendante, c'est à dire que chaque module sera indépendant l'un de l'autre. Ainsi, après avoir défini toutes les fonctionnalités à tester (cf SAW), il s'avérera nécessaire d'organiser des campagnes de tests pour chacune des fonctionnalités définies ; **campagnes de tests** qui **seront** elles-mêmes **découpées** en **scénarios de test**, appelés aussi cas de test.

**Enfin, cette étude apportera, volontairement et sciemment, une contrainte à chaque cas de tests considéré: un cas de test ne testera qu'une et une seule situation ou résultat attendu, sur la base d'un contexte figé et fixé à l'avance. Ultérieurement, tous les cas de tests associées à ces mêmes fonctionnalités, utiliseront alors TOUJOURS le même contexte ou la même situation.**

## IV.A. Jeu de données

Le fait est que dans le contexte urgentiste de *MedHead*, des données existe déjà : ce projet démarrera alors sur une base déjà existante pouvant être enrichie de nouvelles fonctionnalités. Ainsi, face à l'exploitation d'une application en production, il y a (souvent) un besoin récurrent : comment avoir des jeu de données pour développer de nouvelles fonctionnalités et les tester AVANT de les déployer en production ?

La réponse à cette question sera choisie parmi trois options dans le contexte de ce projet au sein de *MedHead*, options dont le choix final reviendra à l'ensemble des parties prenantes du Consortium. Ces trois options, énumérées ci-dessous, seront détaillées dans les paragraphes qui suivent, à savoir par :

- copie de la production,
- génération pure et dure de données,
- création d'un sous-ensemble de données à partir de la production - solution de compromis des deux options précédentes.

### IV.A.1. Copie de la production

La solution la plus simple et la plus directe pour créer un jeu de données est de simplement copier celles extraites de la production et de l'utilisation d'une application existante.

Néanmoins, bien que copier des données de production soit une solution de facilité, cela n'est pourtant pas toujours possible, ou simplement permis.

En effet, cette approche ne satisfait pas toujours aux exigences de sécurité, apportées aux patients de *MedHead*, relativement à l'application de la RGPD notamment.

En pratique, les différentes jeu de données ne doivent comporter aucune donnée qui soit issue de sa mise en production : ces données patient doivent rester confinées dans un environnement hautement sécurisé.

Il sera alors nécessaire d'appréhender des jeux de données suffisamment exhaustifs, réalistes et anonymisées, c'est à dire ne contenant aucune information client.

Cette approche n'est donc pas toujours la plus « *facile* » à mettre en place puisque le seul précodé d'anonymisation peut devenir très très chronophage.

### IV.A.2. Génération de données

Cette approche diffère de la précédente puisqu'à chaque fois qu'un jeu de données sera nécessaire pour le déroulement d'un test, celui-ci sera généré avec de fausses données préfabriquées à l'avance à cette intention.

Ces données pourront alors être décrites dans des fichiers plats ou générées une fois pour toutes à l'initialisation d'un nouveau scénario de tests.

L'avantage de cette approche est présent au sein des processus de fabrication et d'exécution d'un tel jeu de données pouvant être rapides.

Néanmoins, il faudra bien prendre en considération que plus le modèle de données attendu sera complexe, plus sa maintenance le sera également, de façon exponentielle et non pas linéaire.

De plus, s'il est prévu que le modèle de données évolue souvent, cela peut devenir tout simplement ingérable, surtout si le jeu de données en contient un volume important.

Concrètement, cette approche nécessite de recréer une base locale de jeux de données à chaque scénario de test ; travail titanesque si réalisé manuellement et chronophage si effectué automatiquement.

#### IV.A.3. Création d'un sous-ensemble à partir de la production

En ce qui concerne *MedHead*, cette étude préconisera la construction d'un processus permettant de créer un sous-ensemble réduit et anonymisé de données à partir de la production.

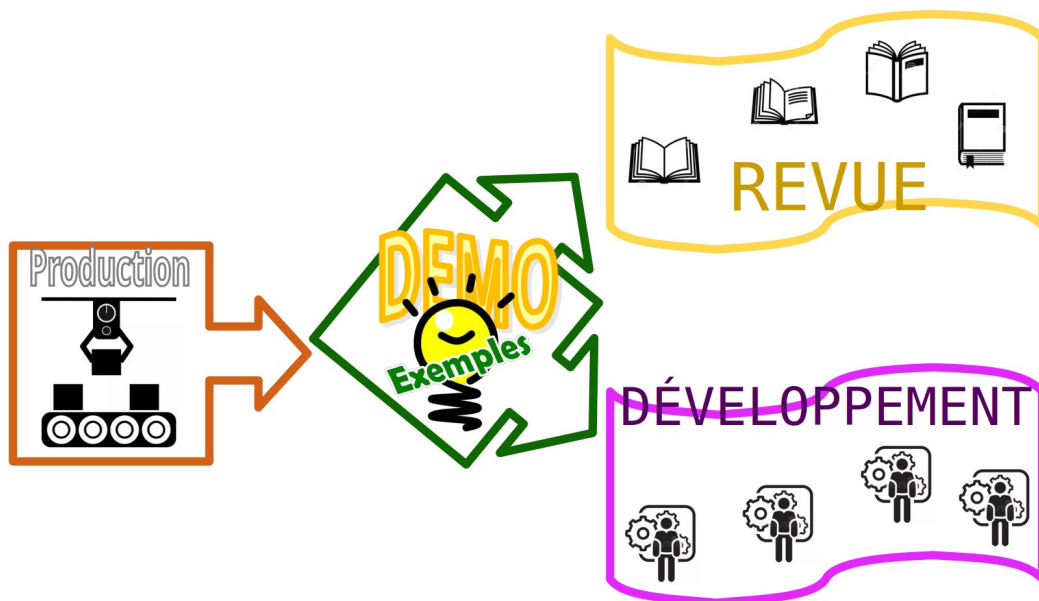
Cette approche permettra d'avoir des données réalistes et riches, en qualité comme en quantité, sans risque pour les données client.

D'une part, l'application chargée de ce processus sera utilisée quotidiennement par les patients et les personnels médicaux. Ainsi, l'équipe informatique, en collaboration avec les personnels médicaux, pourra développer de nouvelles fonctionnalités, à son rythme, en ayant toujours des jeux de données pérennes à disposition pour les tests.

Il faut tout de même être conscient que cette fonction va nécessiter des processus et des outils, en plus de la rigueur et de l'organisation indispensables. En outre, il faudra également ne pas perdre de vue que le cœur de ce système demeurera les **données**.

La mise en œuvre d'une telle solution consistera à maintenir une instance de démonstration, un prototype sommaire, dont le code sera le même que celui de l'instance de production, et dont les données sont différentes. En ce sens, ces dernières seront issues de la production, puis réduites en quantité et enfin anonymisées.

Cette approche, préconisée pour ce projet de PoC, présentera l'avantage de produire de nombreux jeux de données, complets, cohérents et réalistes, tout en leur retirant leur sensibilité, tel qu'imaginée dans le schéma ci-dessous :



## IV.B. Listes de contrôle

Les listes de contrôle proposées ci-dessous sont basées sur les architectures proposées au sein du document de définition d'architecture, elles sont donc également force de proposition face à l'absence d'éléments techniques concrets à la rédaction de cette étude. Ainsi, elles seront à compléter et à ajuster durant tout le projet afin d'être les plus exhaustives possibles lors de l'implémentation des composants.

Ces listes de contrôle couvriront les domaines suivants :

- les technologies employées ;
- l'exécution des tests ;
- la convivialité ;
- les licences et les prix des outils logiciels utilisés ;
- le support et le service après-vente.

TECHNOLOGIE		✓
Technologies GUI de l'application à tester	<b>Applications Java</b> : Swing, JavaFX. AWT, SWT, Eclipse plug-ins, RCP, Applets, JavaWebStart, RIA, ULC, CaptainCasa pour les composants JavaFX, SubScene, JIDE Common Layer	
	<b>Applications Web</b> : Browser: Chrome, Firefox, Opera, Safari, Edge, Chromium basiert, Microsoft Edge Legacy, Internet Explorer; Versions, Headless Browser pour Chrome, Firefox et Edge (basé sur Chromium) Les outils devront supportés des environnements HTML 5, AJAX, Angular, React et Vue.js, mais aussi des toolkits concrets comme Smart, (GWT), ExtGWT, ExtJS, ICEfaces, jQuery UI, jQueryEasyUI, Kendo UI, PrimeFaces, Qooxdoo, RAP, RichFaces, Vaadin et ZK. D'autres toolkits pourront être intégrés avec peu d'effort par exemple: SAP UI5, Siebel Open, UI et Salesforce. Il faudra également tester des applications spécifiques tels que Electron et/ou Webswing...	
	<b>Applications Android</b> : Applications Android natives, applications web mobiles et applications hybrides sur Android à partir d'Android 7 sur des appareils réels et avec l'émulateur d'Android Studio	

## TECHNOLOGIE



Technologies GUI de l'application à tester	<b>Applications Windows</b> : Win32 classique, .NET (souvent développé en C#), Windows Presentation Foundation (WPF), Windows Forms, Windows Apps / Universal Windows Platform (UWP) avec éléments de contrôle XAML, applications C++ modernes (p.ex. avec Qt)	
	<b>Systèmes hybrides</b> considérant la combinaison des plusieurs technologies GUI, tels que des composants incorporés dans le navigateur (JavaFX WebView, JXBrowser, SWT-Browser)	
	Les documents PDF peuvent être testés comme n'importe quelle autre application (Vérifications textuelles et graphiques pour des éléments individuels)	
Support GUI selon le système d'exploitation	<b>Applications Java: Swing</b> et <b>JavaFX</b> : Windows, Linux, Unix, macOS <b>SWT</b> : Windows, Linux-GTK; Solaris-GTK sur demande.	
	<b>Applications Web</b> : Windows, Linux, macOS	
	<b>Applications Windows</b> : Windows	
Principe de test	Capture/Relecture pour la génération directe et efficace de séquences de test pour un traitement ultérieur dans des cas de test plus complexes avec contrôle de flux, paramétrage, modularisation et capacités de script avancées. Tous les tests et leurs environnements associés devront pouvoir être personnalisés.	
Structuration de test	Clair et concis grâce à la représentation graphique des cas de test et des nœuds d'action dans une structure arborescente. Les scénarios de tests pourront être structurés de manière modulaire en utilisant des suites de tests et des bibliothèques	
Reconnaissance des composants, robustesse des tests	Détection stable des composants, indépendamment des propriétés géométriques, même pour des éléments complexes comme les arbres et les tableaux dynamiques. Les tests sont robustes et tolérants aux modifications de l'interface graphique.	

## TECHNOLOGIE



Réutilisabilité, effort de maintenance	Haute réutilisabilité des modules de test grâce à une structure modulaire, par exemple par des procédures, l'encapsulation des accès aux composants, etc	
Tests basés sur des données	Importation directe de fichiers CSV ou Excel, utilisation de requêtes de base de données SQL, fichiers XML. Toute autre source peut être intégrée via une extension de script.	
Tests basés sur les mots-clés/ Tests basés sur le comportement	Utilisation de mots-clés (Keywords) pour mettre en œuvre et contrôler les cas de test, également au moyen de documents ou d'outils externes de spécification des tests (par exemple, Excel ou des outils de gestion des tests)	
Conteneurs Docker	Tous les composants devront supporter des tests conteneurisés via Docker ou tout autre système de conteneur	
Tests de charge et de performance	Tests de charge et de performance grâce à une exécution synchronisée et parallèle, même sur plusieurs machines. Pour le web en combinaison avec des outils comme JMeter ou NeoLoad.	
Protocoles, documentation des tests, Rapports	Des protocoles clairs et détaillés comprenant des captures d'écran de la situation d'erreur sont toujours générés. Des rapports configurables dans différents formats (HTML, XML, JUnit), la documentation des tests et des procédures peuvent être générés par simple pression d'un bouton ou automatiquement.	
JIRA/REST	JIRA et JIRA PlugIns comme TestRail, Zephyr, X-Ray, TM4J peuvent être intégrés via REST, souvent aussi via des outils CI comme Jenkins.	
Extensibilité par des scripts	Extensions de fonctions libres et contrôles/actions personnalisés par des scripts intégrés (Jython, Groovy et JavaScript). Via l'API de script, accès complet à tous les objets de l'application (SUT) et exécution de son propre code dans l'application ou dans le navigateur.	
Gestion des tests	Homogénéiser les fonctionnalités de base pour les petits scénarios de tests inclus dans chaque composant à tester.	
Intégration continue	Intégration disponible ou possible pour : ALM/QualityCenter de MicroFocus/HP, TestBench d'Imbus, QMetry, Klaros de Verit, TestLink, IBM Rational Quality Manager, Jira et Jira PlugIns tels que TestRail, Zephyr, X-Ray, TM4J, entre autres.	
Bureaux virtuels	Tous les types de bureaux virtuels existants, par exemple: Citrix, VMware, VirtualBox	
Gestion des versions	Bonne versionnabilité, par exemple via Git, SVN/Subversion, CVS, Mercurial, grâce au format XML des fichiers pertinents.	
Suivi des bogues	Possibilité de se connecter via des interfaces ouvertes et REST, par exemple Jira, MantisBT, Bugzilla	

## EXÉCUTION DES TESTS



Préparation des tests	Un assistant de démarrage rapide permet de générer une séquence de démarrage adaptée à l'application testée, en fonction de la technologie GUI sous-jacente.	
Préparation des scénarios de test	Gestion des dépendances pour la préparation et le post-traitement des tests pour des cas de test exécutables indépendamment, y compris la gestion automatique des erreurs.	
Points de vérification	L'enregistrement direct des contrôles standard, les vérifications spécifiques au client peuvent être mises en œuvre de manière variable par du scripting.	
Comparaison d'images	Enregistrement direct des contrôles d'image possible. De nombreux algorithmes, y compris pour les comparaisons d'images floues, et une vue différentielle pratique pour la vérification en cas d'écarts.	
Cartographie des objets	Les informations sur les composants sont stockées de manière centralisée pendant l'enregistrement dans une zone dédiée de la suite de tests, modifiable à tout moment. Mécanismes de recherche de références et de mise à jour automatique.	
Localisation intelligente des objets	Avec les SmartIDs (hashcode d'identifiant unique), les composants peuvent être adressés directement sur la base de propriétés caractéristiques, c'est-à-dire le label associé. La définition d'une portée permet de restreindre la recherche de l'objet, par exemple en cas d'éléments multiples.	
Composants génériques	Le mappage des composants spécifiques de l'interface graphique en composants génériques (boutons, champs de texte...) permet la réutilisation des tests entre les technologies ainsi que l'utilisation d'actions généralement valides sans avoir à capturer chaque composant individuel.	
Exécution du test via la ligne de commande	Exécution en mode batch possible avec des options de configuration étendues via des paramètres de ligne de commande, également pour l'intégration dans des environnements de construction.	
Exécution à distance	Exécution du test également sur des ordinateurs distants en mode daemon/service.	
Traitement des erreurs	Le traitement automatique des erreurs garantit la poursuite de l'ensemble du test sans interruption. Les erreurs sont enregistrées pour une analyse ultérieure.	
Débogueur de test	Fonctionnalité complète de débogage, y compris les points d'arrêt et l'analyse des variables.	



CONVIVIALITÉ		✓
Confort d'utilisation	Utilisation facile et intuitive avec une vue arborescente clairement structurée pour une édition pratique des cas de test, par exemple par copier/coller et glisser/déposer. Capture/relecture pour un démarrage rapide	
Connaissances préalables requises	Aucune connaissance en programmation n'est requise pour une utilisation standard.	
Travail en équipe	Pour les scripts, la connaissance des langages de script standard Jython, Groovy et JavaScript est utile.	

LICENCES ET PRIX		✓
Variante de produits	La multiplicité des environnements utilisés par les utilisateurs doivent apparaître dans les cas de tests et donc les cas d'usage. Ainsi, chaque environnement possible devra idéalement être disponible en différentes variantes de produits variables configurables pour les technologies d'interface graphique prises en charge, par exemple Swing, JavaFX, SWT, Web et Windows.	
Types de licences	Licence développeur - pour créer (et exécuter) des cas de test. Licence runtime - pour exécuter des tests (nocturnes).	
Mécanisme de licences	Toutes les licences devront être flottantes (c'est-à-dire qu'elles ne devront pas être liées à une personne spécifique). Les licences standard doivent fonctionner dans un réseau (local), pour une utilisation inter-réseau (internet), un serveur de licences de licences spécifiques devra être à disposition.	
Acheter ou louer	<i>MedHead</i> proposera l'option d'achat et la location des vidéos sur une base annuelle, pour les licences de test de charge également des termes plus courts.	

SUPPORT ET SERVICE		✓
Télécharger et tester gratuitement	Téléchargement gratuit et anonyme de la version démo. La version de démonstration peut être exécutée sans enregistrement de la part d'un utilisateur.	
Installation et application portable	Installation facile sous Windows, macOS et Linux en quelques clics. Possibilité d'utiliser les différents outils via un navigateur web par webapps.	
Support	Assistance directe des développeurs et des testeurs de <i>MedHead</i> en français, anglais et allemand.	

SUPPORT ET SERVICE		✓
Contrat de maintenance	Le contrat de maintenance (support + mises à jour) sur une base annuelle comprend du support par e-mail et par téléphone.	
Formation, Consultation	Formation ou consultation individuelle sur votre site. Formation ouverte régulièrement chez <i>MedHead</i> . Toutes les offres sont également disponibles en ligne.	
Documentation	Manuel complet, tutoriel de démarrage, vidéos, blog, fonction de recherche en ligne, aide en ligne par clic droit dans les différents outils de <i>MedHead</i> , FAQ générale et technique.	



## V. Fonctionnalités testées

Dans ce paragraphe, les tests seront basés sur les deux objectifs principaux d'amélioration de la PoC, à savoir :

- la qualité des traitements d'urgence ;
- la confiance des utilisateurs.

### V.A. Objectifs et critères d'évaluation

Les deux objectifs mentionnés ci-dessus seront évalués selon plusieurs critères listés dans les paragraphes ci-dessous.

#### V.A.1. Objectif : Amélioration de la qualité des traitements d'urgence

L'amélioration de la qualité des traitements d'urgence est fonction des critères énumérés ci-dessous :

OBJECTIF : améliorer la qualité des traitements d'urgence		✓
Critères d'évaluation	Un temps moyen de traitement est calculé	
	Un temps de réponse est calculé	
	La charge de travail est calculée	
	Le temps de réponse est inférieur à 200ms ET une instance de n'importe quel service peut traiter au moins 800 requêtes par seconde	
	Tous les services respectent les normes	
	Les instructions pour mettre en production la PoC sont rédigées	
	La mise en œuvre de la PoC est planifiée	
	Les spécialités médicales sont listées	
	Le nombre de lits de chaque hôpital est une constance connue	
	La géolocalisation de chaque hôpital est une constante	
	Chaque hôpital a indiqué les spécialités médicales qu'il possédait	
	Le lieu de chaque incident est géolocalisable	
	une API RESTful est développée	
	L'API RESTful tient informée les intervenants médicaux en TEMPS REEL	

## OBJECTIF : améliorer la qualité des traitements d'urgence



### Critères d'évaluation

Des tests unitaires existent	
Les tests unitaires couvrent X% du code	
Des tests d'intégration existent	
Les tests d'intégration sont pertinents	
Des tests d'acceptation existent	
Les tests d'acceptation sont pertinents	
Des tests End-To-End existent	
Les tests End-To-End sont pertinents	
L'ensemble des tests est automatisé	
Tous les tests appartiennent à la pyramide de test	
Des tests de stress sont réalisés	
Les tests de stress garantissent la continuité de l'activité	
Les pics d'utilisation sont mesurés	
Le code de l'application est partageable	
Les pipelines CI sont documentés	
Les pipelines CD sont documentés	
La stratégie de tests est documentée	
Les équipes de développement maîtrise la PoC	
Les équipes de développement se servent de la PoC comme une base de construction pour l'ensemble des modules concernés	

## V.A.2. Objectif : Amélioration de la confiance des utilisateurs

L'amélioration de la confiance des utilisateurs est fonction des critères énumérés ci-dessous :

<b>OBJECTIF : améliorer la confiance des utilisateurs</b>			✓
<b>Critères d'évaluation</b>	90% des cas d'urgence sont acheminés vers un hôpital		
	90% des cas d'urgence sont dirigés vers un hôpital disposant de la spécialité médicale appropriée		
	90% des cas d'urgence sont acheminés vers l'hôpital le plus proche		
	Le temps moyen de traitement est inférieur à 18,25 minutes		
	Le temps moyen de traitement est inférieur à 12 minutes		
	Le temps de réponse est inférieur à 200ms		
	Chaque instance de service peut traiter au moins 800 requêtes par seconde		
	Les normes sont respectés		
	La mise en œuvre de la PoC a respecté la planification		
	Les spécialités médicales sont affichées		
	Le nombre de lits de chaque hôpital est affiché		
	Chaque hôpital affiche ses coordonnées		
	Les spécialités médicales de chaque hôpital sont publiées		
	Les données patient sont correctement protégées		
	La continuité de l'activité est garantie malgré les pics d'utilisation		

## **V.B.**

## **V.C. Composants testés**

Associé à ces objectifs, il est possible de discerner plusieurs composants fonctionnels permettant de cadrer ces mêmes objectifs, tels que :

- l'adéquation de l'infrastructure technologique,
- l'adéquation des soins,
- l'amélioration de processus,
- l'assurance qualité,
- la contrainte temporelle (à définir pour pouvoir respecter cet objectif),
- la documentation technique,
- la géolocalisation,
- la gestion des données,
- l'homogénéité,
- la législation,
- la métrologie,
- l'optimisation de processus,
- la planification,
- la prise en charge du patient,
- la robustesse,
- la sécurité,
- la technologie,
- la transparence.

## V.D. Objectifs et composants

Le tableau ci-dessous sera à compléter durant tout le déroulé du projet pour obtenir une synthèse exhaustive des fonctionnalités à tester, de leur composant associé et du domaine idoine à ceux-ci. Ainsi, ils seront présentés au sein des tableaux ci-dessous, et triés, selon leur ordre d'apparition au sein du document d'énoncé dans l'hypothèse de développement (les futurs éléments devront être ajoutés à la suite de ceux-ci).

### V.D.1. Fonctionnel

Critères	Objectifs		Composant testé	✓
	Amélioration la qualité des traitements d'urgence	Gagner la confiance des utilisateurs		
		90 % des cas d'urgence sont acheminés vers un hôpital	Prise en charge	
		90 % des cas d'urgence sont dirigés vers un hôpital disposant de la spécialité médicale appropriée	Adéquation des soins	
		90 % des cas d'urgence sont acheminés vers l'hôpital le plus proche	Géolocalisation	
	Le nombre de cas d'urgence est connu.		Gestion des données	
	La liste des spécialités médicales est connue.		Gestion des données	
	La géolocalisation de chaque hôpital est connue.		Gestion des données	
	Les spécialités médicales par hôpital est connue.		Gestion des données	
	Un temps moyen de traitement est calculé		Métrologie	

Critères	Objectifs		Composant testé	✓
	Amélioration la qualité des traitements d'urgence	Gagner la confiance des utilisateurs		
		le temps moyen de traitement est inférieur à 18,25 minutes	Amélioration de processus	
		le temps moyen de traitement est inférieur à 12 minutes	Optimisation de processus	
	Un temps de réponse est calculé		Métrologie	
		Le temps de réponse est inférieur à 200ms	Adéquation de l'infrastructure technologique	
		Chaque instance de service peut traiter au moins 800 requêtes par seconde	Robustesse technologique	
	Le temps de réponse est inférieur à 200ms ET une instance de n'importe quel service peut traiter au moins 800 requêtes par seconde		Robustesse et Adéquation de l'infrastructure technologique	
		Les normes sont respectés	Législation	
	Tous les services respectent les normes		Homogénéité	
	Les instructions pour mettre en production la PoC sont rédigées		Documentation technique	
	La mise en œuvre de la PoC est planifiée		Planification	
		La mise en œuvre de la PoC a respecté la planification	Assurance qualité	



## V.D.2. Technique

Critères	Objectifs		Composant testé	✓
	Amélioration la qualité des traitements d'urgence	Gagner la confiance des utilisateurs		
		Le nombre de lits de chaque hôpital est affiché	Transparence	
	La géolocalisation de chaque hôpital est connue		Gestion des données	
		Chaque hôpital affiche ses coordonnées	Transparence	
	Chaque hôpital a indiqué les spécialités médicales qu'il possédait		Gestion des données	
		Les spécialités médicales de chaque hôpital sont publiées	Transparence	
	Le lieu de chaque incident est géolocalisable		Géolocalisation	
	une API RESTful est développée		Technologie	
	L'API RESTful tient informée les intervenants médicaux en TEMPS REEL		Contrainte temporelle (à définir pour pouvoir respecter cet objectif)	
		Les données patient sont correctement protégées	Sécurité	
	Des tests unitaires existent		Sécurité	

Critères	Objectifs		Composant testé	✓
	Amélioration la qualité des traitements d'urgence	Gagner la confiance des utilisateurs		
	Les tests unitaires couvrent X% du code		Sécurité (X à définir)	
	Des tests d'intégration existent		Sécurité	
	Les tests d'intégration sont pertinents		Sécurité	
	Des tests d'acceptation existent		Sécurité	
	Les tests d'écacceptation sont pertinents		Sécurité	
	Des tests End-To-End existent		Sécurité	
	Les tests End-To-End sont pertinents		Sécurité	
	L'ensemble des tests est automatisé		Technologie	
	Tous les tests appartiennent à la pyramide de test		Sécurité	
	Des tests de stress sont réalisés		Sécurité	
	Les tests de stress garantissent la continuité de l'activité		Sécurité	
	Les pics d'utilisation sont mesurés		Métrologie	
		La continuité de l'activité est garantie malgré les pics d'utilisation	Sécurité et Robustesse	
	Le code de l'application est partageable		Transparence	

Critères	Objectifs		Composant testé	✓
	Amélioration la qualité des traitements d'urgence	Gagner la confiance des utilisateurs		
	Les pipelines CI sont documentés		Transparence	
	Les pipelines CD sont documentés		Transparence	
	La stratégie de tests est documentée		Transparence et Sécurité	
	Les équipes de développement maîtrise la PoC		Assurance qualité	
	Les équipes de développement se servent de la PoC comme une base de construction pour l'ensemble des modules concernés		Assurance qualité	

## VI. Campagne et scénario de tests

Les campagnes de tests ci-dessous sont issues du document d'hypothèse de développement rédigé par *MedHead*. Ces campagnes auront pour objectif de répondre à l'assertion « *Nous saurons que nous avons réussi quand nous verrons...* ».

De plus, pour chaque campagne de tests préconisées, les scénarii impliqués seront spécifiquement identifiés selon les libellés utilisés au sein de ce document, en spécifiant :

- **CF $n$**  : campagne de tests fonctionnelle numéro  $n$  ;
- **SF $n.m$**  : scénario de tests fonctionnel de la campagne de tests  $n$  ;
- **CT $n$**  : campagne de tests technique numéro  $n$  ;
- **ST $n.m$**  : scénario de tests technique de la campagne de tests  $n$ .

### VI.A. Fonctionnel

CF1	« ...plus de 90 % des cas d'urgence sont acheminés vers l'hôpital compétent le plus proche... »	
	SF1.1	<b>Libellé du scénario :</b> <ul style="list-style-type: none"><li>• 90% des cas d'urgence sont acheminés vers un hôpital.</li></ul> <b>Composant fonctionnel testé :</b> <ul style="list-style-type: none"><li>• <b>Prise en charge du patient.</b></li></ul>
	SF1.2	<b>Libellé du scénario :</b> <ul style="list-style-type: none"><li>• 90% des cas d'urgence sont dirigés vers un hôpital disposant de la spécialité médicale appropriée.</li></ul> <b>Composant fonctionnel testé :</b> <ul style="list-style-type: none"><li>• <b>Adéquation des soins.</b></li></ul>
	SF1.3	<b>Libellé du scénario :</b> <ul style="list-style-type: none"><li>• 90% des cas d'urgence sont acheminés vers l'hôpital le plus proche.</li></ul> <b>Composant fonctionnel testé :</b> <ul style="list-style-type: none"><li>• <b>Géolocalisation.</b></li></ul>
	SF1.4	<b>Libellé du scénario :</b> <ul style="list-style-type: none"><li>• Présence des données indispensables.</li></ul> <b>Composant fonctionnel testé :</b> <ul style="list-style-type: none"><li>• <b>Gestion des données.</b></li></ul>

CF2	« ...le temps moyen de traitement d'une urgence passe de 18,25minutes (valeur actuelle) à 12minutes (valeur souhaitée)... »	
	SF2.1	<b>Libellé du scénario :</b> <ul style="list-style-type: none"> <li>Un temps moyen de traitement est calculé.</li> </ul> <b>Composant fonctionnel testé :</b> <ul style="list-style-type: none"> <li>Métrologie.</li> </ul>
	SF2.2	<b>Libellé du scénario :</b> <ul style="list-style-type: none"> <li>le temps moyen de traitement est inférieur à 18,25 minutes.</li> </ul> <b>Composant fonctionnel testé :</b> <ul style="list-style-type: none"> <li>Amélioration du processus.</li> </ul>
	SF2.3	<b>Libellé du scénario :</b> <ul style="list-style-type: none"> <li>le temps moyen de traitement est inférieur à 12 minutes.</li> </ul> <b>Composant fonctionnel testé :</b> <ul style="list-style-type: none"> <li>Optimisation du processus.</li> </ul>

CF3	« ...nous obtenons un temps de réponse de moins de 200 millisecondes avec une charge de travail allant jusqu'à 800 requêtes par seconde, par instance de service... »	
	SF3.1	<b>Libellé du scénario :</b> <ul style="list-style-type: none"> <li>Un temps de réponse est calculé.</li> </ul> <b>Composant fonctionnel testé :</b> <ul style="list-style-type: none"> <li>Métrologie.</li> </ul>
	SF3.2	<b>Libellé du scénario :</b> <ul style="list-style-type: none"> <li>Le temps de réponse est inférieur à 200ms.</li> </ul> <b>Composant fonctionnel testé :</b> <ul style="list-style-type: none"> <li>Adéquation de l'infrastructure technologique.</li> </ul>
	SF3.3	<b>Libellé du scénario :</b> <ul style="list-style-type: none"> <li>Chaque instance de service peut traiter au moins 800 requêtes par seconde.</li> </ul> <b>Composant fonctionnel testé :</b> <ul style="list-style-type: none"> <li>Robustesse technologique.</li> </ul>
	SF3.4	<b>Libellé du scénario :</b> <ul style="list-style-type: none"> <li>Le temps de réponse est inférieur à 200ms ET une instance de n'importe quel service peut traiter au moins 800 requêtes par seconde.</li> </ul> <b>Composant fonctionnel testé :</b> <ul style="list-style-type: none"> <li>Robustesse ET Adéquation de l'infrastructure technologique.</li> </ul>

CF4	« ...la mise en œuvre explique les normes qu'elle respecte et pourquoi... »	
	SF4.1	<b>Libellé du scénario :</b> <ul style="list-style-type: none"> <li>• Les normes sont respectés.</li> </ul> <b>Composant fonctionnel testé :</b> <ul style="list-style-type: none"> <li>• Législation.</li> </ul>
	SF4.2	<b>Libellé du scénario :</b> <ul style="list-style-type: none"> <li>• Tous les services respectent les normes.</li> </ul> <b>Composant fonctionnel testé :</b> <ul style="list-style-type: none"> <li>• Homogénéité.</li> </ul>

CF5	« ...les instructions pour mettre en production la PoC sont fournies... »	
	SF5.1	<b>Libellé du scénario :</b> <ul style="list-style-type: none"> <li>• Les instructions pour mettre en production la PoC sont rédigées.</li> </ul> <b>Composant fonctionnel testé :</b> <ul style="list-style-type: none"> <li>• Documentation technique.</li> </ul>

CF6	« ...la mise en œuvre est terminée dans le délai imparti... »	
	SF6.1	<b>Libellé du scénario :</b> <ul style="list-style-type: none"> <li>• La mise en œuvre de la PoC est planifiée.</li> </ul> <b>Composant fonctionnel testé :</b> <ul style="list-style-type: none"> <li>• Planification.</li> </ul>
	SF6.2	<b>Libellé du scénario :</b> <ul style="list-style-type: none"> <li>• La mise en œuvre de la PoC a respecté la planification.</li> </ul> <b>Composant fonctionnel testé :</b> <ul style="list-style-type: none"> <li>• Assurance qualité.</li> </ul>

## VI.A.1. Synthèse

Le tableau synthétise l'ensemble des campagnes de tests fonctionnelles mentionnées précédemment :

Objectifs		Composant testé	Campagne	Scénario
Amélioration la qualité des traitements d'urgence	Gagner la confiance des utilisateurs			
	90 % des cas d'urgence sont acheminés vers un hôpital	Prise en charge du patient	CF1	SF1.1
	90 % des cas d'urgence sont dirigés vers un hôpital disposant de la spécialité médicale appropriée	Adéquation des soins	CF1	SF1.2
	90 % des cas d'urgence sont acheminés vers l'hôpital le plus proche	Géolocalisation	CF1	SF1.3
Le nombre de cas d'urgence est connu.		Gestion des données	CF1	SF1.4
La liste des spécialités médicales par hôpital est connue.		Gestion des données	CF1	SF1.4
La géolocalisation de chaque hôpital est connue.		Gestion des données	CF1	SF1.4
Un temps moyen de traitement est calculé		Métrologie	CF2	SF2.1
	le temps moyen de traitement est inférieur à 18,25 minutes	Amélioration de processus	CF2	SF2.2
	le temps moyen de traitement est inférieur à 12 minutes	Optimisation de processus	CF2	SF2.3
Un temps de réponse est calculé		Métrologie	CF3	SF3.1
	Le temps de réponse est inférieur à 200ms	Adéquation de l'infrastructure technologique	CF3	SF3.2
La charge de travail est calculée		Métrologie	CF3	SF3.1
	Chaque instance de service peut traiter au moins 800 requêtes par seconde	Robustesse technologique	CF3	SF3.3
Le temps de réponse est inférieur à 200ms ET une instance de n'importe quel service peut traiter au moins 800 requêtes par seconde		Robustesse et Adéquation de l'infrastructure technologique	CF3	SF3.4
	Les normes sont respectés	Législation	CF4	SF4.1
Tous les services respectent les normes		Homogénéité	CF4	SF4.2
Les instructions pour mettre en production la PoC sont rédigées		Documentation technique	CF5	SF5.1
La mise en œuvre de la PoC est planifiée		Planification	CF6	SF6.1
	La mise en œuvre de la PoC a respecté la planification	Assurance qualité	CF6	SF6.2

## VI.B. Technique

CT1	« ...Le sous-système...est destiné à recevoir une ou plusieurs spécialités médicales... »	
	ST1.1	<b>Libellé du scénario :</b> <ul style="list-style-type: none"> <li>Les données publiques sont affichées.</li> </ul> <b>Composant fonctionnel testé :</b> <ul style="list-style-type: none"> <li>Transparence.</li> </ul>
	ST1.2	<b>Libellé du scénario :</b> <ul style="list-style-type: none"> <li>Le nombre de lits de chaque hôpital est connu.</li> </ul> <b>Composant technique testé :</b> <ul style="list-style-type: none"> <li>Gestion des données.</li> </ul>

CT2	« ...une banque de données d'informations récentes sur les hôpitaux afin de suggérer l'hôpital le plus proche offrant un lit disponible, associé à une ou plusieurs spécialisations correspondantes... »	
	ST2.1	<b>Libellé du scénario :</b> <ul style="list-style-type: none"> <li>Un hôpital offrant un lit disponible est suggéré pour chaque évènement d'intervention d'urgence.</li> </ul> <b>Composant technique testé :</b> <ul style="list-style-type: none"> <li>Prise en charge du patient.</li> </ul>
	ST2.2	<b>Libellé du scénario :</b> <ul style="list-style-type: none"> <li>Le lit de l'hôpital suggéré est en adéquation avec les spécialités médicales détenues par l'hôpital ET répondant aux dommages du patient.</li> </ul> <b>Composant technique testé :</b> <ul style="list-style-type: none"> <li>Amélioration de processus.</li> </ul>

CT3	« ...Le lieu de l'incident d'urgence doit également être fourni... »	
	ST3.1	<b>Libellé du scénario :</b> <ul style="list-style-type: none"> <li>Le lieu de chaque incident est géolocalisable.</li> </ul> <b>Composant technique testé :</b> <ul style="list-style-type: none"> <li>Géolocalisation.</li> </ul>



CT4	« ...Fournir une API RESTful qui tient les intervenants médicaux informés en temps réel sur : le lieu où se rendre et ce qu'ils doivent faire... »	
	ST4.1	<b>Libellé du scénario :</b> <ul style="list-style-type: none"> <li>une API RESTful est développée.</li> </ul> <b>Composant technique testé :</b> <ul style="list-style-type: none"> <li>Technologie.</li> </ul>
	ST4.2	<b>Libellé du scénario :</b> <ul style="list-style-type: none"> <li>L'API tient informée les intervenants médicaux en TEMPS REEL.</li> </ul> <b>Composant technique testé :</b> <ul style="list-style-type: none"> <li>Contrainte temporelle (à définir pour pouvoir valider ce scénario).</li> </ul>

CT5	« ...S'assurer que toutes les données du patient sont correctement protégées... »	
	ST5.1	<b>Libellé du scénario :</b> <ul style="list-style-type: none"> <li>Les données patient sont correctement protégées.</li> </ul> <b>Composant technique testé :</b> <ul style="list-style-type: none"> <li>Sécurité.</li> </ul>

CT6	« ...S'assurer que votre PoC est entièrement validée avec des tests d'automatisation reflétant la pyramide de test (tests unitaires, d'intégration, d'acceptation et E2E) et avec des tests de stress pour garantir la continuité de l'activité en cas de pic d'utilisation... »	
	ST6.1	<b>Libellé du scénario :</b> <ul style="list-style-type: none"> <li>Des tests techniques existent.</li> </ul> <b>Composant fonctionnel testé :</b> <ul style="list-style-type: none"> <li>Sécurité.</li> </ul>
	ST6.2	<b>Libellé du scénario :</b> <ul style="list-style-type: none"> <li>Les tests techniques sont pertinents.</li> </ul> <b>Composant fonctionnel testé :</b> <ul style="list-style-type: none"> <li>Sécurité.</li> </ul>
	ST6.3	<b>Libellé du scénario :</b> <ul style="list-style-type: none"> <li>Les tests unitaires couvrent x% (x à définir) du code.</li> </ul> <b>Composant fonctionnel testé :</b> <ul style="list-style-type: none"> <li>Sécurité.</li> </ul>
	ST6.4	<b>Libellé du scénario :</b> <ul style="list-style-type: none"> <li>Les tests sont automatisés.</li> </ul> <b>Composant fonctionnel testé :</b> <ul style="list-style-type: none"> <li>Technologie et Sécurité.</li> </ul>

CT6	« ...S'assurer que votre PoC est entièrement validée avec des tests d'automatisation reflétant la pyramide de test (tests unitaires, d'intégration, d'acceptation et E2E) et avec des tests de stress pour garantir la continuité de l'activité en cas de pic d'utilisation... »	
	ST6.5	<b>Libellé du scénario :</b> <ul style="list-style-type: none"> <li>Tous les tests appartiennent à la pyramide de test.</li> </ul> <b>Composant fonctionnel testé :</b> <ul style="list-style-type: none"> <li>Sécurité.</li> </ul>
	ST6.6	<b>Libellé du scénario :</b> <ul style="list-style-type: none"> <li>Des tests de stress sont réalisés.</li> </ul> <b>Composant fonctionnel testé :</b> <ul style="list-style-type: none"> <li>Sécurité.</li> </ul>
	ST6.7	<b>Libellé du scénario :</b> <ul style="list-style-type: none"> <li>Les tests de stress garantissent la continuité de l'activité.</li> </ul> <b>Composant fonctionnel testé :</b> <ul style="list-style-type: none"> <li>Sécurité.</li> </ul>
	ST6.8	<b>Libellé du scénario :</b> <ul style="list-style-type: none"> <li>Les pics d'utilisation sont mesurés.</li> </ul> <b>Composant fonctionnel testé :</b> <ul style="list-style-type: none"> <li>Métrologie.</li> </ul>
	ST6.9	<b>Libellé du scénario :</b> <ul style="list-style-type: none"> <li>La continuité de l'activité est garantie malgré les pics d'utilisation.</li> </ul> <b>Composant fonctionnel testé :</b> <ul style="list-style-type: none"> <li>Sécurité et Robustesse.</li> </ul>

CT7	« ...S'assure que la PoC peut être facilement intégrée dans le développement futur : rendre le code facilement partageable, fournir des pipelines d'intégration et de livraison continue (CI/CD) et documenter votre stratégie de test.... »	
	ST7.1	<b>Libellé du scénario :</b> <ul style="list-style-type: none"> <li>Le code de l'application est partageable.</li> </ul> <b>Composant fonctionnel testé :</b> <ul style="list-style-type: none"> <li>Transparence.</li> </ul>
	ST7.2	<b>Libellé du scénario :</b> <ul style="list-style-type: none"> <li>Les pipelines CI sont documentés.</li> </ul> <b>Composant technique testé :</b> <ul style="list-style-type: none"> <li>Transparence.</li> </ul>

CT7	« ...S'assure que la PoC peut être facilement intégrée dans le développement futur : rendre le code facilement partageable, fournir des pipelines d'intégration et de livraison continue (CI/CD) et documenter votre stratégie de test.... »	
	ST7.3	<b>Libellé du scénario :</b> <ul style="list-style-type: none"> <li>Les pipelines CD sont documentés.</li> </ul> <b>Composant fonctionnel testé :</b> <ul style="list-style-type: none"> <li>Transparence.</li> </ul>
	ST7.4	<b>Libellé du scénario :</b> <ul style="list-style-type: none"> <li>La stratégie de tests est documentée.</li> </ul> <b>Composant technique testé :</b> <ul style="list-style-type: none"> <li>Transparence et Sécurité.</li> </ul>

CT8	« ...S'assurer que les équipes de développement chargées de cette PoC sont en mesure de l'utiliser comme un jeu de modules de construction pour d'autres modules... »	
	ST8.1	<b>Libellé du scénario :</b> <ul style="list-style-type: none"> <li>Les équipes de développement maîtrise la PoC.</li> </ul> <b>Composant fonctionnel testé :</b> <ul style="list-style-type: none"> <li>Assurance qualité.</li> </ul>
	ST8.2	<b>Libellé du scénario :</b> <ul style="list-style-type: none"> <li>Les équipes de développement se servent de la PoC comme une base de construction pour l'ensemble des modules concernés.</li> </ul> <b>Composant technique testé :</b> <ul style="list-style-type: none"> <li>Assurance qualité.</li> </ul>

## VI.B.1. Synthèse

Objectifs		Composant testé	Campagne	Scénario
Amélioration la qualité des traitements d'urgence	Gagner la confiance des utilisateurs			
	Les spécialités médicales sont affichées	Transparence	CT1	ST1.1
Le nombre de lits de chaque hôpital est une constance connue		Gestion des données	CT1	ST1.2
	Le nombre de lits de chaque hôpital est affiché	Transparence	CT1	ST1.1
	Chaque hôpital affiche ses coordonnées	Transparence	CT1	ST1.1
	Les spécialités médicales de chaque hôpital sont publiées	Transparence	CT1	ST1.1
Un hôpital offrant un lit disponible est suggéré pour chaque événement d'intervention d'urgence		Prise en charge du patient	CT2	ST2.1
Le lit de l'hôpital suggéré est en adéquation avec les spécialités médicales détenues par l'hôpital ET répondant aux dommages du patient		Amélioration de processus	CT2	ST2.2
Le lieu de chaque incident est géolocalisable		Géolocalisation	CT3	ST3.1
une API RESTful est développée		Technologie	CT4	ST4.1
L'API tient informée les intervenants médicaux en TEMPS REEL		Contrainte temporelle (à définir pour pouvoir valider ce scénario)	CT4	ST4.2
	Les données patient sont correctement protégées	Sécurité	CT5	ST5.1
Des tests unitaires existent		Sécurité	CT6	ST6.1
Les tests unitaires couvrent X% du code		Sécurité (X à définir)	CT6	ST6.3
Des tests d'intégration existent		Sécurité	CT6	ST6.1
Les tests d'intégration sont pertinents		Sécurité	CT6	ST6.2
Des tests d'acceptation existent		Sécurité	CT6	ST6.1
Les tests d'acceptation sont pertinents		Sécurité	CT6	ST6.2
Des tests End-To-End existent		Sécurité	CT6	ST6.1
Les tests End-To-End sont pertinents		Sécurité	CT6	ST6.2
Les tests sont automatisés		Technologie et Sécurité	CT6	ST6.4
Tous les tests appartiennent à la pyramide de test		Sécurité	CT6	ST6.5
Des tests de stress sont réalisés		Sécurité	CT6	ST6.6
Les tests de stress garantissent la continuité de l'activité		Sécurité	CT6	ST6.7
Les pics d'utilisation sont mesurés		Métrie	CT6	ST6.8
	La continuité de l'activité est garantie malgré les pics d'utilisation	Sécurité et Robustesse	CT6	ST6.9
Le code de l'application est partageable		Transparence	CT7	ST7.1
Les pipelines CI sont documentés		Transparence	CT7	ST7.2
Les pipelines CD sont documentés		Transparence	CT7	ST7.3
La stratégie de tests est documentée		Transparence et Sécurité	CT7	ST7.4
Les équipes de développement maîtrisent la PoC		Assurance qualité	CT8	ST8.1
Les équipes de développement se servent de la PoC comme une base de construction pour l'ensemble des modules concernés		Assurance qualité	CT8	ST8.2

## VII. Scénario et cas de tests

Les tests d'acceptation de chaque fonctionnalité décrite précédemment suivront le modèle défini au sein du SAW.

En outre, seuls seront énoncés les tests relatifs aux campagnes de tests fonctionnelles, les campagnes techniques devront être abordées dans un autre document spécifique.

Rappel :

- **CF $n$**  : campagne de tests fonctionnelle numéro  $n$  ;
- **SF $n.m$**  : scénario de tests fonctionnel de la campagne de tests  $n$  ;
- **CT $n$**  : campagne de tests technique numéro  $n$  ;
- **ST $n.m$**  : scénario de tests technique de la campagne de tests  $n$  ;
- **CtF $n.m.p$**  : cas de test  $p$  du scénario  $m$  de la campagne de tests fonctionnelle  $n$ .

CF1 – Gérald ATTARD	
<b>SF1.1</b>	<b><i>90% des cas d'urgence sont acheminés vers un hôpital</i></b>
<b>CtF1.1.1</b>	<b><i>Acheminer le patient vers un hôpital</i></b>
<b>Étant donné</b>	<i>un cas d'urgence déclaré</i>
<b>Quand</b>	<i>Le cas d'urgence est prise</i>
<b>Ensuite</b>	<i>Le cas d'urgence est acheminé vers un hôpital</i>
<b>Résultat attendu</b>	<i>90 % de tous les cas d'urgence pris en charge seront acheminé vers un hôpital</i>

CF1 – Gérald ATTARD	
<b>SF1.2</b>	<i>90% des cas d'urgence sont dirigés vers un hôpital disposant de la spécialité médicale appropriée</i>
<b>CtF1.2.1</b>	<i>Diriger patient vers un hôpital disposant de la spécialité médicale appropriée</i>
<b>Étant donné</b>	<i>un cas d'urgence déclaré</i>
<b>Quand</b>	<i>Le cas d'urgence est pris en charge</i>
<b>Ensuite</b>	<i>Le cas d'urgence est acheminé vers un hôpital</i>
<b>Et</b>	<i>L'hôpital dispose de la spécialité médicale appropriée au cas d'urgence</i>
<b>Résultat attendu</b>	<i>90 % de tous les cas d'urgence pris en charge seront acheminé vers un hôpital disposant de la spécialité médicale appropriée au cas d'urgence</i>

CF1 – Gérald ATTARD	
<b>SF1.3</b>	<i>90% des cas d'urgence sont acheminés vers l'hôpital le plus proche</i>
<b>CtF1.3.1</b>	<i>Acheminer le patient vers l'hôpital le plus proche</i>
<b>Étant donné</b>	<i>un cas d'urgence déclaré</i>
<b>Quand</b>	<i>Le cas d'urgence est pris en charge</i>
<b>Ensuite</b>	<i>L'hôpital le plus proche de l'incident est identifié</i>
<b>Et</b>	<i>Le cas d'urgence est acheminé vers l'hôpital</i>
<b>Résultat attendu</b>	<i>90% des cas d'urgence sont acheminés vers l'hôpital le plus proche</i>

CF1 – Gérald ATTARD	
<b>SF1.4</b>	<i>Le nombre de cas d'urgence est connu</i>
<b>CtF1.4.1</b>	<i>Calculer le nombre de patient pris en charge</i>
<b>Étant donné</b>	<i>un cas d'urgence déclaré</i>
<b>Quand</b>	<i>Le cas d'urgence est pris en charge</i>
<b>Ensuite</b>	<i>Le nombre total de cas d'urgence est incrémenté de 1</i>
<b>Résultat attendu</b>	<i>Connaître le nombre total de cas d'urgence pris en charge</i>

CF1 – Gérald ATTARD	
<b>SF1.4</b>	<i>La liste des spécialités médicales par hôpital est connue</i>
<b>CtF1.4.2</b>	<i>Tenir à jour la liste des spécialités médicales présentes au sein de chaque hôpital</i>
<b>Étant donné</b>	<i>un hôpital comprend plusieurs spécialités médicales</i>
<b>Quand</b>	<i>Un hôpital doit être identifié</i>
<b>Ensuite</b>	<i>La liste des spécialités médicales présentes au sein de l'hôpital est parcourue</i>
<b>Résultat attendu</b>	<i>Connaître les spécialités médicales présentes au sein de chaque hôpital recensé</i>

CF1 – Gérald ATTARD	
<b>SF1.4</b>	<i>La localisation de chaque hôpital est connue</i>
<b>CtF1.4.3</b>	<i>Fournir la géolocalisation de chaque hôpital</i>
<b>Étant donné</b>	<i>Les coordonnées géographiques d'un hôpital</i>
<b>Quand</b>	<i>Un hôpital doit être identifié</i>
<b>Ensuite</b>	<i>La position de l'hôpital est comparé au lieu d'un cas d'urgence déclaré</i>
<b>Résultat attendu</b>	<i>L'hôpital le plus proche d'un cas d'urgence déclaré est identifié</i>

CF2 – Gérald ATTARD	
<b>SF2.1</b>	<i>Un temps moyen de traitement est calculé</i>
<b>CtF2.1.1</b>	<i>Calculer le temps moyen de traitement de la prise en charge d'un cas d'urgence</i>
<b>Étant donné</b>	<i>La déclaration d'un cas d'urgence</i>
<b>Quand</b>	<i>Un intervenant médical d'urgence prend en charge le patient</i>
<b>Ensuite</b>	<i>L'heure de déclaration du cas d'urgence est stockée</i>
<b>Et</b>	<i>L'heure de prise en charge du patient stockée</i>
<b>Résultat attendu</b>	<i>Un temps moyen de prise des patients est calculé</i>



CF2 – Gérald ATTARD	
SF2.2	<i>Le temps moyen de traitement est inférieur à 18,25 minutes</i>
CtF2.2.1	<i>Prendre en charge un patient en moins de 18,25 minutes</i>
Étant donné	<i>La déclaration d'un cas d'urgence</i>
Quand	<i>Un intervenant médical d'urgence prend en charge le patient</i>
<b>Résultat attendu</b>	<i>La prise en charge du patient doit être réalisée en moins de 18,25 minutes</i>

CF2 – Gérald ATTARD	
SF2.3	<i>Le temps moyen de traitement est inférieur à 12 minutes</i>
CtF2.3.1	<i>Prendre en charge un patient en moins de 12 minutes</i>
Étant donné	<i>La déclaration d'un cas d'urgence</i>
Quand	<i>Un intervenant médical d'urgence prend en charge le patient</i>
<b>Résultat attendu</b>	<i>La prise en charge du patient doit être réalisée en moins de 12 minutes</i>

CF3 – Gérald ATTARD	
<b>SF3.1</b>	<i>Un temps de réponse est calculé</i>
<b>CtF3.1.1</b>	<i>Calculer le temps de réponse</i>
<b>Étant donné</b>	<i>Un incident se produit</i>
<b>Quand</b>	<i>L'incident est déclaré en tant que cas d'urgence</i>
<b>Ensuite</b>	<i>Le délai de réponse à la déclaration est stocké</i>
<b>Résultat attendu</b>	<i>Un temps de réponse est calculé</i>

CF3 – Gérald ATTARD	
<b>SF3.1</b>	<i>La charge de travail est calculée</i>
<b>CtF3.1.2</b>	<i>Calculer la charge de travail</i>
<b>Étant donné</b>	<i>Tous les cas d'urgence déclarés</i>
<b>Quand</b>	<i>Un nouveau d'urgence est déclaré</i>
<b>Ensuite</b>	<i>Le nombre de cas d'urgence total est incrémenté de 1</i>
<b>Résultat attendu</b>	<i>La charge de travail à un instant T est calculée</i>

CF3 – Gérald ATTARD	
<b>SF3.2</b>	<i>Le temps de réponse est inférieur à 200ms</i>
<b>CtF3.2.1</b>	<i>Fournir un temps de réponse inférieur à 200ms</i>
<b>Étant donné</b>	<i>Un incident se produit</i>
<b>Quand</b>	<i>L'incident est déclaré en tant que cas d'urgence</i>
<b>Ensuite</b>	<i>Le temps de réponse à la déclaration du cas d'urgence est stockée</i>
<b>Résultat attendu</b>	<i>Le temps de réponse à la déclaration du cas d'urgence doit être inférieur à 200ms</i>

CF3 – Gérald ATTARD	
<b>SF3.3</b>	<i>Chaque instance de service peut traiter au moins 800 requêtes par seconde</i>
<b>CtF3.3.1</b>	<i>Traiter au moins 800requêtes par seconde par microservice</i>
<b>Étant donné</b>	<i>Une charge de travail de 799 cas d'urgence déclarés</i>
<b>Quand</b>	<i>Un nouveau cas d'urgence est déclaré</i>
<b>Ensuite</b>	<i>Le nouveau cas d'urgence est traité nominalement</i>
<b>Résultat attendu</b>	<i>Le système continue de travailler de façon nominale</i>

CF3 – Gérald ATTARD	
<b>SF3.4</b>	<i>Le temps de réponse est inférieur à 200ms ET une instance de n'importe quel service peut traiter au moins 800 requêtes par seconde</i>
<b>CtF3.4.1</b>	<i>Fournir un temps de réponse inférieur à 200ms ET Traiter au moins 800requêtes par seconde par microservice</i>
<b>Étant donné</b>	<i>CtF3.2.1</i>
<b>Quand</b>	<i>CtF3.3.1</i>
<b>Résultat attendu</b>	<i>La temps de réponse à la déclaration d'un cas d'urgence doit être inférieur à 200ms, même si la charge de travail est de 800 requêtes par seconde</i>

CF4 – Gérald ATTARD	
<b>SF4.1</b>	<i>Les normes sont respectées</i>
<b>CtF4.1.1</b>	<i>Respecter les normes en vigueur</i>
<b>Étant donné</b>	<i>Toutes les normes en vigueur relatives à la prise en charge d'un cas d'urgence</i>
<b>Quand</b>	<i>Une nouvelle norme apparaît</i>
<b>Résultat attendu</b>	<i>Toutes les pratiques médicales respectent toutes les normes, peu importe le domaine</i>

CF4 – Gérald ATTARD	
<b>SF4.2</b>	<i>Tous les microservices respectent les normes</i>
<b>CtF4.2.1</b>	<i>Respecter les normes en vigueur par tous les microservices</i>
<b>Étant donné</b>	<i>Toutes les normes en vigueur relatives à la prise en charge d'un cas d'urgence</i>
<b>Quand</b>	<i>Une nouvelle norme apparaît</i>
<b>Résultat attendu</b>	<i>Tous les microservices respectent toutes les normes déclarées, peu importe le domaine</i>

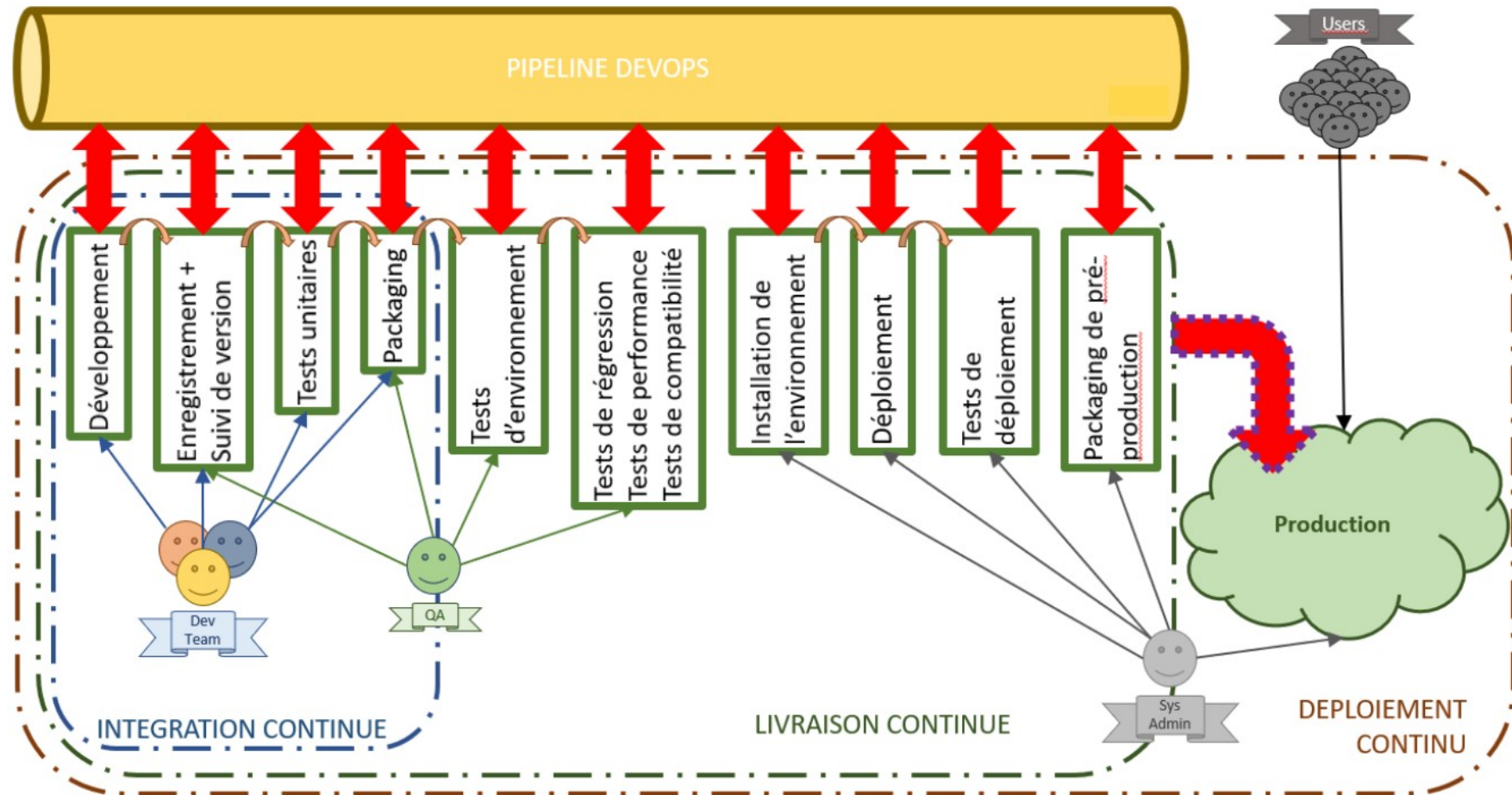
CF5 – Gérald ATTARD	
<b>SF5.1</b>	<i>Les instructions pour mettre en production la PoC sont rédigées</i>
<b>CtF5.1.1</b>	<i>Rédiger la documentation technique relative à la PoC</i>
<b>Étant donné</b>	<i>La réalisation et la mise en œuvre de la PoC</i>
<b>Quand</b>	<i>Une nouvelle fonctionnalité est ajoutée à la PoC</i>
<b>Ensuite</b>	<i>La nouvelle fonctionnalité est ajoutée à la documentation fonctionnelle et technique de la PoC</i>
<b>Résultat attendu</b>	<i>La PoC est documentée</i>

CF6 – Gérald ATTARD	
<b>SF6.1</b>	<i>La mise en œuvre de la PoC est planifiée</i>
<b>CtF6.1.1</b>	<i>Planifier la mise en œuvre de la PoC</i>
<b>Étant donné</b>	<i>La liste de toutes les fonctionnalités devant être implémentées au sein de la PoC</i>
<b>Quand</b>	<i>Chaque fonctionnalité est évaluée</i>
<b>Résultat attendu</b>	<i>La planification de mise en œuvre de la PoC est fournie</i>

CF6 – Gérald ATTARD	
SF6.2	<i>La mise en œuvre de la PoC a respecté la planification</i>
CtF6.2.1	<i>Respecter la planification de mise en œuvre de la PoC</i>
Étant donné	CtF6.1.1
Quand	<i>La mise en œuvre de la PoC est achevée</i>
Ensuite	<i>Une comparaison entre la planification anticipée et la date réelle de mise en œuvre est réalisée</i>
<b>Résultat attendu</b>	<i>La planification de mise en œuvre de la PoC fournie est atteinte</i>

## VIII. Infrastructure préconisée

En se basant sur l'architecture MSA ,préconisée au sein du SAW, la méthode de développement préconisée sera une méthode AGILE, dont les différentes phases sont présentées ci-dessous :



Au sein du diagramme précédent, les campagnes de tests seront réalisées lors des processus de :

- Intégration Continue, par l'équipe de développement (*Dev Team*) ;
- Livraison Continue par les responsables de l'assurance Qualité (QA).

Lors de ces deux précédentes phases, les responsables de tests le réaliseront à des niveaux différents.

En effet, l'équipe de développement réalisera plutôt des tests unitaires, c'est à dire des tests leur permettant de vérifier qu'un extrait de code fonctionne correctement. L'équipe pourra alors utiliser une méthode de type TDD.

Alors que l'équipe d'assurance Qualité se cantonnera davantage vers des tests fonctionnels, c'est à dire des tests servant à s'assurer que toutes les fonctionnalités développées répondent bien au(x) besoin(s). Ces tests visent à démontrer que le logiciel effectue ce pourquoi il a été développé, tout en recherchant d'éventuelles défaillances lorsque le testeur le sollicite sur des valeurs nominales, aux seuils ou hors domaine de définition. Ainsi, peu importe le contexte ou le paramétrage utilisateur demandé, le logiciel doit rester cohérente et stable, tout en renvoyant des données licites autorisées par les développeurs eux-mêmes.

Dans le contexte de tests fonctionnels, ces derniers peuvent être de deux types :

test fonctionnel partiel : dans le contexte de cette étude, en utilisant des méthodologies Agile, c'est ce genre de tests qui seront rencontrés principalement, puisqu'ils seront associées à des livraisons partielles modulaires.

Test fonctionnel total : ce genre de tests n'interviendront alors que lors de la livraison TOTALE du produit, peu importe la méthodologie concernée (Agile ou cycle en V).



## VIII.A. Le pipeline DevOps

Le premier élément à décrire dans un pipeline DevOps est le pipeline lui-même.

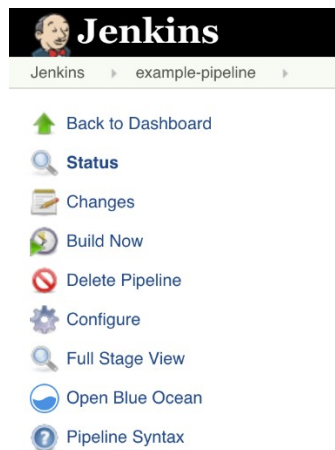
Un pipeline DevOps est un ensemble de pratiques pour lesquelles les équipes de développeurs (Dev) et les équipes d'opérationnels (Ops) concrétisent, testent, et déploient un produit logiciel plus vite et plus facilement que s'ils devaient le faire manuellement, car tout l'intérêt du pipeline réside dans l'automatisation de ces tâches : les développeurs peuvent alors consacrer plus de temps à développer.

Une des premières caractéristiques d'un pipeline est de conserver le process de développement logiciel organisé et focalisé. Ainsi un pipeline DevOps va séquentiellement réaliser les étapes suivantes :

- Planifier ;
- Développer ;
- Compiler ;
- Tester ;
- Déployer ;
- Monitorer (action technique de surveillance d'un sujet).

En termes d'outillage, le choix préconisé s'orientera vers un *Pipeline as Code with Jenkins*, plus communément appelé simplement *Pipeline*.

En effet, *Jenkins Pipeline* est une suite de plugins supportant l'implémentation de l'intégration continue et de la livraison continue au sein d'un pipeline standard déclaré dans une instance standard de *Jenkins*. *Pipeline* fournit ainsi un ensemble extensible d'outils pour la modélisation « *Simple-To-Complex* » d'une pipeline de livraison « *as code* » via le plugin *Pipeline DSL (Domain Specific Language)*.



En termes de prérequis, l'utilisation de *Pipeline* nécessite :

- Une instance de Jenkins 2.x ou supérieur ;
- L'installation du plugin Pipeline (plugin suggéré par défaut à l'installation de Jenkins).

## **VIII.B. L'intégration continue**

### **VIII.B.1. Développement**

#### ***VIII.B.1.a. Phase d'analyse et de conception***

La phase d'analyse est une étape essentielle dans la conception d'un logiciel et un prérequis au développement.

Néanmoins, il est vrai qu'elle n'apparaît pas dans la plupart des diagrammes d'activités techniques car elle est considérée comme une partie réalisée en amont du développement, voire pendant.

#### ***VIII.B.1.b. Méthode d'analyse et de conception***

En ce qui concerne la méthode d'analyse, l'*UML* sera à préconisé pour le projet de *MedHead*.

En effet, UML est particulièrement utilisé et adapté en Génie Logiciel et en conception orientée objet. Son principal intérêt est de modéliser GRAPHIQUEMENT les différents éléments constituant un système, selon 3 types de diagramme :

- des diagrammes de structure,
- des diagrammes de comportement,
- des diagrammes d'interaction.

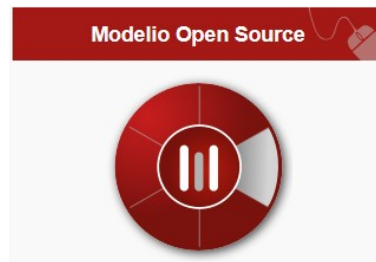
Les principaux diagrammes mentionnés précédemment sont détaillés dans le tableau ci-dessous.

Catégorie	Type de diagramme	Description
Diagramme de structure	Diagramme de classes	Représente les classes
	Diagramme d'objet	Affiche l'état d'un système à un moment donné
	Diagramme des composants	Affiche les dépendances et les composants de structure
	Diagramme de structure composite	Divise les modules ou les classes en leurs composantes et clarifie leurs relations.
	Diagramme de paquetage	Regroupe les classes en paquets, représente la hiérarchie et la structure des paquets
	Diagramme de déploiement	Affiche la distribution des composants aux nœuds de l'ordinateur
	Diagramme de profil	Illustre les relations d'usage au moyen de stéréotypes, de conditions aux limites, etc.
Diagrammes de comportement	Diagramme de cas d'utilisation	Représente divers usages
	Diagramme d'activité	Décrit le comportement de différents processus (parallèles) dans un système.
	Diagramme d'état-transition	Documente la façon dont un objet passe d'un état à un autre par le biais d'un événement.
Diagrammes d'interaction	Diagramme de séquence	Représente le moment des interactions entre les objets.
	Diagramme de communication	Affiche la répartition des rôles des objets au sein d'une interaction.
	Diagramme de temps	Démontre la limitation temporelle pour les événements qui mènent à un changement d'état.
	Diagramme d'aperçu d'interaction	Montre comment les séquences et les activités interagissent.

### VIII.B.1.b.i. Outil d'analyse

*Modelio* est un outil de modélisation UML, intégrant également la modélisation du BPMN. Il propose ainsi un ensemble d'outils supportant la démarche MDA et est compatible avec la norme UML 2.0.

Modelio permet ainsi de générer tous les diagrammes UML décrits dans le paragraphe précédent et permet d'étendre ses fonctionnalités « de base » en ajoutant des extensions de modélisation telles que le BPMN, l'architecture entreprise, l'analyse des objectifs, la définition des dictionnaires, le SOA, le SysML...



### VIII.B.1.c. Outil de développement

L'environnement de Développement sera à choisir en fonction de la nature du projet et du type de technologie appréhendée.

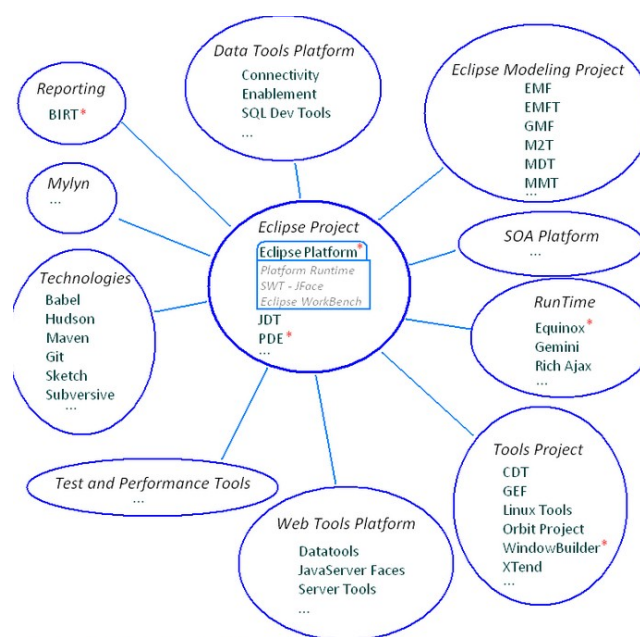
Dans le cas d'un projet JAVA, l'environnement préconisée est Eclipse, développé par la Fondation

Eclipse (<https://www.eclipse.org/>)



Eclipse est à la fois un IDE, un framework et une plateforme de développement, selon que l'on considère le projet, ses composants ou la résultante de leur assemblage.

Le diagramme suivant présente une vue technique de ses possibilités :



\* Eclipse Projects relevant to RCP

## VIII.B.2. Suivi de version

Un logiciel de gestion de version est composé d'une arborescence de fichiers permettant de conserver toutes les versions des fichiers, ainsi que leur historique de différence entre chaque mise à jour (*check out*).

Cette notion de suivi de version s'accompagne intrinsèquement par la notion de branche de version ou branche de développement.

Avant d'aborder le type de branchement, préconisé au sein des *MedHead* parmi tous les types de branchement existant, il est nécessaire de décrire l'objectif de la société : **Avoir le minimum de branches possibles.**

A ce stade, il est normal de se poser la question quant à la légitimité d'un tel axiome. Après tout, avoir beaucoup de branches différentes « *devrait être* » synonyme de flexibilité ou de persistance. Par expérience, il n'en est pas du tout ainsi.

En effet, le fait d'avoir beaucoup de branches engendrent beaucoup de gaspillage de temps et de ressources, et c'est précisément cela que le DevOps s'efforce de minimiser, voire de supprimer.

Aussi, afin de réduire le champ de divergence entre branches et améliorer la Qualité ainsi que la productivité de reprise de code, cette étude se bornera à avoir le minimum de branche de version possible...autant que faire se peut et en fonction des briques fonctionnelles développées.

### VIII.B.2.a. Les branches de développement

#### VIII.B.2.a.i. Un mouvement en deux temps

Le mouvement général du développement à plusieurs équipiers, ou à plusieurs équipes, est toujours un mouvement en deux temps :

- un temps de développement à l'abri des modifications simultanées des autres ;
- un temps d'intégration.

Du point de vue du gestionnaire de sources, la création d'une « *branche* » dans laquelle on enregistre les versions successives est le procédé qui permet d'isoler les développements de ceux d'autres développeurs, ou de séparer les versions de deux équipes de développement distinctes.

#### VIII.B.2.a.ii. Choisir son tempo

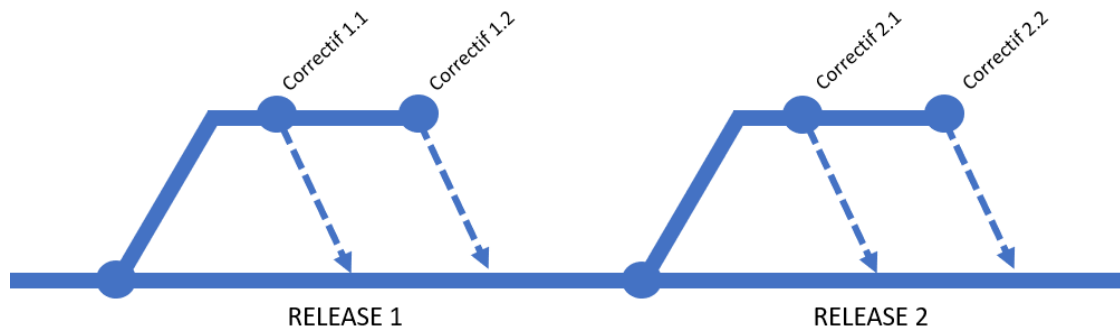
La question restante porte alors sur le rythme, c'est à dire la cadence à donner à ce mouvement de va-et-vient : faut-il produire beaucoup, se protéger longtemps et fusionner au plus tard, ou au contraire consolider souvent ses travaux en tant qu'équipier et ses développements en tant qu'équipe ?

Du point de vue de *MedHead*, cette étude privilégiera la consolidation périodique et rapprochée afin d'obtenir une itération de codage de l'écart la plus brève possible, la fusion la plus fréquente et la pulsation la plus rapide possible.

De toutes les options de branchement existantes et en suivant l'axiome de Qualité que *MedHead* s'impose au travers de sa politique d'entreprise, énoncée dans le cahier des charges, cette étude privilégiera ainsi le branchement par *release* adapté au contexte industriel.

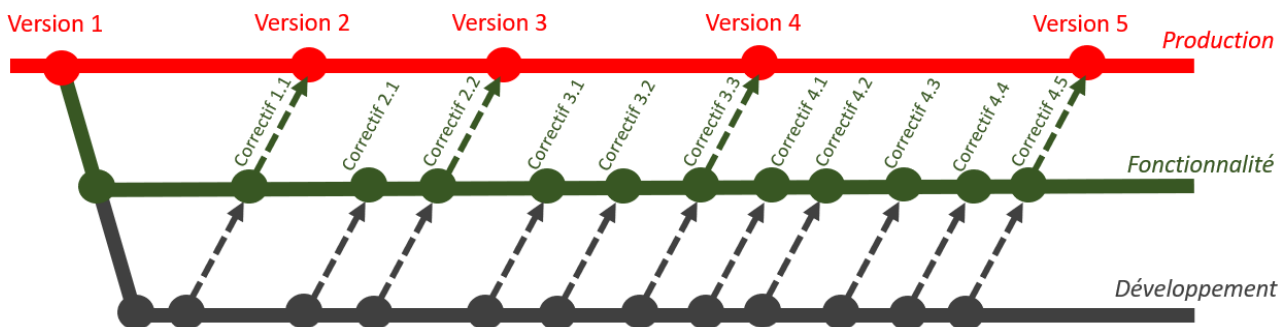
### VIII.B.2.b. Le branchement par *release*

Par définition, l'intégration continue se dispense de branches sauf lors des mises en production. Ainsi, une branche de version d'Intégration Continue « *pure* » devrait ressembler au schéma suivant :



Néanmoins, dans un contexte industriel tel que celui de *MedHead*, cela reste utopique ; il n'en est pas moins vrai, qu'en terme d'Amélioration Continue, cette étude adoptera un modèle y tendant, voire y ressemblant fortement.

Aussi, la proposition ci-dessous se veut être une combinaison de l'axiome décrit précédemment (« avoir le minimum de branche possible ») tout en prenant en compte l'aspect opérationnel sur lequel l'appliquer, c'est un consensus des deux propositions :




Ainsi, le consensus obtenu dans l'exemple ci-dessus peut se décomposer de 3 branches :

- production ou pré-production selon le contexte du projet ;
- fonctionnalité appelée couramment *release* ;
- développement - cette branche est la somme des branches locales de chaque développeur de ou des équipe(s).

En outre, chaque version est considérée par un numéro Majeur de version et chaque *release* est considérée par un numéro Mineur de version ; en prenant comme base d'exemple le schéma précédant, la version 5 correspond à la *release* 4.5 .

### VIII.B.2.c. Outil

Cette étude n'ayant pas encore les spécifications techniques des outils de développement utilisés chez *MedHead*, elle préconisera l'outil gratuit de gestion de version nommé *Tuleap*  **tuleap** de la société *Enalean SAS*©.

Tuleap est un outil offrant plusieurs fonctionnalités s'intégrant dans le pipeline DevOps, dont notamment une compatibilité avérée avec *Jenkins* présenté dans un paragraphe précédent.

En ce qui concerne le suivi de version, *Tuleap* intègre nativement un module d'intégration de branches *Git*, *GitHuh*, *svn* et *cvs* en son sein.

Ainsi, quand un « *push* » (*check out*) est effectué dans un dépôt *Tuleap-git*, la tâche liée au *Jenkins* sera automatiquement réalisée.

En terme de dépôts, *Tuleap-git* en proposent deux types :

- les projets de référence : branche par défaut, se focalisant sur l'enregistrement et le suivi des versions officielles des dépôts de chaque projet ;
- les projets clone (*fork*) : chaque membre du projet peut cloner la branche référence afin de la publier dans son espace de travail personnel. Ceci permet de publier et d'intégrer le développement de modèle personnel tout en protégeant le branche référence du projet. Le *check-out* sera alors réalisé quand le développement personnel sera mature et aura passé tous les tests unitaires avec succès.

Pour plus de détails, les sources de cette étude sont issues de <https://plugins.jenkins.io/tuleap-git-branch-source/>



### VIII.B.3. Package

Le terme *Package* est utilisé ici en termes d'archive et non pas en termes programmatique d'organisation ou de structuration du code lui-même.

Ainsi, lorsque le produit logiciel sera suffisamment mature et possèdera les fonctionnalités minimales pour en effectuer une première version, un *package* sera créé afin d'obtenir une archive de l'incrément réalisé.

Concrètement, en prenant comme base un projet à dominante Java, et suite à l'écriture du code constituant le produit logiciel lui-même, le principal outil sera naturellement le binaire « *jar* » inclus au sein du *JDK* (*Java Development Kit*).

```
jar
Usage: jar [OPTION...] [ [--release VERSION] [-C dir] files] ...
Try `jar --help' for more information.

jar --help
Usage: jar [OPTION...] [ [--release VERSION] [-C dir] files] ...
jar creates an archive for classes and resources, and can manipulate or
restore individual classes or resources from an archive.

Examples:
# Create an archive called classes.jar with two class files:
jar --create --file classes.jar Foo.class Bar.class
# Create an archive using an existing manifest, with all the files in foo/:
jar --create --file classes.jar --manifest mymanifest -C foo/ .
# Create a modular jar archive, where the module descriptor is located in
# classes/module-info.class:
jar --create --file foo.jar --main-class com.foo.Main --module-version 1.0
  -C foo/ classes resources
# Update an existing non-modular jar to a modular jar:
jar --update --file foo.jar --main-class com.foo.Main --module-version 1.0
  -C foo/ module-info.class
# Create a multi-release jar, placing some files in the META-INF/versions/9 directory:
jar --create --file mr.jar -C foo classes --release 9 -C foo9 classes
```





## **IX. Hypothèses et risques**

### **IX.A. Hypothèse en Ressources Humaines**

Tel qu'il a été présenté dans le SAW et le présent plan de tests, l'architecture préconisée se basera sur la MSA.

Aussi, pour mettre en place cette architecture techniquement, MedHead devra constituer une équipe de développement, de préférence AGILE pour la concrétiser et la mettre en œuvre.

En fonction des ressources mises à disposition, l'équipe de QA chargée des tests fonctionnels, aussi bien d'intégration que de livraison, devrait à minima être constituée de trois ETP.

### **IX.B. Hypothèse temporelle**

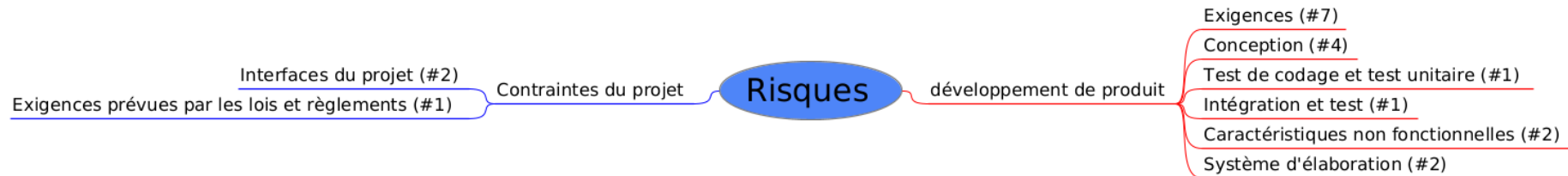
En ce qui concerne, la durée des campagnes de tests définis, celles-ci seront à évaluer en fonction du nombre de cas de tests dénombrés au sein de chaque scénario de la campagne.

Pour prévoir une estimation moyenne d'une campagne de tests, il serait souhaitable de partir sur une durée de trois semaines par campagne (et donc par fonctionnalité testée) pour un effectif de trois ETP.

## IX.C. Risques

Tel qu'il a été annoncé dans le SAW, dans le cadre de cette étude, seule la sécurité relative aux données, aux applications et à la technologie utilisée, et donc aux informations, seront abordées ; le contexte d'entreprise devra être assumés par d'autres autorités compétentes.

Ainsi, les risques dont il est question ici seront liés au déroulement du projet de réalisation d'une preuve de concept. Ces risques peuvent être caractérisés selon deux domaines spécifiques : les contraintes liées au projet et le développement de la solution applicative, telles que présentées dans la carte heuristique ci-dessous :



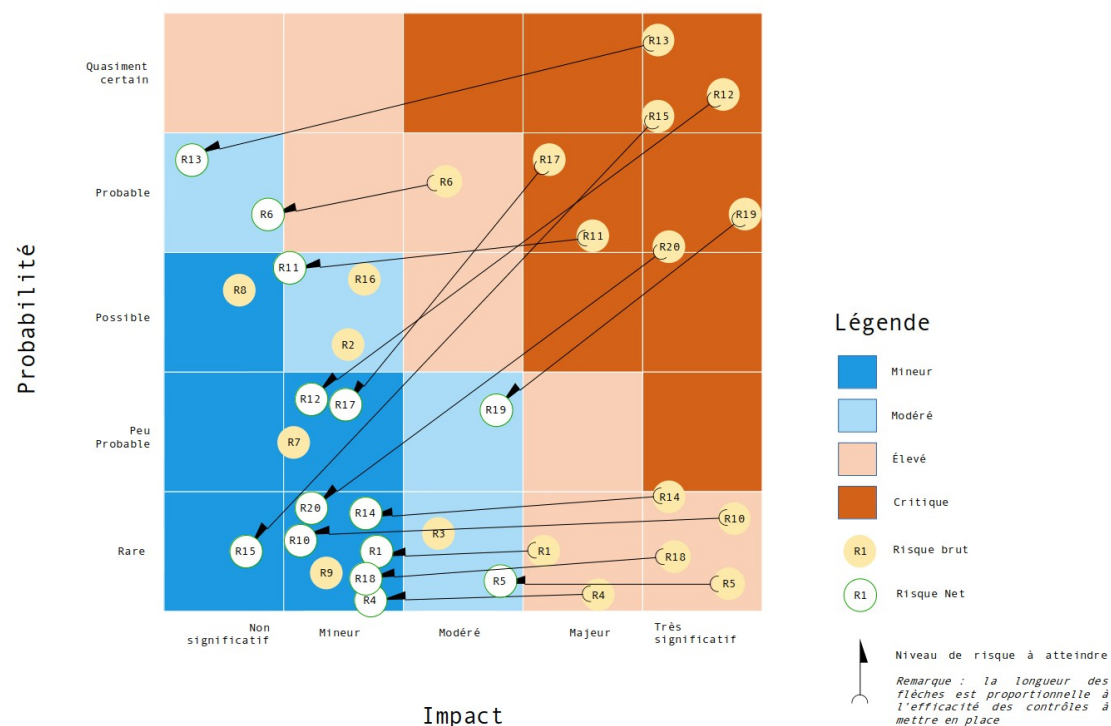
A partir de cette carte heuristique, il est alors possible d'en extrapoler une cartographie des risques découlant des domaines et sous-domaines de risques identifiés :

Id.	Risque	Description du risque	Type de risque	Domaine du risque	Impact	Probabilité	Action	Responsabilité	Date de l'examen	Efficacité des contrôles
R1	Stabilité	Stabilité du système précaire	Exigences	Développement de produit	Majeur	Rare (<10%)	S'assure de la qualité et de la stabilité des matériels et logiciels choisis	TBD		
R2	Exhaustivité	Choix réduit des services proposés par l'extranet	Exigences	Développement de produit	Mineur	Possible (30-50%)	Le système doit proposer les mêmes ressources que le précédent	Gérald ATTARD		
R3	Clarté	Manque de clarté des services offerts	Exigences	Développement de produit	Modéré	Rare (<10%)	Le nouveau système doit être tout aussi convivial que l'ancien	Gérald ATTARD		
R4	Validité	Informations affichées non valides ou périmées	Exigences	Développement de produit	Majeur	Rare (<10%)	Les informations affichées doivent être actuelles, exactes et valides	TBD		

Id.		Risque	Description du risque	Type de risque	Domaine du risque	Impact	Probabilité	Action	Responsabilité	Date de l'examen	Efficacité des contrôles
R5	Faisabilité	Services proposés dépassent le budget	Exigences	Développement de produit	Très significatif	Rare (<10%)	Estimer l'adéquation de la réalisation des services relativement au budget à disposition		TBD		
R6	Unicité	Informations dupliquées et/ou contradictoires	Exigences	Développement de produit	Modéré	Probable (50-90%)	S'assurer que les informations sur l'extranet sont issues d'une source uniques		Gérald ATTARD		
R7	Ampleur	Ressources technologiques sous-évaluées	Exigences	Développement de produit	Mineur	Peu probable (10-30%)	Les ressources techniques doivent être au moins dimensionnées pour accueillir les partenaires, les fournisseurs et les clients		Gérald ATTARD		
R8	Fonctionnalité	Fonctionnalités inadaptées	Conception	Développement de produit	Non significatif	Possible (30-50%)	Les fonctionnalités devront être systématiquement validées par le client sur site de l'équipe AGILE		Gérald ATTARD		
R9	Difficulté	Difficulté de conception et/ou de réalisation rendant le projet irréalisable	Conception	Développement de produit	Mineur	Rare (<10%)	S'assurer que les besoins exprimées par <i>MedHead</i> sont fonctionnellement et techniquement réalisables		Gérald ATTARD		
R10	Interfaces	IHM confuses	Conception	Développement de produit	Très significatif	Rare (<10%)	Les IHM de l'extranet devront fournir les mêmes services que l'ancien SI		Gérald ATTARD		
R11	Contraintes informatiques	Contraintes techniques relatives à l'existant non prises en compte	Conception	Développement de produit	Majeur	Probable (50-90%)	L'extranet devra être compatibles avec les technologies employées par le SI existant		Gérald ATTARD		
R12	Mise à l'essai	Les fonctionnalités ne sont pas testées	Test de codage et test unitaire	Développement de produit	Très significatif	Quasiment certain (>90%)	Mettre en place une politique de TDD pour assurer la qualité du code		TBD		
R13	Environnement	Non prise en compte de l'environnement technique	Intégration et test	Développement de produit	Très significatif	Quasiment certain (>90%)	Analyser exhaustivement les structures matérielles et logiques lors de l'étude de l'existant		Gérald ATTARD		
R14	Fiabilité	Informations affichées non pertinentes ou erronées	Caractéristiques non fonctionnelles	Développement de produit	Très significatif	Peu probable (10-30%)	S'assurer de la pertinence des informations affichées pour fournir des services de qualité		Gérald ATTARD		
R15	Sécurité	Permissivité de connexion	Caractéristiques non fonctionnelles	Développement de produit	Très significatif	Quasiment certain (>90%)	S'assurer de l'activation des comptes et de l'intégrité des utilisateurs accédant à l'extranet		Gérald ATTARD		
R16	Connaissance	Les besoins métiers ne sont pas répondus	Système d'élaboration	Développement de produit	Mineur	Possible (30-50%)	Les services rendus par l'extranet doivent être au moins équivalents à ceux traduit par le SI existant		TBD		
R17	Convivialité	Services de l'extranet brouillon	Système d'élaboration	Développement de produit	Majeur	Probable (50-90%)	Les services de l'extranet doivent pouvoir être accessibles avec le moins de clic possible		Gérald ATTARD		
R18	Entrepreneurs délégués	Représentant de compte non identifiés	Interfaces du projet	Contraintes du projet	Très significatif	Rare (<10%)	Identifier les représentants de compte pour les fournisseurs, les partenaires et les clients		TBD		

Id.	Risque	Description du risque	Type de risque	Domaine du risque	Impact	Probabilité	Action	Responsabilité	Date de l'examen	Efficacité des contrôles
R19	Entrepreneur principaux	Fournisseurs, Partenaires et Client non identifiés	Interfaces du projet	Contraintes du projet	Très significatif	Probable (50-90%)	Identifier les représentants de compte pour les fournisseurs, les partenaires et les clients	TBD		
R20	Respect de la vie privée	Informations saisies accessibles publiquement	Exigences prévues par les lois et règlements	Contraintes du projet	Très significatif	Probable (50-90%)	Les informations relatives aux entreprises et aux différents intervenants doivent être sécurisées autant pendant les phases de transaction que celle de stockage.	Gérald ATTARD		

### IX.C.1. Cartographie des risques



La cartographie présentée ci-contre fait référence à la matrice des risques établies au sein du paragraphe précédent. Relativement au domaine sanitaire couvert par ce projet, dont notamment la notion d'urgence, la solution envisagée sera susceptible d'être victime de risques techniques et métiers qui devront être traités en collaboration des parties prenantes spécialistes du domaine de Santé, ayant des notions d'urgence et de priorité, tels que :

- une non conformité des messages d'urgence émis ;
- une non prise en compte ou la perte des messages d'urgence émis ;
- une mauvaise interprétation des messages d'urgence émis ;
- une interruption de services, dont ceux d'urgence ;
- une pénurie de ressource allouée aux contexte d'urgence ;
- une perte de confiance dans le dispositif d'urgence de la part des patients ;
- un temps de rétablissement de services d'urgence inacceptable suite à leur interruption.

MedHead+