Enoncé des travaux d'architecture

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	05/01/2023	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
ADatP-3	Allied Data Publication v3 (trad. *publication de données alliées version 3*)
AMQP	Advanced Message Queuing Protocol (trad. *protocole avancé de file d'attente de messages*)
API	Application Programming Interface (trad. *interface de programmation d'application*)
APP-11	NATO message catalogue (trad. *catalogue de messages de l'OTAN*)
CRQS	Command and Query Responsibility Segregation (trad. *ségrégation des responsabilités de commande et de requête*)
ERM	Enterprise Risk Management (trad. *gestion des risques d'entreprise*)
ESB	Enterprise Service Bus (trad. *bus de service d'entreprise*)
HTML	HyperText Markup Language (trad. *langage de balisage hypertext*)
IDE	Integrated Development Environment (trad. *environnement de développement intégré*)
IER	Information Exchange Requirement (trad. *exigence d'échange d'information*)
IoT	Internet of Things (trad. *internet des objets*)
ISM	Information Security Management (trad. *gestion de la sécurité de l'information*)
JSON	JavaScript Object Notation (trad. *notation d'objet Javascript*)
JVM	Java Virual Machine (trad. *machine virtuelle Java*)
MSA	MicroServices Architecture (trad. *architecture en microservices*)
MSGID	MeSsaGe Identification (trad. *identifiant de message*)
MTF	Message Text Format (trad. *format de message texte*)
MTFID	Message Text Format Identifier (trad. *identifiant de format de message texte*)
OTAN	Organisation du Traité de l'Atlantique Nord
REST	REpresenational State Transfer (trad. *tranfert d'état représentatif*)
SOAP	Simple Object Access Protocol (trad. *protocole d'accès aux objets simples*)
Trigger	*Évènement déclencheur*
URI	Uniform Resource Identifier (trad. *identifiant uniforme de ressource*)
URL	Uniform Resource Locator (trad. *localisateur uniforme de ressource*)
XML	Extensible Markup Language (trad. *langage de balisage extensible*)

Table des matières

I. Contexte et portée du projet d'architecture

Issue d’un consortium de quatre entrepises majeures dans le domaine de la Santé, l’organisation MedHead a décidé de réaliser un projet de rationalisation de leurs méthodes et moyens ; le consortium en question regroupant les sociétés :

- Ursa Major Health,
- Jupiter Scheduling,
- Emergency Expert Systems,
- Schedule Shed (anciennement Schedule Shack).

Ce regroupement d’acteurs majeurs, dans le domaine de la Santé, a pris la décision de créer une plateforme logicielle unique, ayant pour mission d’améliorer leurs services sanitaires, en fournissant une réponse adaptée et appropriée à la prise en charge, en temps réel, des patients, notamment en situations d’urgence.

Ainsi, cette plateforme devra permettre de :

- coordonner les flux de travail entre différents systèmes hétérogènes, notamment en spécifiant l’intégration de leurs données ;
- fournir une intégration qualitative de ces données pour améliorer les prises de décisions médicales ;
- gérer de manière cohérente les ressources en fonction de plusieurs critères, tels que leur spécialité, leur disponibilité, leur localisation…

L’objectif de cette plateforme sera donc de combiner les différentes forces des acteurs du Consortium, et de pallier collectivement aux faiblesses de chacun. Plus concrètement, le partage des informations d’un patient, du savoir-faire de chaque spécialité indépendamment de son entité d’appartenance, ou encore des moyens à disposition, sont autant d’axes d’amélioration quant à la prise en charge de ces mêmes patients.

La coopération et la collaboration entre chaque entité, sous-entité, domaine de spécialité ou encore moyen à disposition, seront ici les bases sur lesquelles cette nouvelle plateforme travaillera pour mettre en concert, méthodiquement et systèmiquement, tous ces entrants, de façon coordonnée et rationnalisée.

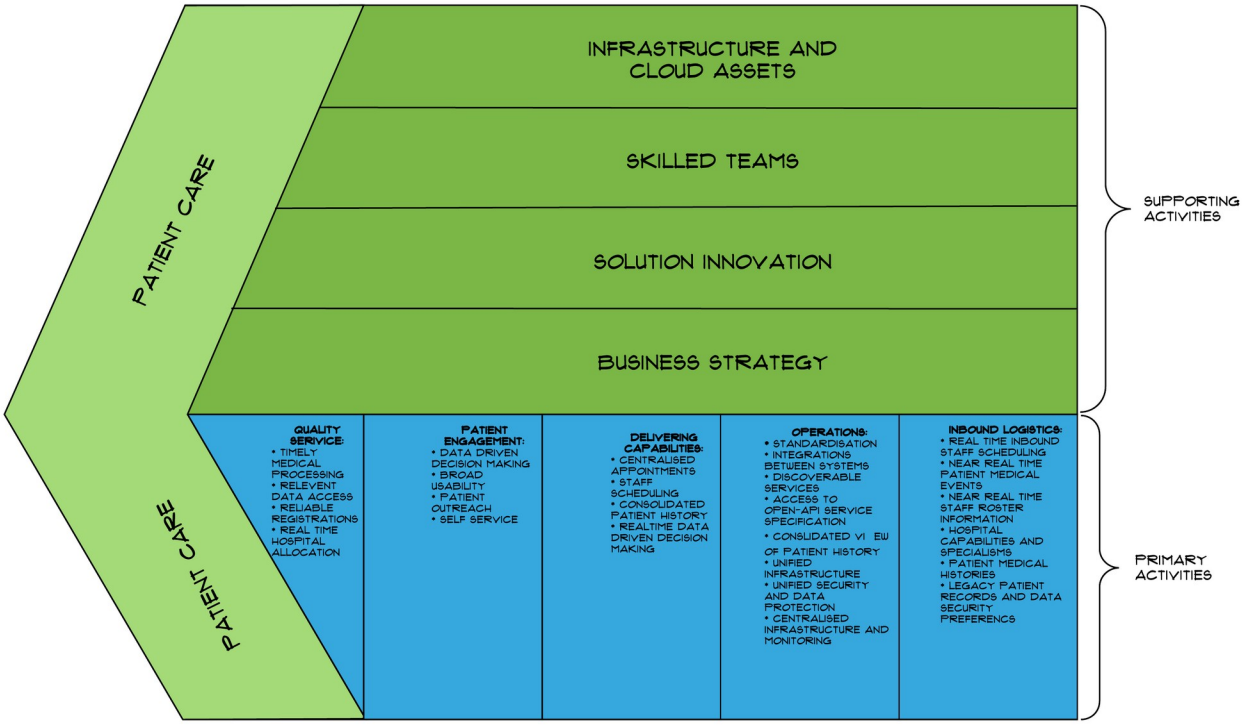
Enfin, l’objectif final de ce projet en particulier sera de développer une preuve de concept, spécifique au sous-système d’intervention d’urgence en temps réel.

En outre, tel qu’il est précisé dans le document de définition d’architecture, l’«...a*rchitecture de systèmes relative aux solutions héritées et internes qui résident en dehors des domaines problématiques exprimés dans le contexte métier et les objectifs du projet...* » seront hors de portée de l’ensemble dudit projet.

II. Vue d'ensemble de la vision de l'architecture

II.A. Chaîne de valeurs des capacités métiers

Tel qu'il est décrit dans le document de définition d'architecture, la cible métier de la chaîne de valeurs liée aux capacités de la nouvelle plateforme doit correspondre au diagramme ci-dessous :



Ainsi, le diagramme ci-dessous comprend trois parties disctinctes :

- une **cible** : les soins délivrés au patient ;
- une **partie d'activités primaires** ou principales comprenant :
  - la qualité des services ayant un lien direct ou indirect avec le patient ;
  - l'engagement pris envers ce patient ;
  - la fourniture de capacités ;
  - le descriptif de l'ensemble des opérations et moyens à la disposition du patient ;
  - toute la logistique à disposition du patient.
- une **partie d'activités de soutien** ou supports comprenant :
  - tous les actifs technologiques nécessaires aux informations du patient ;
  - les matrices de compétences de toutes les équipes qualifiées pour traiter le patient ;
  - les propositions de solutions techniques innovantes en rapport avec la pathologie du patient ;
  - la liste des stratégies métiers pour traiter le patient médicalement.

En outre, chacune des parties décrites sera soumise à des strates de confidentialité pour limiter les accès au besoin d'en connaître, en fonction de la décision finale du patient lui-même.

II.A.1. Définition d’un système d’intervention d’urgence

Ce paragraphe devra être corroboré et validé par un expert métier en intervention d’urgence.

Les systèmes d'intervention d'urgence sont conçus pour fournir une réponse rapide et coordonnée aux urgences telles que les accidents, les incendies, les catastrophes naturelles, ou encore les attaques terroristes.

Dans de nombreux cas, les systèmes d'intervention d'urgence sont organisés et exploités par des agences gouvernementales. Néanmoins, les entreprises et organisations privées peuvent également avoir leurs propres systèmes d'intervention d'urgence.

Un système d'intervention d'urgence comprend généralement quatre composants principaux :

1. un centre de commandement et de contrôle qui coordonne la réponse à une urgence ;
2. les premiers intervenants qui sont dépêchés sur les lieux d'une urgence ;
3. un système de communication qui permet aux premiers intervenants de communiquer entre eux et avec le centre de commandement et de contrôle ;
4. un plan qui décrit comment la réponse à une urgence sera coordonnée.

Ainsi, un système d'intervention d'urgence peut être utilisé pour une variété d'urgences sur de nombreux contextes et théatres divers et variés ; les plus efficaces étant ceux qui sont complets et coordonnés. Ils doivent alors être en mesure d'évaluer rapidement la situation, de mobiliser des ressources et de fournir une assistance de manière coordonnée et efficace.

Les éléments clés d'un système d'intervention d'urgence efficace comprennent :

- un plan clair et concis qui décrit les rôles et les responsabilités de tous les organismes et intervenants concernés;
- un plan de communication complet qui inclut toutes les parties prenantes, telles que les médias, les autorités locales et le public ;
- une infrastructure robuste et fiable, comprenant un centre de commandement et de contrôle efficace, qui peut être rapidement activé en cas d'urgence ;
- des ressources suffisantes, y compris du personnel formé et de l'équipement, pour répondre efficacement à une urgence ;
- un programme efficace de formation et d'exercices qui garantit que tous les organismes et intervenants sont prêts à intervenir en cas d'urgence ;
- un plan de rétablissement solide qui décrit les rôles et les responsabilités de tous les organismes et intervenants impliqués dans le processus de rétablissement.

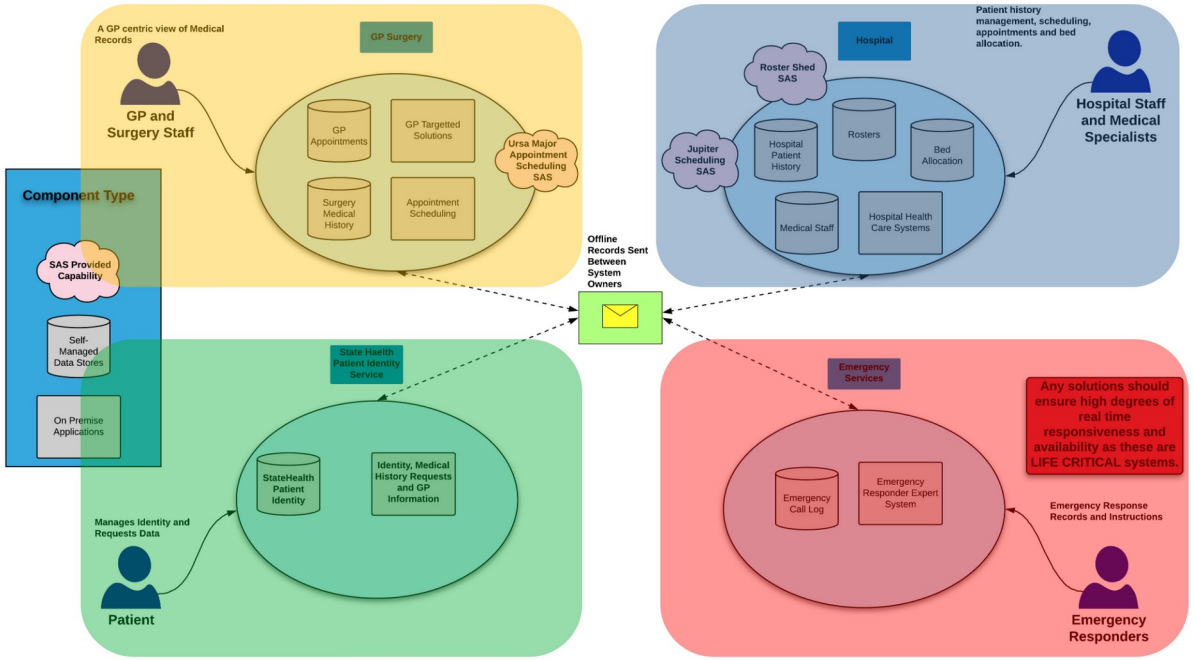
Ainsi, le système d’intervention d’urgence de MedHead devra répondre à tous les critères édictés ci-dessus.

II.B. Architecture de base

Tel qu'il a été présenté dans le document de définition d'architecture, 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'acceuil 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 :



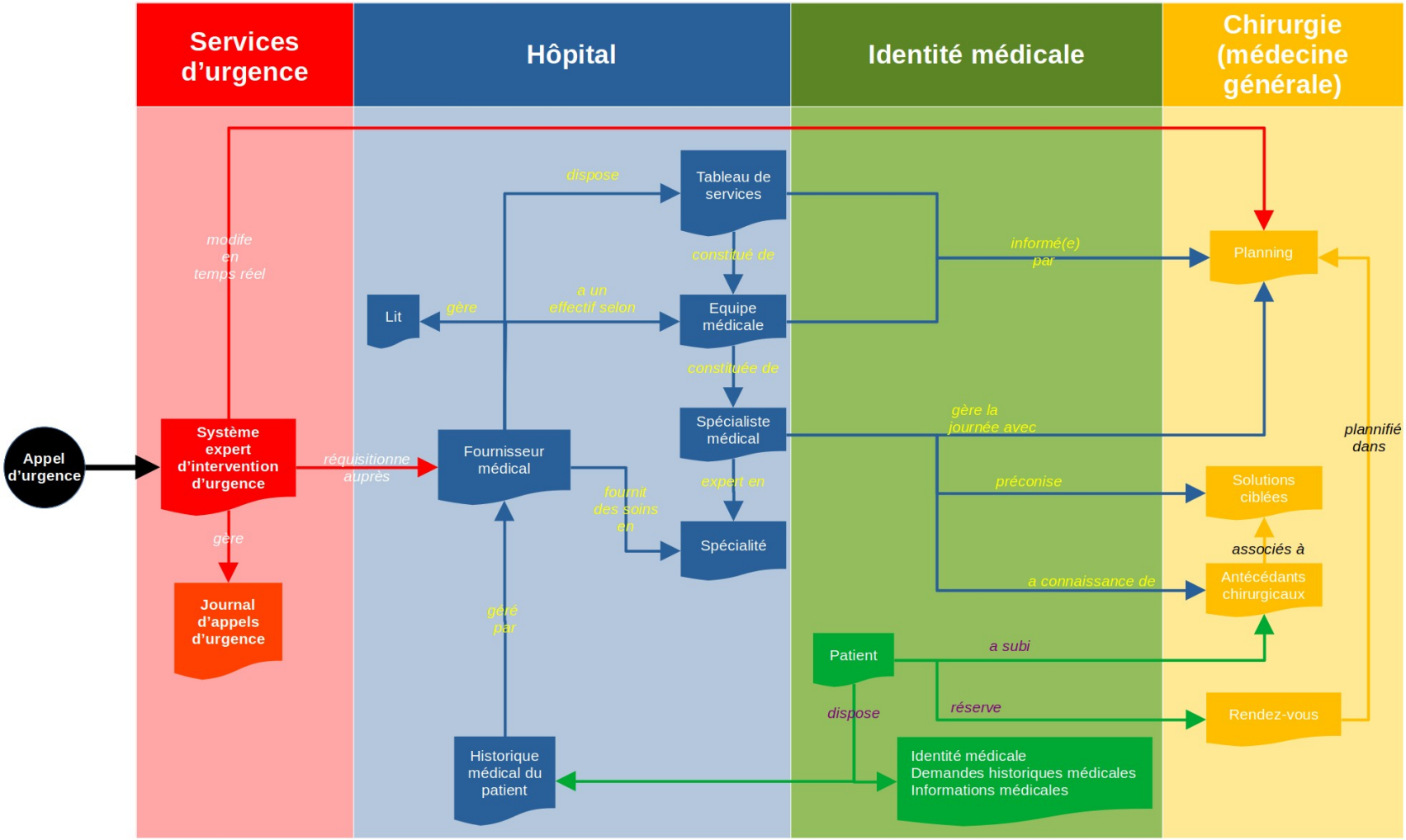
II.C. Architecture de données et de l'information

Relativement aux quatre contextes présentés dans le paragraphe précédent, il est possible de les compléter avec des données, telles que :

Contexte	Données
Chirurgie (médecine générale)	• Planification de la gestion des informations (programmation médicale à court et long termes des activités quotidiennes et de la disponibilité) • Gestion des rendez-vous • Gestion des antécédents médicaux de chirurgie • Gestion des solutions prévues et ciblées
Hôpital	• Gestion des prestataires médicaux (fournisseur hospitalier) • Gestion des spécialités • Gestion des équipes médicales • Gestion des services médicaux • Gestion de l'historique des passages en milieu hospitalier du patient • Gestion des lits disponibles en milieu hospitalier • Systèmes de santé hospitaliers
Intervention d'urgence	• Gestion du registre d'appels d'urgence • Système expert d'intervention d'urgence
Identité médicale	• Informations sur le patient (identité, demande d'antécédents médicaux, informations chirurgicales diverses...) • Informations sur les spécialistes médicaux intervenant

Il est nécessaire de préciser que la réalisation de cette PoC se concentrera sur le domaine d'Intervention d'urgence, présentée au sen de la ligne **ROUGE** du tableau ci-dessus.

Les étapes de communications des données recensées dans le précédent tableau ci-dessus peuvent être représentées graphiquement comme suit :



II.D. Architecture de Sécurité

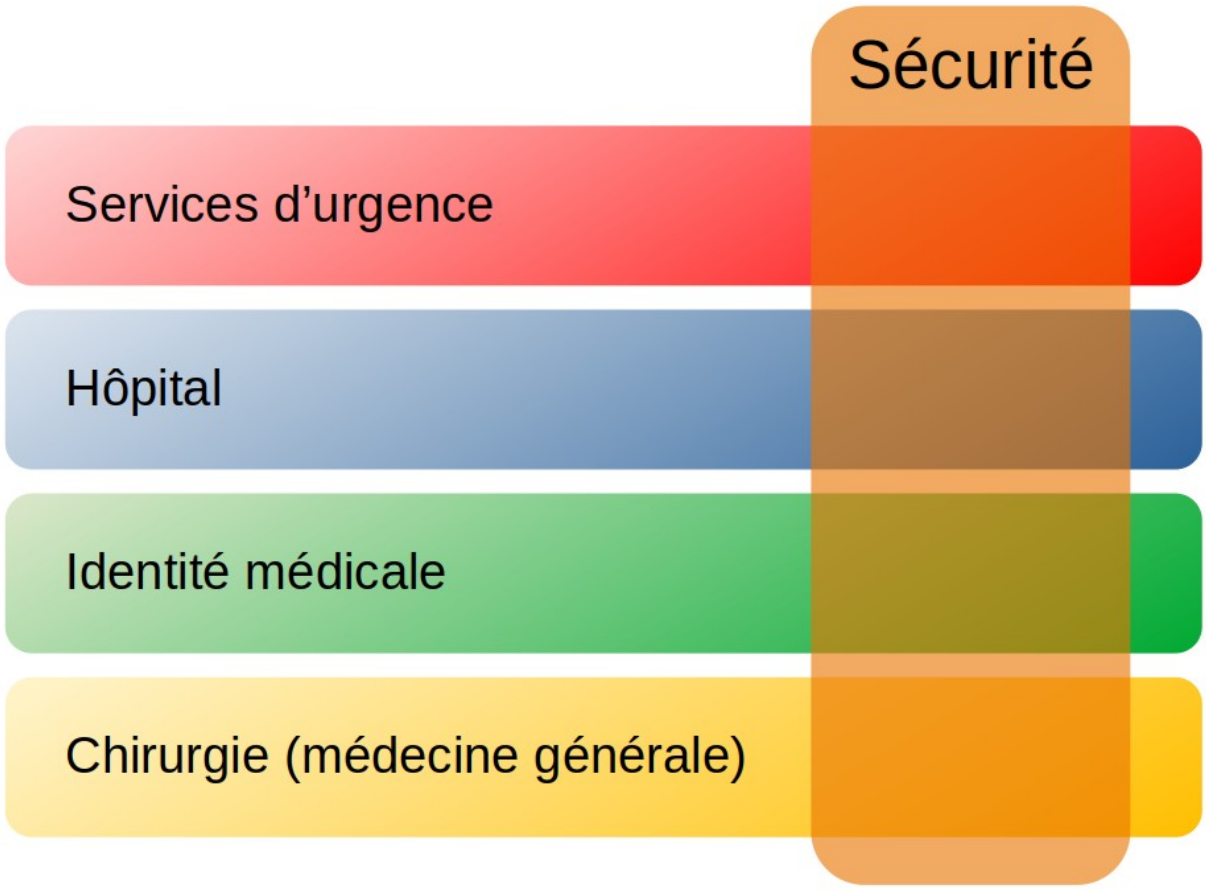
L'architecture de sécurité est une constante à prendre en compte dans ce projet. Elle doit être un sujet d'étude et d'amélioration transversale, omniprésent au sein de l'organisation même du Consortium.

La représentation d'une architecture de sécurité est relative au point sous laquelle un observateur l'appréhende. En effet, celle-ci peut être perçue comme une superposition de calques sur un système en place. Cet ensemble cohérent de vues, ou de points de vue, comprend notamment les perspectives de sécurité, le risque opérationnel et tous les sujets connexes d'objectifs desservant les services de sécurité.

Telle qu'elle a été qualifiée plus haut, cette architecture peut alors être considérée de transversale puisqu'elle interagit avec tous les domaines contextuels : l'entreprise, les données, les applications et la technologie utilisée.

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.

Cette transerversalité de la sécurité, appliquée au contexte hospitalier, peut se représenter graphiquement telle que :



Ainsi, la gestion de la sécurité de l'information, ou ISM (*Information Security Management*), doit garantir au sein des systèmes hospitaliers considérés :

- la **confidentialité** : le fait de s'assurer que l'information n'est accessible qu'à ceux dont l'accès est autorisé ;
- l'**authenticité** : état, prouvant l'identité, obtenu après avoir passé une étape de vérification de l'identité, en fournissant des preuves, telles qu'une carte d'identité ;
- la **non-répudiation** : possibilité de vérifier que l'envoyeur et/ou le destinataire sont bien les parties qui disent avoir respectivement envoyé et/ou reçu l'information ;
- l'**intégrité** : ce critère fait référence à la **fiabilité** et à la **crédibilité** d'une information durant tout son cycle de vie (sa **pérennité**) ;
- la **disponibilité** des systèmes et de leurs données afférentes : cet état est le critère assurant la garantie d'accès à un système, une application ou une donnée.

La notion de **résilience** est volontairement absente de la liste ci-dessus puisqu'il s'agit d'une liste de conséquences, alors que ce critère est de l'ordre de la causalité. Néanmoins, la **résilience** fait référence à la capacité d'un système informatique à continuer de fonctionner en cas de panne, d'incident, d'action malveillante ou d'augmentation des opérations commerciales.

C'est sur les critères listés supra que se base l'ISM, qui est elle-même la base de la gestion des risques logiques.

Cette ISM a pour objectif de garantir une utilisation raisonnable des ressources d'information de l'organisation et une gestion appropriée des risques de sécurité liés à l'information elle-même.

Ainsi, l'ISM peut être considérée à la fois comme un système et aussi comme un processus de traitement, selon le point de vue.

II.D.1. ISM

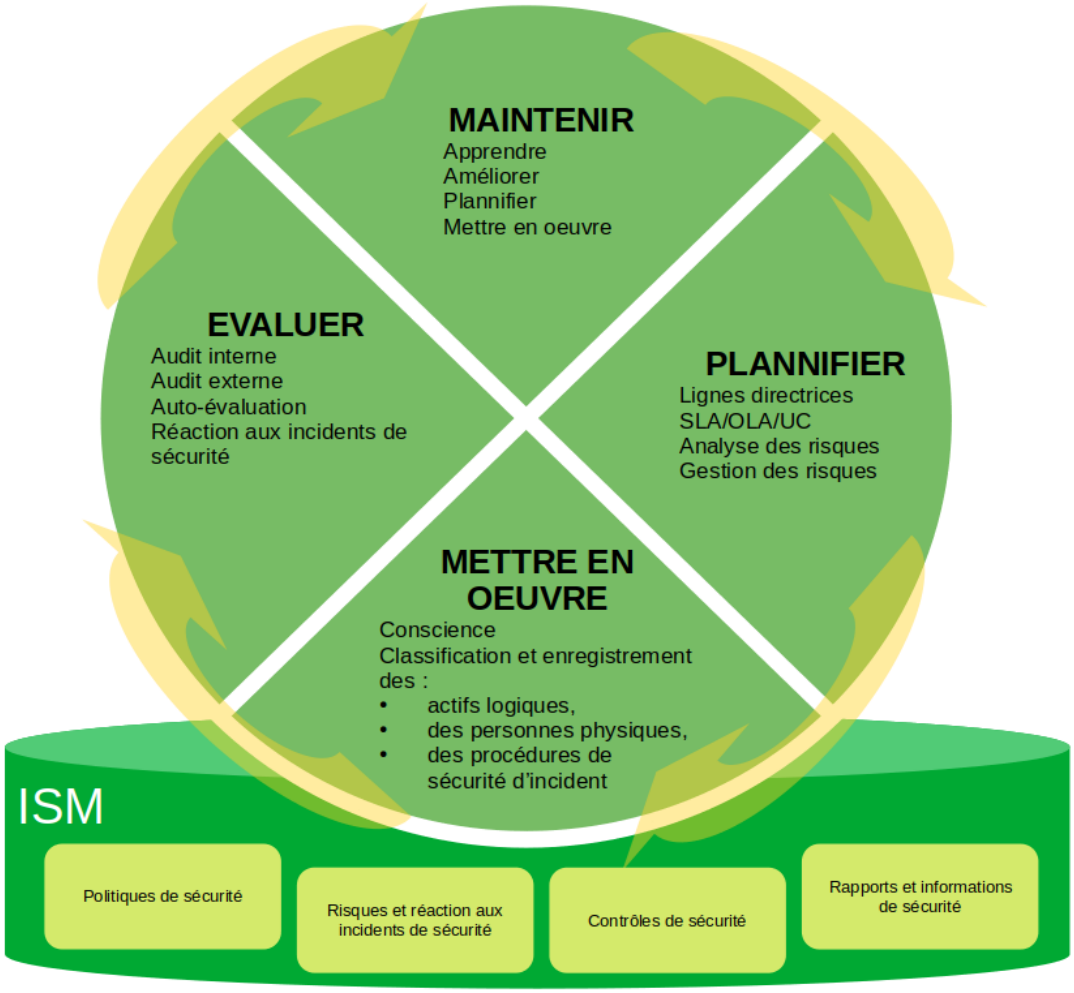
Ainsi, en respectant les critères de sécurité énoncés dans le paragraphe précédent, il est alors possible de considérer que la sécurité de l'information peut être considérée comme respectée lorsque :

- les informations sont observées et/ou divulguées uniquement par les personnes autorisées (**confidentialité** et **authenticité**) ;
- les transactions commerciales ainsi que les échanges d'informations entre organisations et/ou avec des partenaires peuvent être fiables (**authenticité** et **non-répudiation**) ;
- les informations détenues sont fiables et crédibles au moment de leur utilisation (**intégrité**) ;
- les informations sont disponibles (**disponibilité**) et utilisables en cas de besoin ;
- les systèmes récupèrent ou préviennent les pannes (**disponibilité**)
- les systèmes fournissant les informations résistent aux attaques (**résilience**) .

Aussi, vis à vis des assertions précédentes, il est nécessaire que les politiques de sécurité ISM couvrent certains domaines appropriés de la Sécurité :

- une politique d'utilisation, ou de mauvaise utilisation, des actifs informatiques,
- une politique de contrôle d'accès,
- une politique de contrôle des mots de passe,
- une politique de messagerie,
- une politique internet,
- une politique antivirus,
- une politique de classification des informations,
- une politique d'accès à distance,
- une politique de cession des actifs,
- une politique globale de sécurité de l'information.

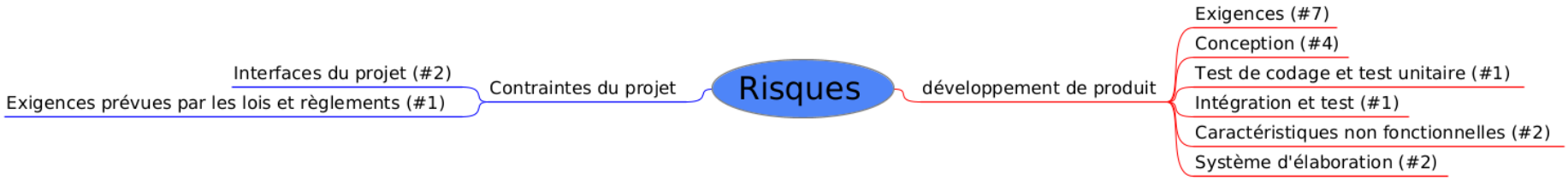
Ainsi, toutes ces politiques seront complémentaires et s'inscriront dans un cercle virtueux d'amélioration continue de la sécurité basé sur l'ISM, telle que représenté graphiquement ci-dessous :



II.D.2. Gestion des risques

Tel qu'il a été annoncé dans le paragraphe précédent, 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 :



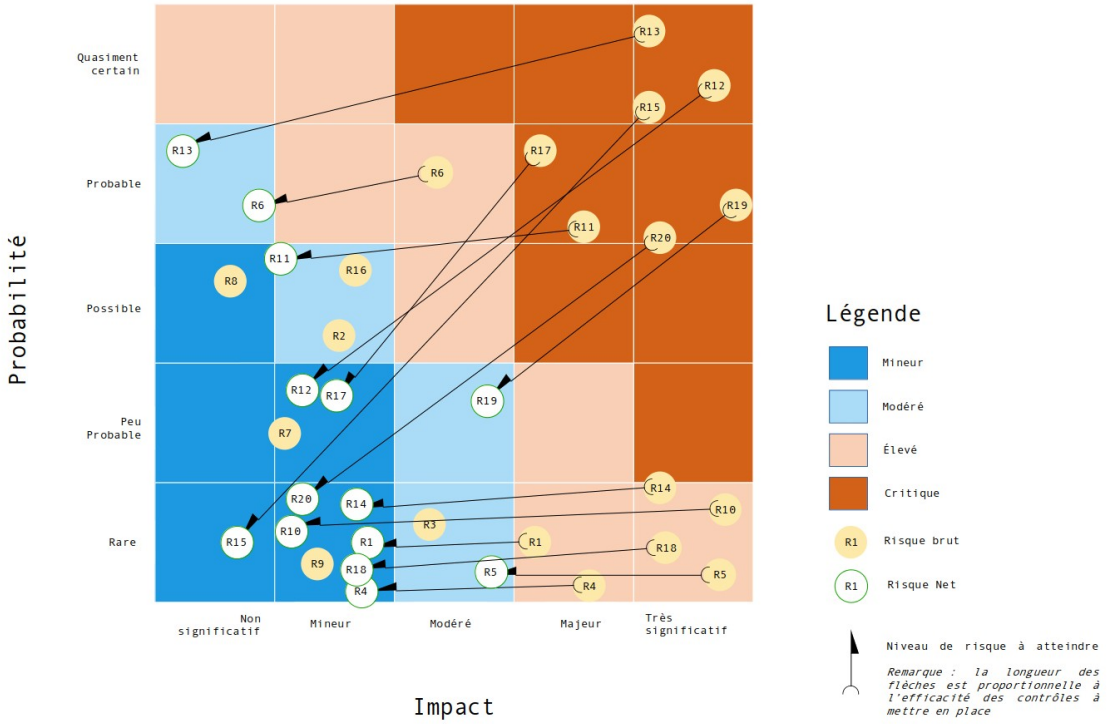
A partir de cette carte heuristique, il est alors possible d'en extrapoler une cartigraphie des risques découlant des domaines et sous-domaines de risques identifiés :

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 *MedHead* 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 équivalent 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		

II.D.2.a.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 long de rétablissement de services d'urgence inacceptable suite à leur interruption.

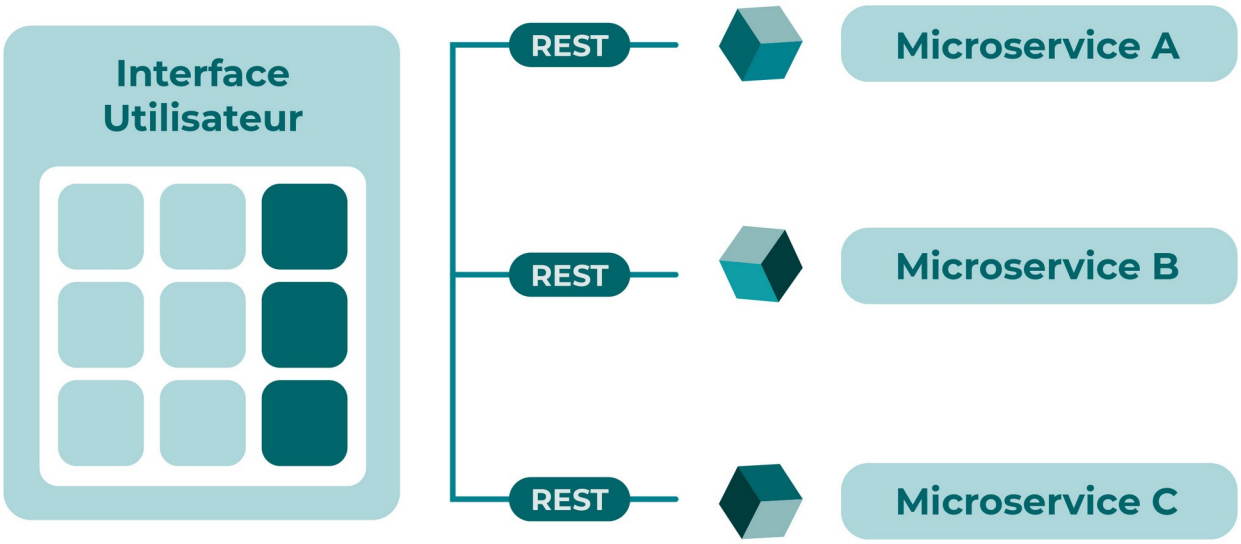
II.E. Architecture technologique

L’architecture technologique préconisée pour mettre en œuvre la solution commune au Consortium, sera basée sur une architecture de microservices communiquant à l’aide d’un protocole synchrone de type HTTP/REST.

II.E.1. Architecture de microservices

II.E.1.a. Description

L'architecture Microservices propose une solution en principe simple : **découper** une application en **petits services**, appelés Microservices, parfaitement **autonomes**, qui exposent une API REST que les autres microservices pourront consommer. La figure ci-dessous expose le principe de fonctionnement d’une telle architecture.

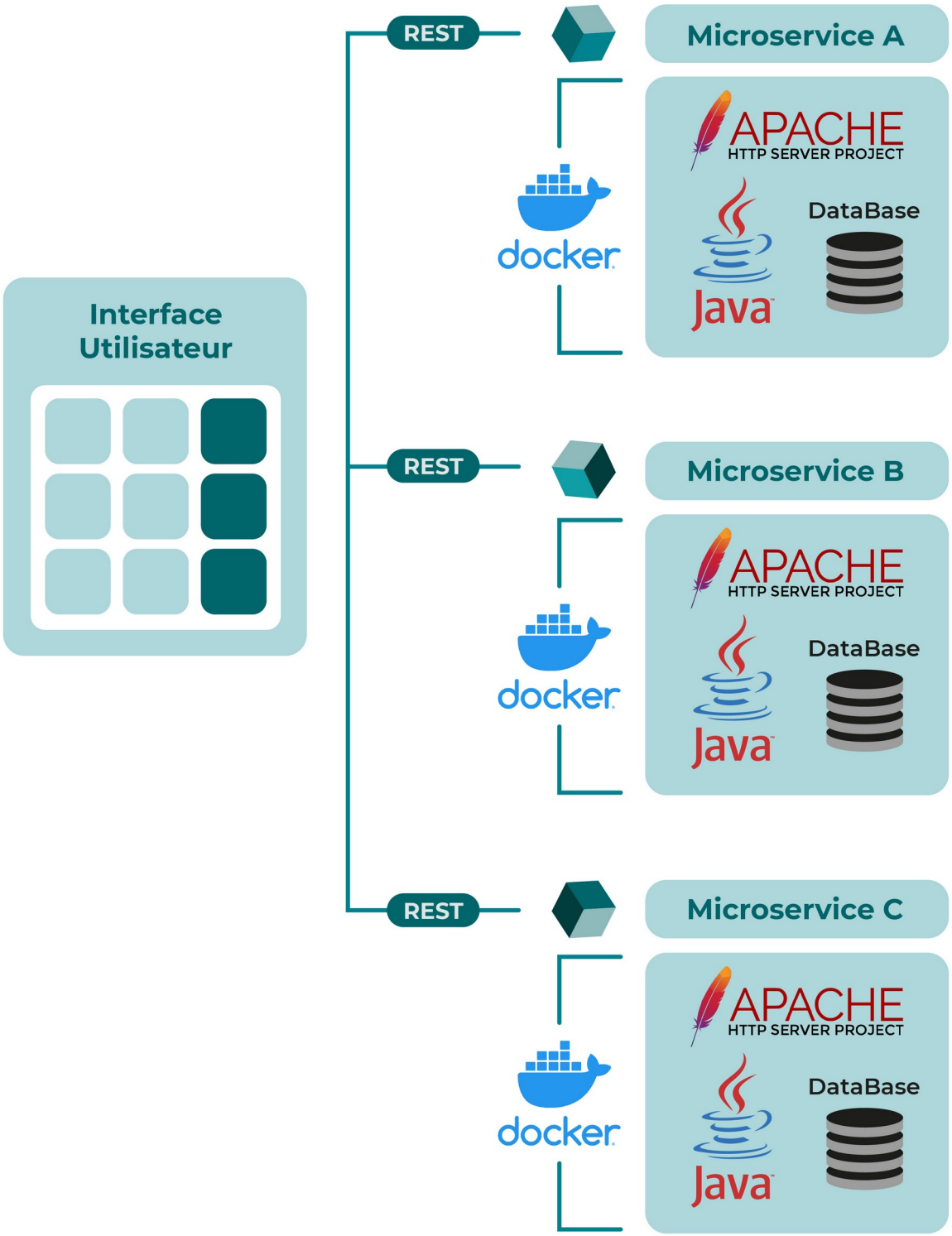


Dans une architecture traditionnelle, l’application est en général diponible au format WAR comportant tous les composants nécessaires à son fonctionnement. Par exemple, lorsqu'un utilisateur demande une fiche de produit, l'application utilise sa logique interne et va puiser dans une base de données, puis produit finalement un HTML résultat.

En reprenant l’exemple précédent dans le contexte de microservice, l'interface utilisateur est elle-même un microservice ayant pour responsabilité d'appeler les autres microservices, et de rassembler cette "fiche produit" partie par partie avant de la servir à l'utilisateur final, peu importe le format du résultat souhaité.

Ici, **chaque microservice est parfaitement autonome** : il a sa propre base de données, son propre serveur d'application (*Tomcat*, *Jetty*, etc.), ses propres librairies, etc. Généralement, ces microservices sont contenus dans un conteneur *Docker*. Ils sont donc totalement indépendants les uns des autres, et vis-à-vis de la machine physique ou logique sur laquelle ils s’exécutent.

Il est maintenant possible de représenter graphiquement ces microservices, de manière plus détaillée, comme suit :



En outre, une architecture en microservices se compose généralement de plusieurs parties :

- un **Gestionnaire d'évènement (Monitoring)** : cette partie représente une activité de surveillance et de mesure des différents microservices en action. C'est une partie permettant la supervision globale du système ;
- un **Planificateur de tâches (Process/Orchestration)** : cette partie permet de planifier l'exécution automatique et périodique de microservices et ainsi de les ordonnancer en vue de rendre leur exécution cohérente et pertinente vis à vis du système global ;
- des **Services de données** : cette partie propose une technologie évoluée d'échange de données via un réseau et dont les données échangées sont regroupées dans un Espace Global de Données distribué dans le réseau pour éviter les problématiques liées aux éventuels goulots d'étranglement et aux pannes de gestionnaires de données ;
- une **Abstraction de données** : cette partie est représentée par un modèle mathématique de types de données définissant le comportement d'un microservice en fonction de la sémantique d'une donnée d'un point de vue utilisateur ;
- la **Sécurité** : la sécurité globale d'un système utilisant des microservice est appliqué indépendamment à chacun d'entre eux afin de leur assurer la confidentialité des données qu'ils traitent, leur authenticité, leur intégrité et leur disponibilité. Chacune des opérations réalisées par un microservice est aussi tracée et imputée de façon unitaire ;
- la **Gouvernance** : cette partie définit le système d'informations global utilisant l'ensemble des microservices. Cette démarche permet de définir la manière dont le système d'informations, au travers des microservices, contribue à la création de valeur en précisant le rôle de chaque microservice utilisé.

II.E.1.b.Avantages

II.E.1.b.i. Le 'Time to market'

C'est certainement l'avantage le plus précieux, celui qui permet d'être plus libre et plus rapide dans l'introduction de nouvelles technologies. Plus innovantes, les entreprises gagnent aussi en compétitivité.

Les microservices peuvent être mis à jour, étendus et déployés indépendamment les uns des autres et par conséquent, beaucoup plus rapidement qu'une architecture plus traidtionnelle.

L'indépendance fonctionnelle et technique des microservices permet l'autonomie de l'équipe en charge sur l'ensemble des phases du cycle de vie (développement, tests, déploiement, et exploitation), et favorise par conséquent l'agilité et la réactivité.

En outre, il est possible pour les développeurs de faire évoluer les microservices dont ils ont la charge selon des cycles de vie différents et donc d'agir selon l'importance et l'urgence, dont notamment en ce qui concerne :

- la création de valeur ;
- la réactivité aux évolutions du marché ;
- la satisfaction des utilisateurs.

II.E.1.b.ii. L’agilité technologique

Le choix technologique dans une MSA est totalement ouvert. Il dépend majoritairement des besoins, de la taille des équipes et de leurs compétences, et peut être adapté aux besoins spécifiques de chaque microservice.

En effet, les microservices étant indépendants et interopérables, ils ne sont pas contraints à une uniformité technologique.

Cette flexibilité technologique permet d’introduire des innovations technologiques progressivement, en limitant le risque encouru : le périmètre impacté est réduit à celui du micro-service concerné.

Il est, par exemple, possible d’utiliser un serveur de bases de données non-relationnel plus performant en écriture et plus extensible en fonction d’un besoin particulier, comme les *Time Series*.

Nota : les bases de données de type Time Series *mesure les changements au fil du temps. Elles proposent des fonctionnalités de gestion du cycle de vie des donées, d’agrégation et de scan de larges rangées d’enregistrements.*

Il sera possible de privilégier une technologie particulière dans chaque microservice pour répondre à un besoin, par exemple, développer des services de calcul scientifique en *C, C++* ou en *Python* pour profiter d’un code efficace en exécution avec des capacités de distribution ou encore de bibliothèques spécifiques, alors qu’un autre service sera développé en *C#*.

II.E.1.b.iii. La modernisation facilitée

Passer à une architecture microservices permet de moderniser son application notamment lors du basculement d’un mode *OnPremise* vers le *Cloud*, comme cela va être le cas pour le Consortium MedHead, ou encore dans le cadre de l’évolution du modèle métier d’une application.

L’approche incrémentale permet de décomposer l’application en fonctionnalités qui sont isolées au sein de microservices. Lorsque l’application est assez bien découpée, elle peut cohabiter avec une architecture microservices, le temps de moderniser le reste de l’application.

II.E.1.b.iv. L’évolutivité

Dans une application, certaines fonctionnalités sont plus utilisées que d’autres.

À mesure que la demande de certains services augmente, il est possible d’étendre les déploiements sur plusieurs serveurs et infrastructures pour répondre spécifiquement aux besoins.

Si un microservice pour une raison ou pour une autre, par exemple pour une question de coût, devait être revu, il serait plus simple de reprendre le développement d’un seul de ces microservices pour l’adapter à la nouvelle plateforme plutôt qu’adapter l’intégralité d’une application monolithique. Ainsi, le délai de mise en production des évolutions fonctionnelles est réduit au délai nécessaire à la mise en œuvre des évolutions du ou des microservices concernés.

Pour les mêmes raisons, la suppression d’un microservice ou son agrégation avec un autre microservice sont également des opérations nettement plus simples à mener que pour une application monolithique.

De plus, les déploiements de microservices en production sont gérés de manière indépendante les uns des autres, ce qui réduit également le coût de mise en production d’une évolution technologique et le risque associé.

Ces éléments favorisent l’évolutivité d’une application constituée de microservices.

II.E.1.b.v. La fiabilité

Lorsqu'ils sont développés correctement, les microservices indépendants n'ont aucun impact les uns sur les autres. Cela signifie que, lorsqu'un élément tombe en panne, l'ensemble de l'application ne cesse pas de fonctionner comme c'est le cas avec le modèle monolithique.

La distribution des microservices en processus systèmes techniquement indépendants permet donc une meilleure continuité de service.

Cette gestion des risques s'accompagne généralement d'un retour sur investissement optimisé.

II.E.1.c. Inconvénients

Au fur et à mesure de l'évolution d'un système d'information vers une MSA, les développeurs applicatifs doivent se soucier de la qualité de service en termes de performance, d'évolutivité et de transparence de la localisation ; ces notions n'étant pas forcément en adéquation avec l'état d'esprit des développeurs. Sans cette prise de conscience de la part des développeurs, le système peut rapidement devenir lourd et présenter des ralentissement, voire des interruptions de services.

Les développeurs doivent alors faire face à la complexité supplémentaire de la création d'un système distribué en :

- implémentant le mécanisme de communication inter-services et gérant les pannes partielles ;
- mettant en œuvre des requêtes couvrant plusieurs services (ce process nécessite une coordination minutieuse entre les équipes) ;
- testant les interactions entre les services ;
- utilisant des outils de développement, tels que des IDE orientés pour la création de microservices. En effet, certains IDE, spécialisés pour la création d'applications monolithiques, ne fournissent pas de support explicite pour le développement d'applications distribuées.

Suite à leur intégration, il persistera une complexité relative au déploiement. En production, la complexité opérationnelle de déploiement et la gestion d'un système composé de nombreux services différents demanderont une attention particulière.

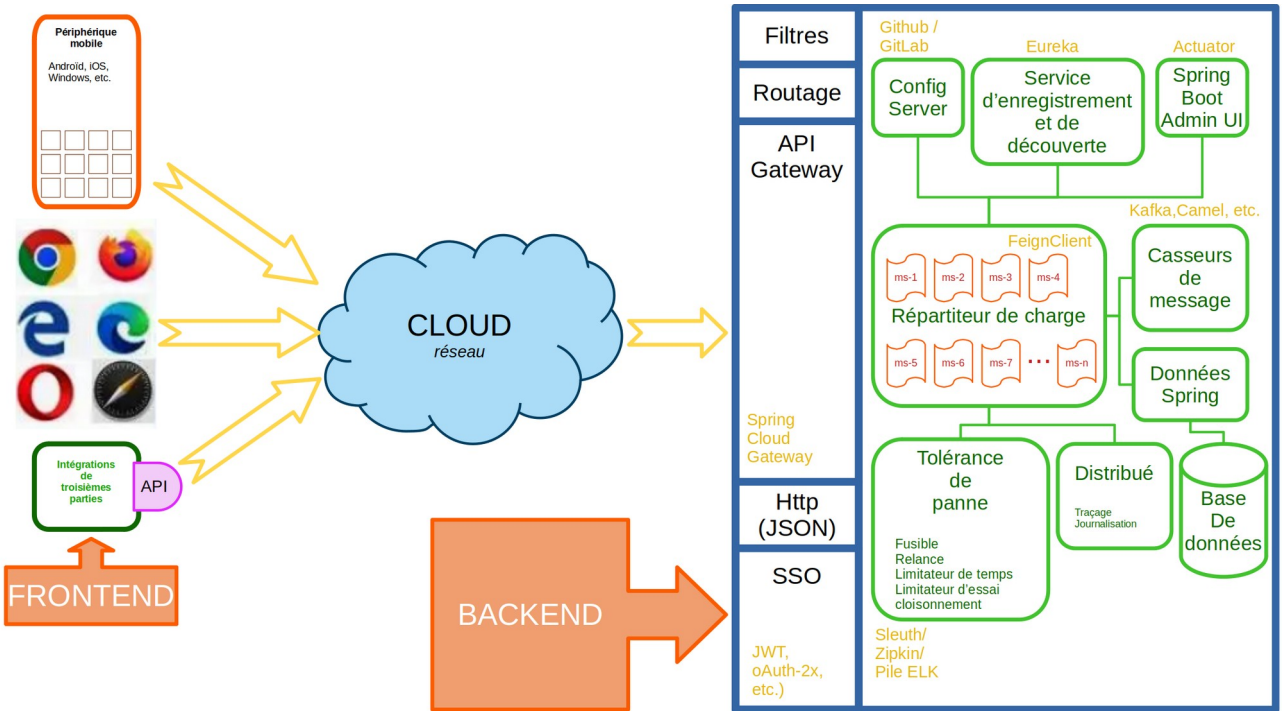
En outre, relativement à la performance, il faudra tenir compte d'une augmentation de la consommation de mémoire : une MSA remplace N instances d'application monolithiques par NxM instances de ses microservices. Si chaque microservice s'exécute dans sa propre JVM (ou équivalent), ce qui est généralement nécessaire pour isoler les instances, il y a alors une surcharge de M fois plus d'exécutions de JVM. De plus, si chaque microservice s'exécute sur sa propre machine virtuelle (par exemple, une instance EC2), comme c'est le cas chez Netflix, la surcharge est encore plus élevée...

II.E.1.d.Utilisation

La mise en place d'une architecture microservices peut être envisagée de deux manières :

1. En utilisant les services **REST** : c'est la manière la plus usitée et simple à mettre en place dans un premier temps. Cependant, cette approche présente des inconvénients. Les services REST sont majoritairement des appels synchrones et, pour certains, ne sont pas rejouables simplement en cas d'erreur (principalement les opérations PUT/POST). La scalabilité, dans ce cas, est fortement conditionnée à la mise en place d'un système de balancing permettant d'équilibrer la charge entre les différents microservices répliqués. On ajoute donc des points de contention au système avec pour conséquence des pertes de performances.
2. En utilisant un **bus de messages** : son rôle étant de transmettre les messages, il doit être le plus simple possible aucune logique ne doit y être intégrée. Le système devient totalement asynchrone et les microservices n'ont aucun lien entre eux. Le système devient plus tolérant aux pannes, si un service tombe le bus conservera le message qui sera alors consommé à la remise en place du service. Dans le cas d'un bus, un système de monitoring doit être mis en place pour observer la consommation des messages et l'état des services en général, le système ne générant pas d'erreur en cas de service indisponible.

Dans le cas d'usage hospitalier de MedHead, la solution applicative préconsiée utilsera la première manière enoncée ci-dessus, et sera donc basée sur une MSA communiquant à l'aide de requête HTTP d'API RESTful, tel que représentée dans le diagramme ci-dessous :



II.E.2. Protocole HTTP/REST

Avant de rentrer dans le vif du sujet, cette étude va définir un certains nombrer d’éléments nécessaires à la compréhension de cette partie, notammant en définissant une API, REST et enfin une API REST.

II.E.2.a. Défintion d’API

Une API est un ensemble de définitions et/ou de protocoles qui facilite la création et l'intégration de logiciels d'applications.

Elle est parfois considérée comme un contrat entre un fournisseur d'informations et un utilisateur d'informations, qui permet de définir le contenu demandé au consommateur (l'appel) et le contenu demandé au producteur (la réponse).

Par exemple, l'API conçue pour un service de météo peut demander à l'utilisateur de fournir un code postal et au producteur de renvoyer une réponse en deux parties : la première concernant la température maximale et la seconde la température minimale.

En d'autres termes, lorsqu’un système souhaite récupérer des informations ou exécuter une fonction, une API permet d'indiquer au système ce qu’il est attendu de lui, afin qu'il puisse comprendre la demande et y répondre.

Il est alors concevable de représenter une API comme un médiateur entre les utilisateurs ou clients et les ressources ou services web auxquels ils souhaitent accéder.

Pour une entreprise, c'est aussi une solution pour partager des ressources et des informations, tout en maintenant un certain niveau de sécurité, de contrôle et d'authentification, en déterminant qui est autorisé à accéder à quoi.

Autre avantage des API : il n’est pas nécessaire de connaître le fonctionnement exact de leur mise en cache, c'est-à-dire de savoir comment les ressources sont récupérées ni d'où elles proviennent.

II.E.2.b. Définition de REST

REST est un ensemble de contraintes architecturales. Il ne s'agit ni d'un protocole, ni d'une norme et les développeurs d'API peuvent mettre en œuvre REST de nombreuses manières.

Lorsqu'un client émet une requête par le biais d'une API RESTful, celle-ci transfère une représentation de l'état de la ressource au demandeur ou point de terminaison. Cette information, ou représentation, est fournie via le protocole HTTP dans un des formats suivants : JSON , HTML, XLT, Python, PHP ou texte brut. Le langage de programmation le plus communément utilisé est JSON, car, contrairement à ce que son nom indique, il ne dépend pas d'un langage et peut être lu aussi bien par les humains que par les machines.

Il y a un autre point à considérer : les en-têtes et paramètres jouent également un rôle majeur dans les méthodes HTTP d'une requête HTTP d'API RESTful, car ils contiennent des informations d'identification importantes concernant la requête (métadonnées, autorisation, URI, mise en cache, cookies, etc.). Il existe des en-têtes de requête et des en-têtes de réponse. Chacun dispose de ses propres informations de connexion HTTP et codes d'état.

Une API RESTful se doit remplir une liste de critères stricte, à savoir :

- une architecture client-serveur constituée de clients, de serveurs et de ressources, avec des requêtes gérées via HTTP ;
- des communications client-serveur stateless, c'est-à-dire que les informations du client ne sont jamais stockées entre les requêtes GET, qui doivent être traitées séparément, de manière totalement indépendante ;
- la possibilité de mettre en cache des données afin de rationaliser les interactions client-serveur ;
- une interface uniforme entre les composants qui permet un transfert standardisé des informations impliquant que :
  - les ressources demandées soient identifiables et séparées des représentations envoyées au client ;
  - les ressources puissent être manipulées par le client au moyen de la représentation reçue, qui contient suffisamment d'informations ;
  - les messages autodescriptifs renvoyés au client contiennent assez de détails pour décrire la manière dont celui-ci doit traiter les informations ;
  - l'API possède un hypertexte/hypermédia, qui permet au client d'utiliser des hyperliens pour connaître toutes les autres actions disponibles après avoir accédé à une ressource.
- Un système à couches, invisible pour le client, qui permet de hiérarchiser les différents types de serveurs (pour la sécurité, l'équilibrage de charge, etc.) impliqués dans la récupération des informations demandées ;
- du code à la demande (facultatif), c'est-à-dire la possibilité d'envoyer du code exécutable depuis le serveur vers le client (lorsqu'il le demande) afin d'étendre les fonctionnalités d'un client.

Bien qu’une API REST doive répondre à l'ensemble des critères édictés ci-dessus, elle est considérée comme étant plus simple à utiliser qe certains protocoles, tel que SOAP est soumis à des contraintes spécifiques, dont la messagerie XML, la sécurité intégrée et la conformité des transactions, ce qui le rend plus lourd et moins rapide.

En outre, puisque REST est un ensemble de directives mises en œuvre à la demande, les API REST sont plus rapides et légères, et offrent une évolutivité accrue. Elles sont donc idéales pour l'IoT et le développement d'applications mobiles.

II.E.2.c. Définition d’une API REST

A prèsent que les définitions d’une API et de REST sont définies, celle d’une API REST devient traiviale :

une API REST, également appelée API RESTful, est une API respectant les contraintes du style d'architecture REST, et permettant d'interagir avec des services web RESTful.

II.E.2.d. Description

Le protocole HTTP/REST est un protocole populaire du World Wide Web dont le concept central repose obligatoirement sur les ressources suivantes :

- **adressabilité** : chaque ressource doit pourvoir être identifié par un URI ;
- **interface homogène** : chaque ressource doit pouvoir être facilement utilisée, de manière homogène, grâce à des méthodes standard, telles que *GET*, *POST* ou *PUT* ;
- **structure client-serveur** : le principe de la structure client-serveur est appliqué lorsque le serveur met un service à disposition qui peut être demandée par un client ;
- **serveur sans état** : la communication entre le serveur et le client doit être sans état, car chaque requête du client vers le serveur doit contenir les informations requises afin de permettre au serveur de traiter la requête, sans qu’il n’y ait de dépendance d'un contexte conservé sur le serveur ;
- **différentes représentations des ressources** : chaque ressource peut afficher différentes représentations en fonction des exigences du client, différents langages ou formats comme HTML, JSON ou XML ;
- **hypermédia** : la mise à disposition des ressources a lieu via Hypermedia, par exemple sous forme d’attributs « href » et « src » dans des documents HTML ou pour l’interface définie des éléments JSON et XML. Ainsi, le client d’une API REST navigue uniquement avec des URLs mises à disposition par le serveur.

Ainsi, le style architectural de l’application REST permet le développement de services structurés, qui sont faciles à intégrer et qui communiqueront le protocole HTTP dans le projet de MedHead.

De plus, grâce à la structure orientée ressources, il est alors possible de renoncer à la recherche de protocoles d’applications lors de la conception d’un service Web REST ; alors que cette étape est nécessaire, voire indispensable, avec des alternatives comparables telles que SOAP.

II.E.2.e. Avantages

Dans le cadre de ce projet, ce sont les méthodes HTTP standard qui seront utilisées, à savoir : GET, POST, PUT/PATCH et DELETE.

Les méthodes HTTP ci-dessus seront mises en œuvre par et pour le développement de service Web REST. Cette architecture présente un avantage indéniable puisqu'elle est avant tout adaptée à la gestion de données simples.

En effet, outre le fait que l'architecture REST elle-même livre d'excellents moyens pour constituer et mettre en place des services Web de toutes sortes, il faut prendre en compte la grande compatibilité des périphériques du marché avec le protocole HTTP. Actuellement, la quasi-totalité des appareils est compatible avec l'HypertexT Transfer Protocol, les clients mobiles et desktop peuvent travailler avec l'interface REST sans encombre et sans applications supplémentaires.

Ainsi, les services Web se caractérisent grâce à cela par les critères suivants :

- indépendance vis-à-vis des plateformes,
- évolutivité,
- performance,
- interopérabilité,
- flexibilité.

Leur utilisation requiert néanmoins un minimum de connaissances au préalable, notamment lorsque différentes ressources son utilisées en parallèle.

Pour les habitués des protocoles tels que SOAP, il demeure plus difficile d'effectuer la transition avec REST, mais cela permet de bénéficier d'un service pratique et utilisable dans différents contextes de travail.

II.E.2.f. Inconvénients

Le principe même du *stateless*, ou sans-état, des ressources REST peut sembler diminuer les possibilités de l'architecture, voire volontairement diminuer celles-ci, et c'est très exactement ce qu'il fait.

En effet, avec un traitement adéquat des ressources, l'interface REST offre de nombreuses possibilités pour pallier à cet état de fait réducteur, en dehors du simple ajout d'ensemble de données, comme par exemple pour :

- les services Web avec traitement des transactions : pour mettre sur place un projet de service Web avec traitement des transactions, un gestionnaire est indispensable. Sachant que toutes les ressources sont sauvegardées entre deux requêtes de par leur nature sans-état, l'utilisation de REST offre deux possibilités :

    1. les ressources sont créées de manière à ce que les transactions puissent être travaillées au cours d'une requête ;

    2. une ressource est créée de manière à ce que les requêtes soient gérées par les transactions. Chaque requête créée automatiquement une autre ressource qui sera identifiée par un URI et représente une transaction qui va de pair. Les modifications peuvent par la suite être stockées sur le serveur.

- Les Services Web asynchrones : il est souhaitable pour certains services Web que la requête et la réponse soient couplées. Comme HTTP ne propose pas de mécanismes pour cela, la seule option qui persiste consiste à gérer soi-même les requêtes en tant que prestataire de services et de transmettre les requêtes précises des utilisateurs ;

- les services Web avec interopérabilité : les services REST se démarquent par leur grande flexibilité. Les clients mobiles bénéficient également de la souplesse de l'architecture REST et les ressources sont trouvées facilement par les moteurs de recherche.

II.E.2.g.Utilisation

Afin de mettre en place un service Web, il sera donc indispensable dans ce projet d'avoir recours à l' HTTP/REST, et sa version cryptée HTTPS très répandue et dont tous les ports requis des pare-feux existants sont ouverts.

De plus, comme HTTP est construit de manière relativement simple, sans avoir recours à des implémentations en particulier, celui-ci définit de nombreuses commandes, avec lesquelles il est possible d'avoir recours pour accéder aux différentes ressources :

Méthode	Description
GET	Envoie une demande à la ressource en question du serveur, sans changer le statut.
POST	Permet de créer une nouvelle ressource sous la ressource indiquée, qui sera adressée automatiquement à partir de l'URL. Peut également être utilisé pour modifier des ressources déjà existantes.
HEAD	Envoie une demande à l'en-tête de la ressource du serveur, par exemple pour s'assurer de la validité d'un fichier.
PUT	Dépose la ressource en question sur le serveur ou modifie une autre ressource.
PATCH	Modifie une partie de ladite ressource.
DELETE	Supprime la ressource.
TRACE	Renvoie la requête telle qu'elle a été obtenue par le serveur, pour que les modifications soient effectuées de la même manière.
OPTIONS	Dévoile une liste des méthodes soutenues par le serveur.
CONNECT	Remet la question via un tunnel SSL, pour établir une connexion via un serveur Proxy.

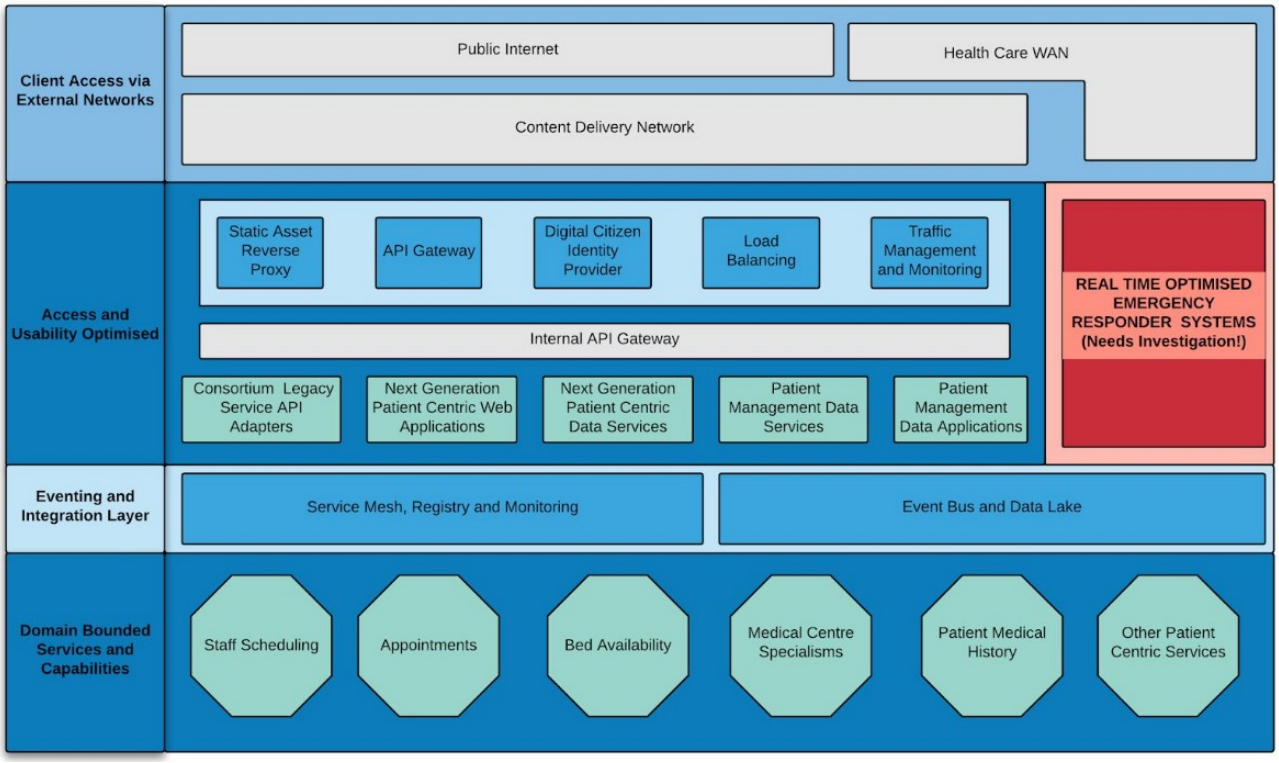
La palette de commandes présentée ci-dessus peut être élargie à l'aide de l'implémentation d'autres protocoles, tels que le protocole WebDAV comprenant lui-aussi plusieurs méthodes utiles, telles que :

- **COPY** permettant la copie des ressources ;
- **MOVE** permettant le déplacement des ressources ;
- **LOCK** permettant de verrouiller des ressources ;
- **UNLOCK** permettant de déverouiller des ressources ;
- **MKCOL** permettant de créér un répertoire.

II.F. Architecture des nouveaux systèmes et intégration avec leur(s) partenaire(s)

II.F.1. Architecture du système d'intervention d'urgence

Au sein du graphique de l'architecture cible présentée dans le document de définition d'architecture, il est possible d'y décerner le positionnement du système d'intervention temps réel, représenté dans un encart rouge dans le graphique :

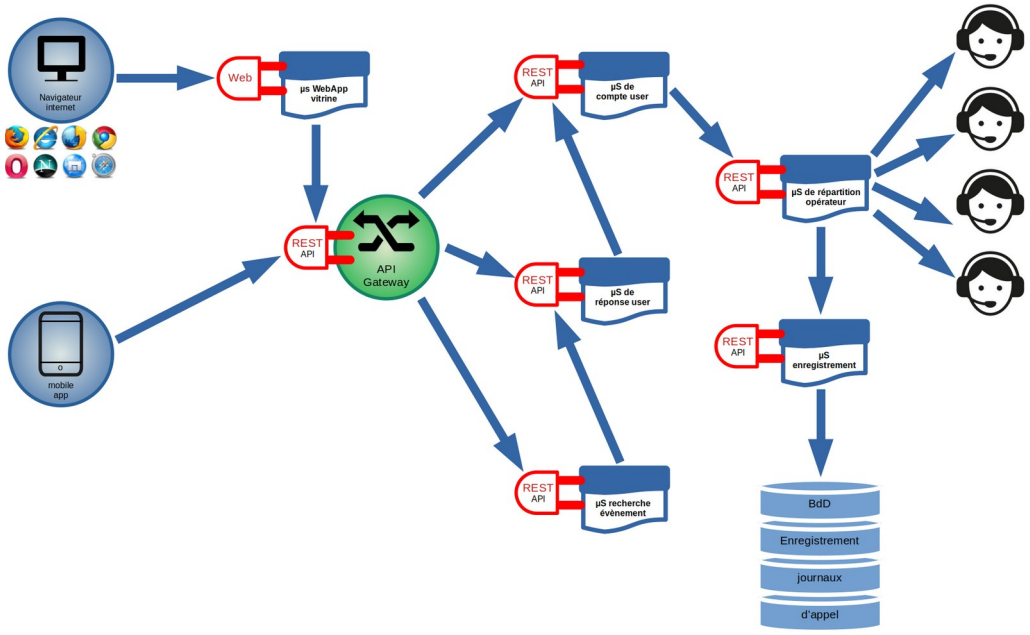


Il va maintenant devoir définir un design réfléchi et structuré représentatif de la PoC souhaitée. Ce paragraphe va donc énoncer les concepts de design-clés qui rendront l'aPI de la PoC facile d'utilisation et *scalable*/évolutive. Pour ce dernier point, il s'agira pour la PoC de pouvoir s'adapter et fonctionner même en cas de forte augmentation, soudaine, du trafic, des demandes ou juste d'un grand nombre de fonctionnalités implémentées.

Cette étude va alors répondre à quatre questions :

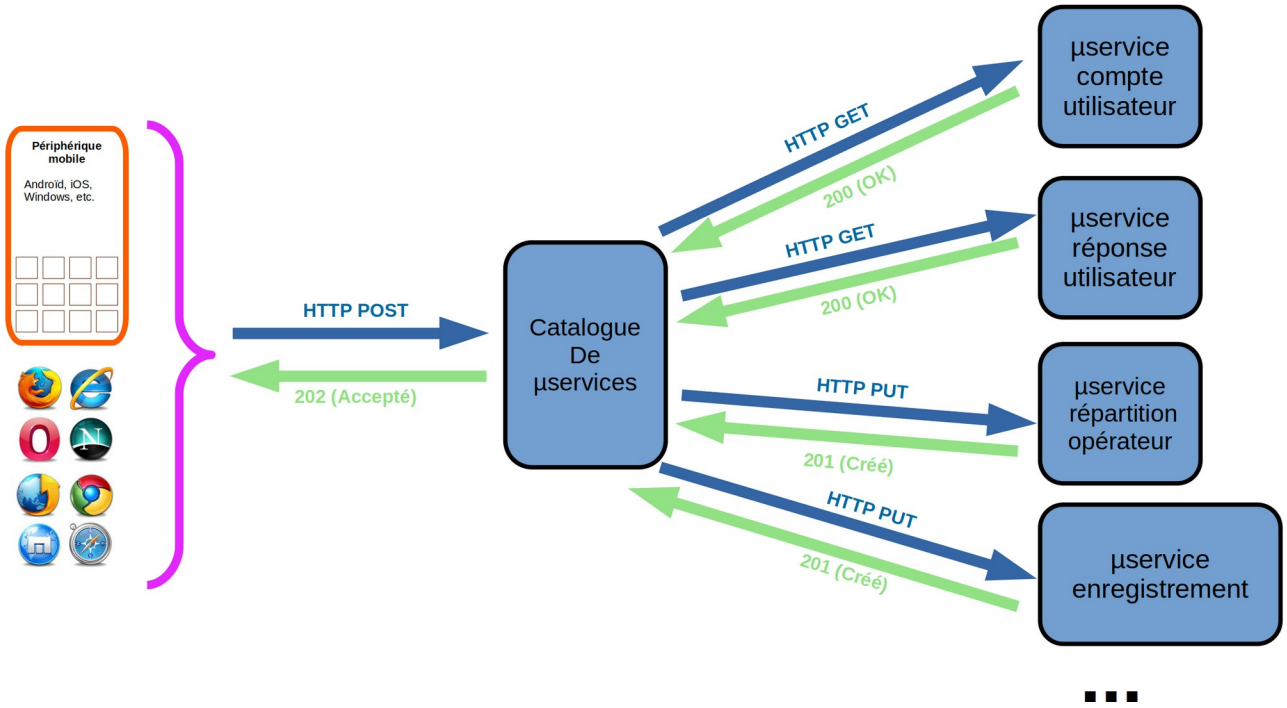
- De quel type d'endpoints la PoC a-t-elle besoin ?
- Quelle ressource doit-elle créer ?
- De quelle ressources la PoC a-t-elle besoin pour effecteur les opérations CRUD principales ?
- Pour chaque ressource identifiée, les quatre opérations CRUD sont intégralement ou partiellement nécessaires.

Imbriqué à l’architecture ci-dessus, l’encart rouge du graphique représentant la gestion des appels d’urgence pourra alors se représenter graphiquement comme suit :

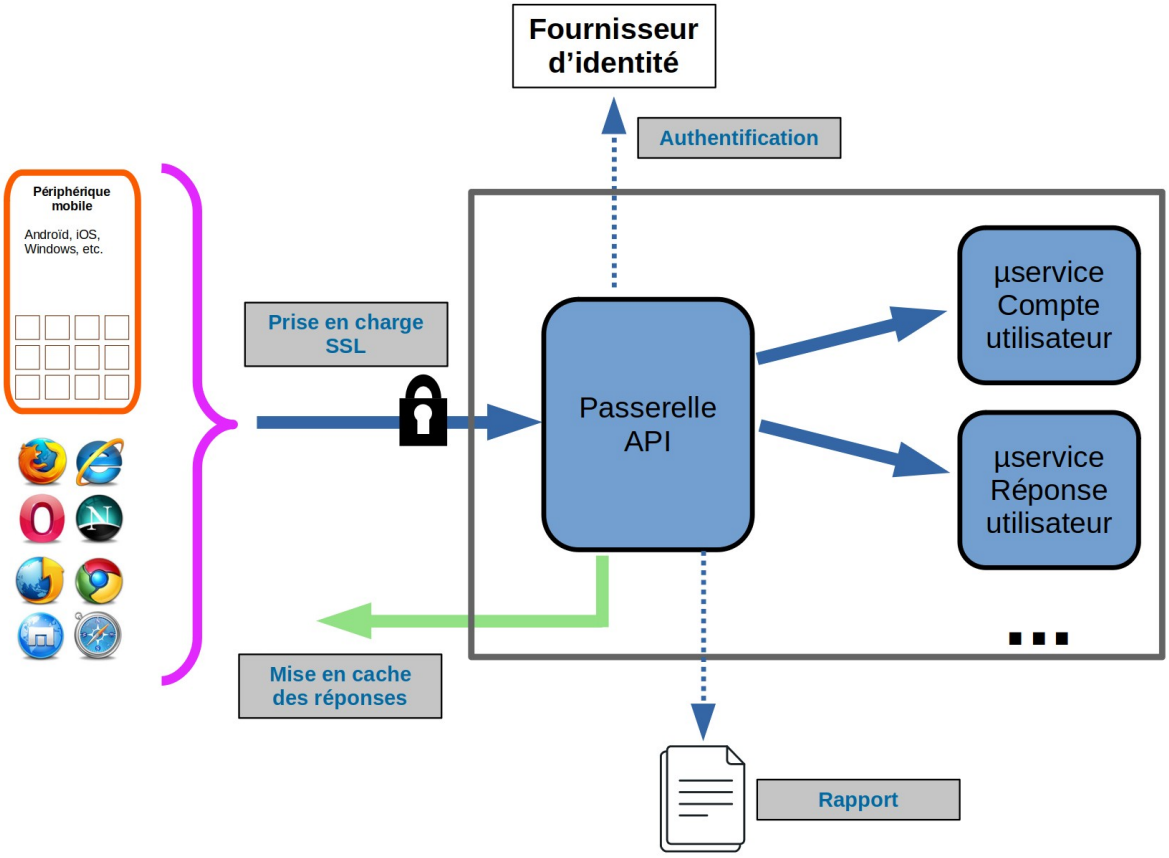


Ainsi, le diagramme précédent pourrait aussi se représenter deux autres façon :

- une représentation mettant en avant le catalogue de microservices :



- une représentation plus orientée en tant que passerelle API :



II.F.2. Format des messages d’urgence

Maintenant que le système d’urgence a été présentée, dont notamment l’API de sa passerelle, il est à présent le moment de se questionner sur le type de message qui y transiteront quant à leur formalisme et à leur contenu. Pour mémoire, ces messages seront primordiaux pour faire communiquer les instances de commandements avec les équipes de terrain et vice et versa.

Le système sera donc conçu pour échanger des informations au sein de messages formattés, interprétables et compréhensibles à la fois pour des opérateurs humains que pour des ordinateurs. L’objectif d’un tel système sera donc de :

- améliorer l’interopérabilité entre les différents organes d’intervention d’urgence ;
- fournir les règles, les constructions et le vocabulaire pour les messages textuels formattés ;
- s’assurer que les messages incluent les données essentielles et primordiales décrivant la situation d’urgence ;
- diminuer le risque d’incompréhension ou de désinformation relatives à un message d’intervention d’urgence.

II.F.2.a. Concept

Les messages d’intervention d’urgence se composeront de deux parties; un **en-tête** de transport (par exemple SMTP) et une charge utile d'information, le **corps**, tels que :

- l'**en-tête du message** se rapporte au protocole de transport utilisé comprenant des informations telles que l'expéditeur, le destinataire, la classification et les informations d'acheminement ;
- le **corps du message** contiendra les informations réelles à communiquer ; c'est cette partie du message qui peut contenir un message MTF.

A noter que les messages MTF seront eux-mêmes indépendants du protocole de transport utilisé et peuvent être communiqués de nombreuses manières par tous types de protocoles choisis.

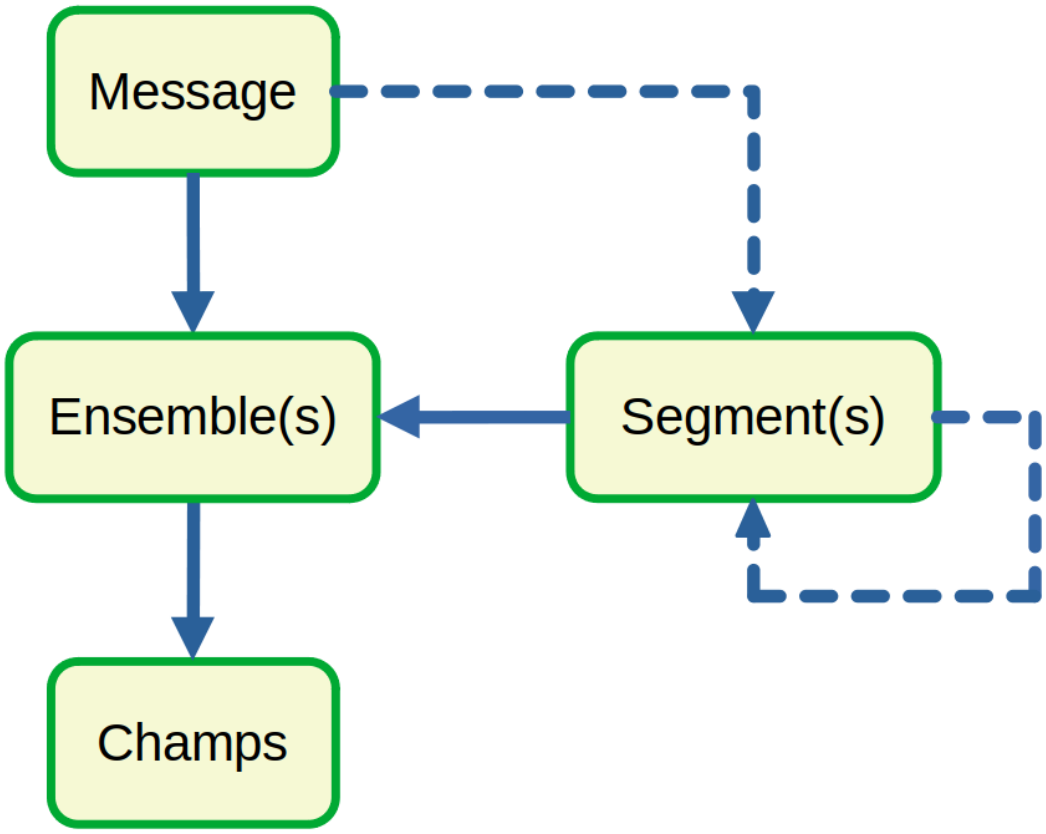
Ainsi, un message formaté est défini dans un langage simple adapté à l’échange d'informations axées sur les personnes.

Le système comprend ainsi les règles régissant la représentation des définitions conceptuelles convenues et les dispositions de ces représentations dans des formats prédéterminés.

L'application des règles d'échange d'informations, ainsi que leurs exigences, aboutit à un inventaire ouvert des représentations convenues, c'est-à-dire des formats de champ, tel que la définition des formats eux-mêmes et des formats de messages qui sont conçu conformément à un ensemble défini de règles pour produire une structure formelle.

II.F.2.b. Structure de format

La structure d'un message d'intervention d'urgence pourra se représenter selon le diagramme ci-dessous :



Les messages formatés ont une structure formelle qui doit être suivie selon des règles définies entre les principales composantes d'un MTF. Chaque message formaté est généré à partir d'une IER.

L'IER sert donc à préciser les informations à échanger dans le cadre de la mission. Elle décrit les tâches clés, le degré d'interopérabilité requis et les paramètres des systèmes de communication et d'information concernés.

L'ordre des composants d'un message, c'est-à-dire message, segment(s), ensemble(s) et champs, sont prédéfinis dans le format et sont appelés positions de format. Ainsi, le contenu d'un message formaté est un consitué d'un ordre d'élément sous le forme d'une collection subdivisée en 'Ensembles' couvrant chacun un aspect spécifique du message. Les ensembles dans un message peut être regroupés au sein de segments et un segment peut être imbriqué dans un autre segment.

Ainsi, de manière systèmique, un message peut être constitué :

- soit de segment(s), d'ensemble(s) et de champs ;
- soit d'ensemble(s) et de champs.

Ainsi, un format de message a un identifiant (MTFID) unique et une séquence spécifique de segment(s) et/ou d'ensemble(s). Le MTFID est un mot-clé unique et mnémonique qui associe le type d'information qui doit être fourni par le message. Il est contenu dans l'ensemble d'identification de message (MSGID) qui se trouve au début de chaque message. Il identifie également les ensembles autorisés par le format du texte du message et l'ordre dans lequel ils doivent être disposé.

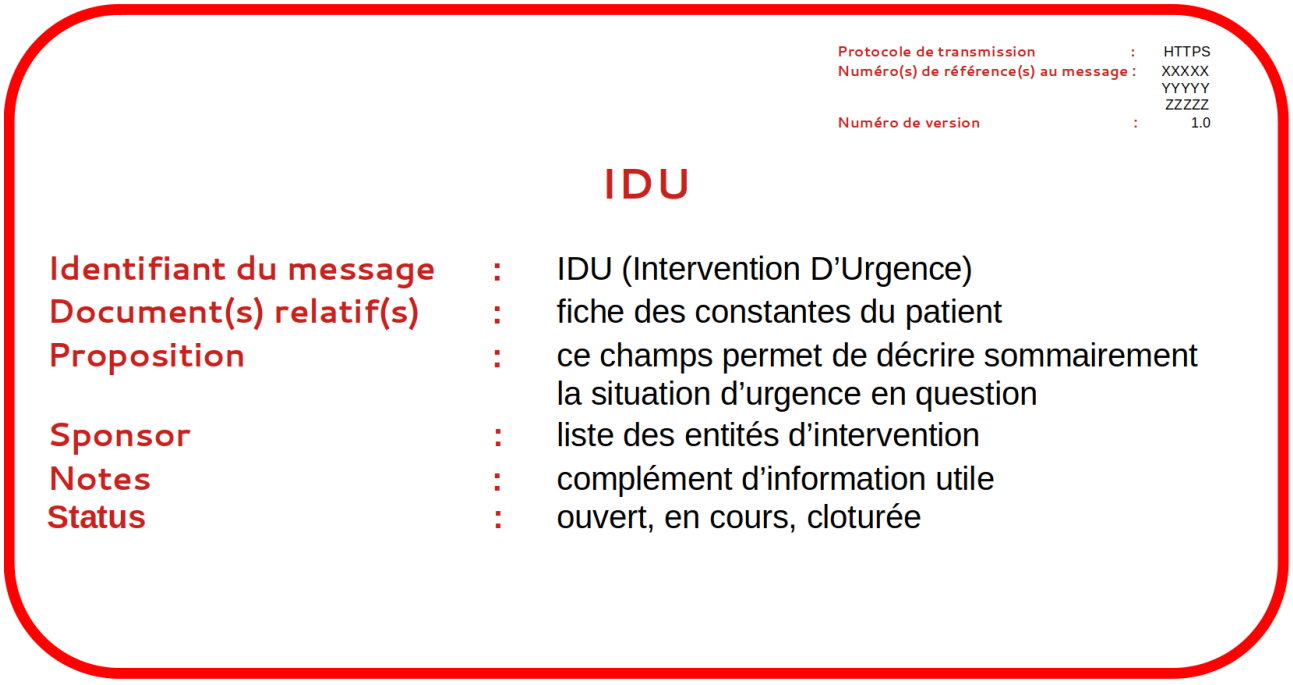
Le MTF suit donc une structure logique et sans ambiguïté basée sur les techniques et procédures tactiques (TTP) opérationnelles, ce qui aidera les opérateurs à s'assurer que toutes les informations nécessaires sont incluses dans un message dans un format que le destinataire peut comprendre ou traiter dans le système.

Le contenu réel d'un MTF se compose uniquement des valeurs/textes des champs plutôt que des en-têtes de champ ou de la mise en forme, ce qui les rend très efficaces en termes de bande passante. Les codes abrégés reconnus seront à définir avec les parties prenantes métiers ; cela augmentera encore l'efficacité de leur bande passante, permettant une transmission efficace même dans les environnements les plus difficiles avec des communications telles que les radios tactiques, HF et VHF.

Le catalogue de messages devra être la somme d'expériences condensées des besoins opérationnels en matière d'échange d'informations dans des milieux d'urgence contraints.

Les MFT devront constamment être mis à jour et travaillé par des groupes de travail afin de s'assurer de leur adéquation opérationnelle, peu importe le milieu ou la spécialité médicale impliquée.

L'en-tête d'un message d'intervention d'ugence pourra être représenté selon le graphique ci-dessous :



En qui concerne le corps d’un message d’intervention d’urgence celui-ci pourra être structuré selon plusieurs colonnes spécifiant chacune une information spécifique, telles que :

- **Seg** : un groupement d’ensemble au sein d’un segment numéroté ;
- **Alt** : une proposition d’alternative aux soins prodigués ;
- **Rpt** : la répétititon d’un ou plusieurs ensemble(s) au sein du message ;
- **Occ** : la catégorie d’occurence au sein d’un ensemble ;
- **SETID** : l’identifiant unique de l’ensemble dont il est question ;
- **Seq** : le numéro de séquence de l’ensemble en question ;
- **SFN** (Set Format Name) : le nom de l’ensemble dont il est question ;
- **Description** : une brève description/information.

Exemple de représentation du corps d’un message d’intervention d’urgence :

Seg	Alt	Rpt	Occ	SETID	Seq	SFN	Descript.
	1#			UDI	1	EXERCISE IDENTIFICATION	

Associé au MTF, il sera nécessaire de rédiger un dictionnaire de données décrivant spéicifiquement chaque type de message d’intervention d’urgence pouvant décrire chaque situation sanitaire particulière.

En outre, deux positions de format d'ensemble séquentielles ou plus, dans un format de message qui sont liées par leur contenu, peuvent être désignées comme une structure de segment. Les instances correspondant à une telle structure de segment peut se produire autant de fois que nécessaire dans un format des messages. Il n'y a pas de limite au nombre de fois qu'un segment peut être répété dans une instance d'un message ; cependant, dans certains messages, le nombre de répétitions d'un segment peut être restreint dans la conception même du message. Toutes les restrictions sont incluses à la fin de la ligne de départ du segment.

Enfin, en spécifiant une limite d'une seule occurrence, un segment non répétable peut être présent dans un message.

En ce qui concerne spécifiquement la colonne **Alt**, des ensembles/segments adjacents peuvent avoir une interdépendance. Les conditions autorisées pour les ensembles alternatifs sont :

- les ensembles/segments sont mutuellement exclusifs mais un seul est requis, c'est-à-dire une seule des alternatives doit être sélectionnée et ainsi un seul nombre est affiché.
- Les ensembles/segments sont mutuellement exclusifs mais aucun n'est requis, c'est-à-dire qu'une seul des alternatives peut être sélectionnée.
- Au moins une des alternatives est requise mais plusieurs peuvent être utilisées, c'est-à-dire qu’une ou plusieurs alternatives doivent être sélectionnées.

II.F.2.c. Exemple de message

Le code exemple fourni ci-dessous n'est pas opérationnel et présente, à des fins pédagogiques, un fichier XML formatté pouvant représenter une fiche d'intervention d'urgence :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:Header>
    <xs:InterventionDUrgence mtfid="UDI" xmlns:xs="urn:nato:mtf:app-11(d):1:aco"/>
    <xs:MessageIdentifier>
        <MessageTextFormatIdentifier>UDI</MessageTextFormatIdentifier>
        <Standard>APP-11(D)</Standard>
        <Version>1</Version>
        <Originator>STATION MOBILE INTERVENTION</Originator>
        <ReferenceTimeOfPublication>
            <DateTimeIso>
                <Year4Digit>2023</Year4Digit>
                <MonthNumeric>01</MonthNumeric>
                <Day>12</Day>
                <TimeDelimiter>T</TimeDelimiter>
                <HourTime>10</HourTime>
                <MinuteTime>30</MinuteTime>
                <SecondTime>30</SecondTime>
                <TimeZone>Z</TimeZone>
            </DateTimeIso>
        </ReferenceTimeOfPublication>
        <Qualifier>URG</Qualifier>
        <SerialNumberOfQualifier>1</SerialNumberOfQualifier>
        <MessageSecurityPolicy>GPOGlobal</MessageSecurityPolicy>
        <MessageClassification>
            <SecurityClassification>PROTECT</SecurityClassification>
        </MessageClassification>
        <MessageSecurityCategory>MEDICAL</MessageSecurityCategory>
    </xs:MessageIdentifier>
</xs:Header>
<xs:Body>
    <xs:OperationCodeword>
        <OperationCodeword>Intervention</OperationCodeword>
        <PlanOriginatorAndNumber>
            <PlanOriginator>SACEUR</PlanOriginator>
            <BlankSpaceCharacter> </BlankSpaceCharacter>
            <PlanNumber>106</PlanNumber>
        </PlanOriginatorAndNumber>
        <OptionNickname>PAPER WASTE</OptionNickname>
        <SecondaryOptionNickname>ORANGE</SecondaryOptionNickname>
        <description>Chute du 4ème étage</description>
        <supported>SAMU</supported>
        <Conveyance>Hélicoptère</Conveyance>
    </xs:OperationCodeword>
</xs:Body>
```

Le fichier XML mis en exemple n'est pas contractuel et n'a pour objectif d'exposer le type de messages qui pourraient être générés.

En outre, le fichier XSD associé définira chaque champs de l'exemple.

Néanmoins, comme il a été énoncé précédemment, ce fichier XSD, ainsi que le dictionnaire de données l'accompagnant devra être réalisé en collaboration avec des parties prenantes expertes dans le domaine de la Santé et des interventions d'urgence.

Il est cependant évident de distinguer deux parties dans ce fihier XML :

- `<xs:Header>...</xs:Header>` relatif aux informations descriptives du message lui-même ;
- `<xs:Body>...</xs:Body>` relatif aux données sémantique du message.

Ce formattage de message est issu du catalogue de message de l'OTAN APP-11 et de l'ADatP-3.

Source https://www.systematicinc.com/products/a/military-messaging/app-11-and-adatp-3/

Le même exemple de code de message présenté ci-dessus peut aussi être rédigé avec du JSON :

```
{
      "Header" : [
            "InterventionDUrgence": {
                  "mtfId" : "UDI",
                  "xmlns:xs" : "urn:nato:mtf:app-11(d):1:aco"
            },
            "MessageIdentifier" [
                  { "MessageTextFormatIdentifier" : "UDI" },
                  { "Standard" : "APP-11(D)" },
                  { "Version" : "1" },
                  { "Originator" : "STATION MOBILE INTERVENTION" },
                  { "ReferenceTimeOfPublication" : [
                        "DateTimeIso" : [
                              { "Year4Digit" : "2023" },
                              { "MonthNumeric" : "01" },
                              { "TimeDelimiter" : "T" },
                              { "HourTime" : "10" },
                              { "MinuteTime" : "30" },
                              { "SecondTime" : "30" },
                              { "TimeZone" : "Z" }
                        ]
                  {
            ] ,
            { "Qualifier" : "URG" },
            { "SerialNumberOfQualifier" : "1" },
            { "MessageSecurityPolice" : "GPOGlobal" },
            { "MessageClassification" [
                  { "SecurityClassification" : "PROTECT" }
            ] },
            { "MessageSecurityCategory" : "MEDICAL" }
      ],
      "Body" : [
            "OperationCodeWord" : [
                  { "PlanOriginatorAndNumber" [
                        { "PlanOriginator" : "SACEUR" },
                        { "BlanckSpaceCharacter" : " " },
                        { "PlanNumber" : "106" }
                  ] },
                  { "OptionNickname" : "PAPER WASTE" },
                  { "SecondaryOptionNickname" : "ORANGE" },
                  { "description" : "Chute du 4ème étage" },
                  { "supported" : "SAMU" },
                  { "Conveyance" : "Hélicoptère" }
            ]
      ]
}
```

III. Procédures spécifiques de changement de périmètre

Durant le développement, plusieurs contextes pourront être rencontrés pour chaque microservice développé. Ces contextes particuliers, ou les problèmes qu'ils pourraient engendrer lors du développement, pourront être appréhendés en utilisant des modèles conception, appelés communément Design Pattern.

En outre, tous les microservices développés devront manipuler des données ; il faudra donc établir des modèles de gestion qui définiront des règles de gestion de ces données.

III.A. Design Pattern

Ainsi, relativement aux paragraphes précédents cette étude à identifier plusieurs modèles de conception qui seront forcément utilisés lors du développement :

- pattern **ADAPTER** : ce modèle de conception fournit l'interface qu'un client attend en utilisant les ervices d'une classe dont l'interface est différente.
- Pattern **FACADE** : ce modèle de conception fournit une interface simplifiant l'emploi d'un sous-système.
- Pattern **COMPOSITE** : ce modèle de conception permet aux clients de traiter de façon uniforme des objets individuels et des compositions d'objet.
- Pattern **BRIDGE** : ce modèle de conception découple un objet, ou une classe d'objets, qui s'appuie sur des opérations abstraites de l'implémentation de ces opérations. Ainsi, cela permet à l'objet, ou à sa classe, de varier indépendamment.
- Pattern **SINGLETON** : ce modèle de conception garantit qu'un objet, ou une classe d'objet, ne possède qu'une et une seule instance UNIQUE, en fournissant un point d'accès global à celle-ci.
- Pattern **OBSERVER** : ce modèle de conception définit une dépendance du type un-à-plusieurs (*1*,*n*) entre les objets de manière à ce que lorsquin objet change d'état, tous les objets dépendant en soient notifiés et soient actualisés afin de pouvoir réagir conformément.
- Pattern **MEDIATOR** : ce modèle de conception définit un objet qui encapsule la façon dont un ensemble d'objets interagissent. Cela promeut un couplage faible, ou laĉhe, en évitant aux objets d'avoir à se référer explicitement les uns aux autres. Cela permet également de faire varier leur interaction indépendamment.
- Pattern **CHAIN OF RESPONSABILITY** : ce modèle de conception évite de coupler l'émetteur d'une requête à son récepteur en permettant à plus d'un objet d'y répondre.

Néanmoins, cette liste est non exhaustive et devra être enrichie tout au long du processus de développement.

En outre, bien que c'est modèle de conception soient impostants à la réalisation du projet MEDHEAD, il y en a deux indispensables que cette étude va développer plus en détail ; il s'agit des :

- pattern **MICROSERVICE**;
- pattern **DISCOVERY**.

III.A.1. pattern MICROSERVICE

Ce modèle de conception se présente essentiellement lors du développement d'une application d'entreprise côté serveur pour laquelle il est nécessaire de prendre en charge une variété de clients différents, y compris les navigateurs de bureau, les navigateurs mobiles et les applications mobiles natives.

L'application peut également exposer une API à des tiers et s'intégrer à d'autres applications via des services Web ou un courtier de messages. Elle gère les requêtes (requêtes et messages HTTP) en :

- exécutant la logique métier ;
- accédant à une base de données ;
- échangeant des messages avec d'autres systèmes ;
- renvoyant une réponse HTML/JSON/XML.

Il existe des composants logiques correspondant à différents domaines fonctionnels de l'application.

Néanmoins, la question relative au déploiement d'un tel pattern va se poser à un moment ou à un autre et il faudra en tenir compte dès la définition de l'architecture de la solution.

En effet, une architecture basée sur un pattern MICROSERVICE, structure l'application comme un ensemble de services collaboratifs faiblement couplés. Ainsi, chaque microservice est :

- hautement maintenable et testable, ce qui permet un développement et un déploiement rapides et fréquents ;
- librement associé à d'autres microservices : ce qui laisse un grande marge de manœuvre à une équipe pour travailler de manière indépendante, la majorité du temps, sur son ou ses microservices, sans être affectée par les modifications apportées aux autres microservices, et sans affecter les autres microservices ;
- déployable indépendamment : l'équipe peut alors déployer son microservice sans avoir à se coordonner avec d'autres équipes ;
- capable d'être développé par une équipe restreinte : cette notion est essentiel pour une productivité élevée en évitant de faire appel au haut responsable de la communication des grandes équipes...

Ainsi, les microservices communiquent à l'aide de protocoles synchrones tels que HTTP/REST ou de protocoles asynchrones tels que AMQP ou SOAP. Les microservices peuvent alors être développés et déployés indépendamment les uns des autres et chacun possède sa propre base de données afin d'être le plus découplé possible des autres microservices.

Enfin, la cohérence des données entre les microservices peut être est maintenue à l'aide de modèle de gestion de données.

III.A.2. pattern DISCOVERY

La force des microservices réside dans leurs approche communautaire dont le mot d'ordre pourrait être « *l'union fait la force* ». En effet, les microservices doivent généralement s'appeler mutuellement.

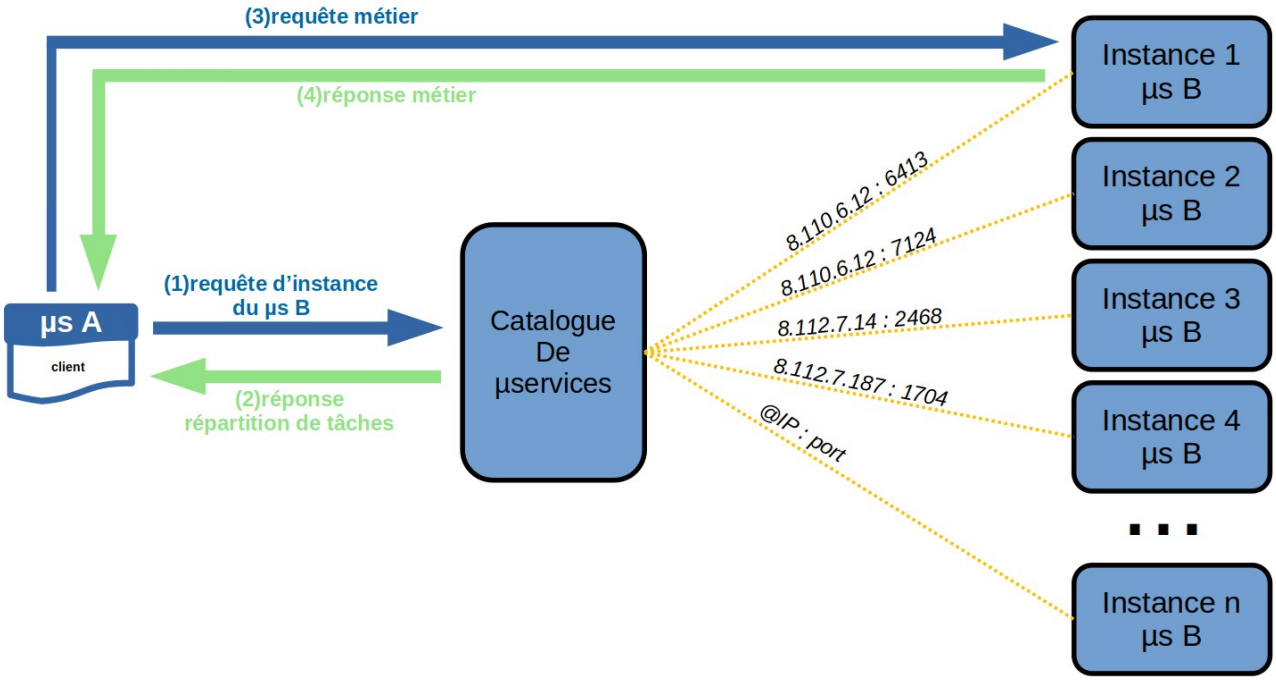
Dans une application monolithique, les services s'invoquent les uns les autres via des appels de méthode ou de procédure au niveau du langage. Dans un déploiement de système distribué traditionnel, les services s'exécutent à des emplacements fixes et bien connus (hôtes et ports) et peuvent donc facilement s'appeler à l'aide de HTTP/REST ou d'un mécanisme RPC.

Cependant, une application moderne basée sur des microservices s'exécute généralement dans des environnements virtualisés ou conteneurisés où le nombre d'instances d'un service et leurs emplacements changent de manière dynamique.

Par conséquent, il sera nécessaire d'implémenter un mécanisme permettant aux clients du service d'adresser des requêtes à un ensemble dynamique d'instances de service éphémères.

Néanmoins, le problème pour le client serait alors de trouver un moyen pour découvrir la localisation de l'instance du service destinataire. La réponse à cette question est fourni par un microservice tiers faisant office de registre central, et il faut également considérer que chaque microservice possède plusieurs instances de lui-même pouvant s'exécuter sur plusieurs machines différentes.

En effet, lorsqu'un client fait une demande à un autre service, il obtient alors l'emplacement d'une instance active du service destinataire, en interrogeant un registre de services connaissant les emplacement de toutes les instances de tous les services en activité. Ce procédé peut se représenter graphiquement comme suit :



Ainsi, la découverte côté client présente un avantages indéniable relatif au faible nombre de pièces mobiles et de sauts de réseau par rapport à la découverte côté serveur.

Néanmoins, la découverte côté client présente également deux inconvénients :

- il associe le client au registre des services, ce qui peut être considéré comme une contrainte ;
- il est nécessaire d'implémenter une logique de découverte de service côté client pour chaque langage/framework de programmation utilisé par l'application.

III.B. Modèle de gestion de données

Comme cette étude l'a présenté précédemment, la solution sera basée sur une MSA pour laquelle il va être indispensable de réfléchir à la gestion des données manipulées par les microservices.

Ainsi, les données seront au cœur de la plateforme de MedHead et son succès reposera sur une gestion efficace des données ; dit autrement, si des données sont perdues, la plateforme ne sera tout simplement pas viable.

Cependant, pour pallier à ce risque, il existe 6 modèles de gestion des données qui permettront de gérer les données efficacement :

- le **base de données par service** : avec ce modèle, chaque microservice doit posséder ses propres données. La communication et l'échange de données ne peuvent alors se faire qu'en utilisant un ensemble d'API bien définies. La définition du contexte de limite pour chaque microservices est très importante pour éviter d'obtenir une application *spaghetti*.
- La **base de données partagée** : ce modèle est une option sûre lors de la migration d'une application monolithique vers MSA. Il permet également de mettre en œuvre facilement les transactions ACID (atomicité, cohérence, isolation et durabilité).
- Le **modèle Saga** : ce modèle est la solution pour mettre en œuvre des transactions commerciales couvrant plusieurs microservices. Fondamentalement, c'est une séquence de transactions. Avec SAGA, chaque microservice qui effectue une transaction publiera un événement. Le microservice suivant qui effectue la prochaine transaction dans les chaînes se déclenchera par la sortie de la transaction précédente, puis continuera jusqu'à la dernière transaction dans les chaînes. En cas d'échec de l'une des transactions en chaîne, SAGA exécutera une série d'actions de secours pour annuler l'impact de toutes les transactions précédentes.
- La **composition API** :ce modèle est complémentaire à la solution de base de données par service. La mise en œuvre de base de données par service signifie qu'il n'est pas possible d'obtenir de commandes «*JOIN*» pour obtenir des données de différentes tables si celles-ci sont situées sur une base de données différente, peuvant être sur des emplacements différents. La solution est alors de mettre en œuvre un micrservice d'*API Composer* qui appellera d'autres microservices dans l'ordre requis pour obtenir des données, puis il effectuera une jointure en mémoire des données avant de les fournir au consommateur.
- Le **CQRS**: la séparation des commandes et des requêtes permet au modèle d'entrée et de sortie de se concentrer davantage sur la tâche spécifique qu'ils exécutent. Cela simplifie également le test des modèles car ils sont moins généralisés et ne sont donc pas surchargés de code supplémentaire.

- L’**Event Sourcing** : ce modèle nous aide à résoudre le problème de la mise à jour atomique de la base de données et de la publication d'un événement. Il devient alors possible d’utiliser ce modèle en conjonction avec CQRS. Ainsi, en faisant cela, de nombreux défis liés à la gestion des événements et à la maintenance des données de requête peuvent être résolus.

Dans le cadre de ce projet, il sera alors préconisé que **chaque microservice prenne en charge sa propre base de données**, et qu’il n’y ait ainsi pas de base de données partagée. Ce cloisonnement volontaire aidera non seulement à l’indépendance et au couplage faible de chaque microservice, et contribuera également à la sécurité d’ensemble de l’application.

IV. Rôles, responsabilités et livrables

Toutes les parties prenantes de ce projet ayant été identifiées au sein de l’organigramme du Consortium de MedHead, ce paragraphe va néanmoins préciser les rôles de chacun d’entre eux.

Ainsi, chaque membre de MedHead devra participer, chacun à son niveau, à la détermination de l’essence du projet, telle que la vision, la mission, les attentes, les motivations, les objectifs et les buts, le concept, l’envergure, les priorités, les spécifications, les dépendances, la détermination de l’environnement ou la constitution des livrables.

En prenant en compte l’historique du Consortium, il sera donc indispensable que cette vision globale soit partagée par tous afin de rendre disponible les informations sur le projet ; le but étant de ne les laisser ni clairsemées, ni imprécises.

Aussi, comme chaque acteur impliqué dans le projet aura éventuellement un effet sur son déroulement, la vision globale devra donc :

- concilier des opinions divergentes,
- être collectivement discutée,
- approuvée par les instances de gouvernance,
- être mise à jour, au besoin.

En outre, après la réalisation de cette preuve de concept, un suivi régulier devra être effectué afin de perpétuer cette vision globale.

Enfin, tous les efforts nécessités par la construction d’une compréhension commune et exhaustive du projet, devront faire l’objet d’un stockage minutieux pour ne pas *perdre* ce précieux gisement d’informations. Celui-ci constituera une base solide afin de faciliter la prise de décisions ultérieure.

Plus concrètement, au sein de la population de MedHead, et relativement à ce projet de preuve de concept, il n’est pour l’instant concevable que d’identifier un rôle de membre du Consortium ayant chacun un domaine opérationnel sous sa responsabilité, tel que présenté dans le document de définition d’architecture.

Ces différents rôles sont détaillés et décrits dans le tableau ci-dessous :

Nom	Rôle et organisation	Fonction
Kara « *Starbuck* » Trace	CIO, Ursa Major Health *pilote au sein de la flotte coloniale du Galactica*	Intégration des systèmes de médecine générale en temps quasi réel avec des prestataires de soins de santé.
Anika Hansen	PDG, Jupiter Scheduling Inc	Architecture de domaine consolidée avec des événements sécurisés circulant en temps réel entre les systèmes.
Équipe d'intégration des systèmes de santé du Royaume-Uni	Emergency Expert Systems	Enrichissement des données et accès aux données anonymisées des hôpitaux et des patients.
Chris Pike	Architecte métier principal, Schedule Shed	Réductions de pannes. Flux d'événements normalisés et infrastructure de sécurité pour réduire la responsabilité de l'organisation causée par des erreurs de planification.

IV.A. Positionnement des parties prenantes

Cette section permettra de développer une bonne compréhension des parties prenantes les plus importantes et d'enregistrer cette analyse pour référence.

Cette partie du document sera, également, à actualiser au cours du projet.

Nom Prénom	Fonction	Capacité à perturber le projet	Compréhension courante	Compréhension requise	Engagement actuel	Engagement requis	Support requis
Kara « *Starbuck* » Trace	CIO	H	TBD	H	TBD	H	H
Anika Hansen	PDG	H	TBD	H	TBD	H	H
Équipe d'intégration des systèmes	Emergency Expert Systems	H	TBD	H	TBD	H	H
Chris Pike	Architecte, Schedule Shed	M	TBD	H	TBD	M	M

Légende :

- H : *Haute*
- M : *Moyen*
- B : *Bas*
- TBD : To be defined (trad. *à définir)*

IV.B. Gestion des parties prenantes

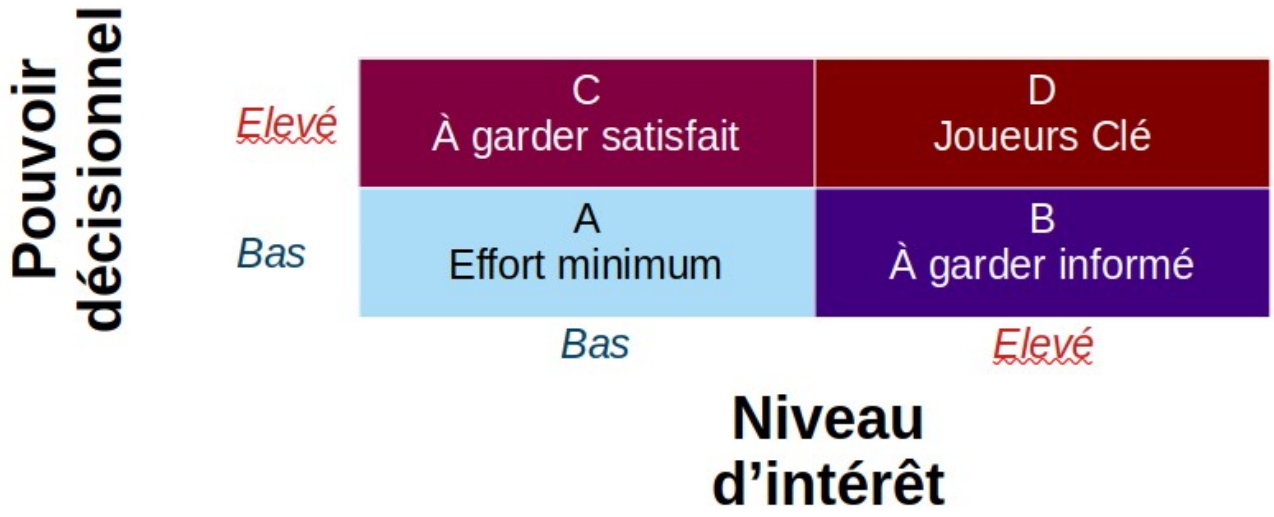
Les paragraphes précédents ont identifié une liste de personnes et d'organisations concernées par le projet d'architecture d'entreprise.

Certains d'entre eux peuvent avoir le pouvoir de bloquer ou d'avancer. Certains peuvent être intéressés par ce que fait l'initiative d'Architecture d'Entreprise, d'autres peuvent ne pas s'en soucier.

Cette étape permet aux équipes d'AMOA et de MOE d'identifier facilement quelles parties prenantes sont censées être des bloqueurs et/ou des critiques, et quelles parties prenantes sont susceptibles d'être des défenseurs et des partisans de l'initiative.

Ainsi, il est primordial de déterminer le pouvoir, l'influence et l'intérêt des parties prenantes, de manière à concentrer l'engagement de l'architecture d'entreprise sur les personnes clés. Ceux-ci peuvent être cartographiés sur une matrice pouvoir/intérêt, qui indique également la stratégie à adopter pour s'engager avec eux.

Ainsi, la matrice représente la catégorisation à réaliser pour chaque partie prenante, en fonction de son pouvoir décisionnel et de son intérêt pour le projet :



A partir de la matrice ci-dessus, nous pouvons l'appliquer à l'identification des parties prenantes réalisée précédemment :

Nom Prénom	Fonction	Niveau d'implication
Kara Trace	CIO	C – à garder satisfait
Anika Hansen	PDG	C – à garder satisfait
Equipe d'intégration des systèmes de Santé	Responsable des systèmes expert d'urgence	D - joueur clé
Chris Pike	Architecte métier	D - joueurs clé

Tout comme les précédents paragraphes précédents, celui-ci sera à maintenir à jour durant le cycle de vie du projet si de nouvelles parties prenantes étaient identifiées.

IV.C. Engagement sur les livrables à fournir

Cette section a pour objectif d'identifier les catalogues, les matrices et les diagrammes que l'engagement d'architecture doit produire et valider avec chaque groupe de parties prenantes pour fournir un modèle d'architecture efficace.

Il est important d'accorder une attention particulière aux intérêts des parties prenantes en définissant des catalogues, des matrices et des diagrammes spécifiques qui sont pertinents pour un modèle d'architecture d'entreprise particulier.

Cela permettra de communiquer l'architecture afin qu'elle soit comprise par toutes les parties prenantes, tout en leur permettant de vérifier que l'initiative d'architecture d'entreprise répondra à leurs préoccupations.

Nom Prénom	Fonction	Catalogue, Matrices et schémas à lui fournir
Kara Trace Anika Hansen	CEO	• Diagramme de l'empreinte commerciale • Diagramme But/Objectif/Service commercial • Diagramme de décomposition de l'organisation • Catalogue des capacités métier • Matrice capacité/organisation • Carte des capacités de l'entreprise • Matrice Stratégie/Capacité • Matrice capacité/organisation • Diagramme du modèle d'affaires • Catalogue de flux de valeur • Catalogue des étapes de la chaîne de valeur • Matrice de flux de valeur/capacité • Carte de la chaîne de valeur
Chris Pike Equipe d'intégrations des systèmes de santé	Responsable technique & chef de maintenance technique	• Catalogue des exigences • Diagramme du contexte du projet • Diagramme des avantages • Diagramme de l'empreinte commerciale • Diagramme de communication d'application • Carte de l'organisation • Catalogue des capacités métier • Matrice capacité/organisation • Carte des capacités de l'entreprise • Matrice Stratégie/Capacité • Matrice capacité/organisation • Diagramme du modèle d'affaires • Catalogue de flux de valeur • Catalogue des étapes de la chaîne de valeur • Matrice de flux de valeur/capacité • Carte de la chaîne de valeur

V. Procédures et critères d’acceptation

La réussite de ce projet dépendra de la capacité de l’équipe de développement à répondre aux besoins décrit dans le *§Objectifs et besoins fonctionnels*.

La communication entre les parties prenantes et l’équipe de développement jouera un rôle essentiel dans la livraison d’une solution correspondant aux exigences du produit et du marché.

De façon générale, les difficultés surviennent quand les besoins sont expliqués de façon vague et que l’équipe ne peut obtenir d’exigences claires. Afin d’éviter ce genre de situation, il est primordial de disposer d’une documentation logicielle de haute qualité, incluant des *User Stories* et des critères d’acceptation au sein d’une documentation d’exigences.

Une US (*User Story)* est une description formelle rédigée en langage naturel et décrivant une et une seule fonctionnalité souhaitée. Une US est généralement accompagnée de critères d’acceptation qui sont des conditions de sortie que l’US doit satisfaire pour être acceptée par un l’émetteur de l’US.

V.A. Procédures d’acceptation

Pour établir chacun des critères d’acceptation décrits, le plan de tests suivra les étapes suivantes :

- **clarification de la portée de la fonctionnalité** : les critères d’acceptation définiront les limites de l’*epic* ou de l’*US* en fournissant des détails précis sur leurs fonctionnalités nécessaires à l’équipe pour comprendre **QUAND elles seront terminées** et **SI elles fonctionneront comme prévu**.
- **description des scenarii négatifs** (non passants) : ces scenarii représentent l’ensemble des cas de tests non passants, c’est à dire ne devant jamais aboutir. Par exemple, les critères d’acceptation pourront exiger que le système reconnaisse les entrées de mot de passe dangereuses et ainsi empêcher un utilisateur de continuer. Le format de mot de passe invalide est un exemple de scénario dit négatif, lorsqu’un utilisateur effectue des entrées non valides ou se comporte de manière inattendue. Les critères d’acceptation définissent ces scénarios et expliquent comment le système doit y réagir.

- **amélioration de la communication** : les critères d'acceptation synchroniseront les visions des parties prenantes et de l'équipe de développement, en garantissant que tout le monde a une compréhension commune des exigences définies : les développeurs sauront exactement quel type de comportement la fonctionnalité doit démontrer, tandis que les parties prenantes comprendront ce qui est attendu de la fonctionnalité.

- **rationalisation des tests d'acceptation** : les critères d'acceptation seront la base des tests d'acceptation de chaque *epic* et *US* définies, et chacun d'entre eux devra pouvoir être testé indépendamment et décrire des scenarii de réussite ou d'échec clairs et précis.

- **estimation de fonctionnalité** : les critères d'acceptation préciseront ce qui doit être développé exactement par l'équipe technique. Une fois que l'équipe de développement aura des exigences claires, précises et sans ambiguïté, elle divisera les *epics* fonctionnelles en *US* pouvant être correctement estimées.

V.B. Critères d'acceptation

Les critères d'acceptation sont les conditions que le projet devra remplir pour être accepté par les parties prenantes. Chaque critère d'acceptation sera associé à un plan de test, et pourra intervenir à deux niveaux différents :

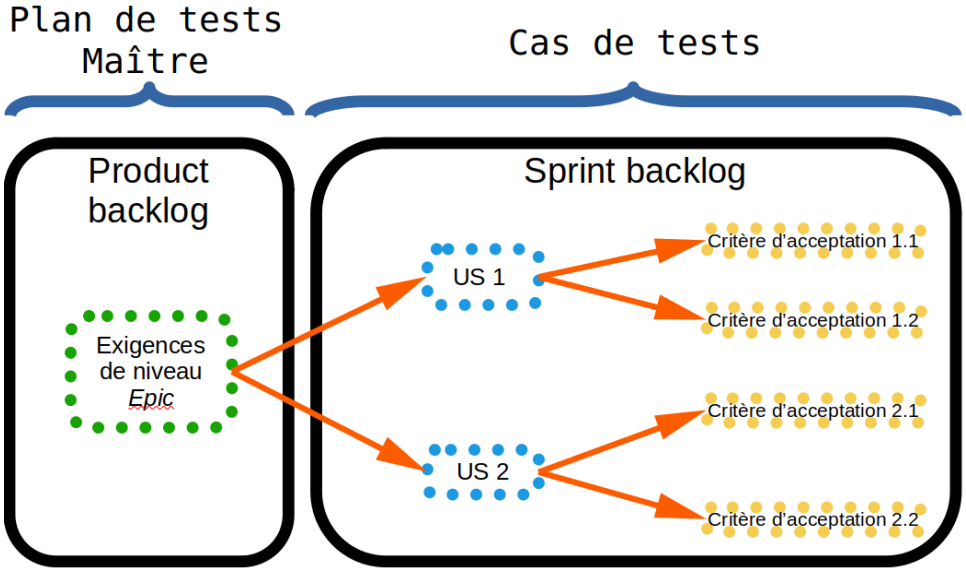
- le plan de tests Maître, appelé également *plan de tests Projet*, dont l'objectif est d'implémenter le stratégie de test sur le projet en particulier, sera applicable au niveau d'un *Product backlog* ;

- le plan de tests de niveau, appelé aussi communément *cas de tests*, ayant pour but de décrire les activités précises à mettre en œuvre pour chaque niveau de tests (tests unitaires, tests d'intégration, tests système et tests d'acceptance), sera applicable au niveau d'un *Sprint backlog*.

Chaque critère, associé à chaque *epic* ou *US,* sera unique et définira le comportement de la fonctionnalité décrite au sein de l'*epic* ou l'*US,* du point de vue de l'utilisateur final.

Des critères d'acceptation bien rédigés aident à éviter des résultats inattendus à la fin d'une phase de développement, et garantissent que toutes les parties prenantes et tous les utilisateurs sont satisfaits de ce qu'ils obtiennent.

Ces critères d'acceptation peuvent être schématisés selon le diagramme suivant :



Le format d'écriture permettant de définir un cas de test comprendra à la fois une **méthode orientée scénario**, **connue sous le nom de *Given* / *When* / *Then* (GWT)**, et également le **critère d'acceptation** lui-même, **sous la forme d'un résultat attendu**.

- **Étant donné** (*Given*) une condition préalable ;
- **Quand** (*When*) une action est réalisée ;
- **Alors** *(Then)* ce résultat est attendu.

Cette approche, héritée du *Behavior Development Driven*, fournit une structure cohérente aidant les testeurs à définir quand commencer et quand terminer le test d'une fonctionnalité particulière. Cette méthode permet également de réduire le temps consacré à la rédaction des cas de test, puisque le comportement du système est décrit à l'avance.

Chaque critère d'acceptation sera alors rédigé selon le format ci-dessous :

Campagne - Responsable	***le nom de la campagne qui sera décrite – le nom du responsable du test***
Scénario	***le nom du comportement qui sera décrit***
Étant donné	*définit l'état initial du scénario*
Quand	*spécifie le moment à partir duquel l'utilisateur effectue l'action*
Ensuite	*le résultat de l'action dans "Quand"*
Et	*ajoute une assertion au point précédent*
Critère d'acceptation	*présente le résultat attendu à l'issue du test*

Lorsqu'elles sont combinées, ces instructions couvrent toutes les actions qu'un utilisateur entreprend pour terminer une tâche et connaître le résultat.

VI. Critères de priorisation et KPI d’implémentation

Avant de rentrer dans le détail de certains critères de priorisation ou des KPI à utiliser, il est nécessaire d’aborder quelque peu la méthode qui sera appliquée pour appréhender l’ensemble des étapes énumérées dans les paragraphes précédents.

En effet, il sera considéré l’établissement préalable d’un *Product Backlog* recensant toutes les TÂCHES FONCTIONNELLES (*epic*) à réaliser pour l’aboutissement de ce projet.

Une fois ce *Product Backlog* réalisé, celui-ci sera répercuté au sein d’un *Sprint Backlog*, recensant toutes les TÂCHES TECHNIQUES à implémenter durant un jalon temporel (*sprint*) ; le *Sprint Backlog* ne considérera que quelques tâches fonctionnelles à chaque *sprint*.

Ainsi, relativement à cette notion de *Sprint Backlog*, tous les KPI dont il sera question au sein de cette section existeront pour :

- **évaluer la PRIORISATION des tâches FONCTIONNELLES au sein du *Product Backlog ;***
- **estimer l’AVANCEMENT des tâches TECHNIQUES au sein du *Sprint Backlog*.**

Ainsi, pour chaque phase de chaque étape, cette étude utilisera deux notions complémentaires :

- des critères de priorisation,
- des KPI d’avancement.

VI.A. Critères de priorisation

Tel qu'il a été annoncé en introduction, les changements induits par ce projet devront répondre à des critères de productivité et d'efficacité, tout en prenant en compte l'aspect humain de ceux qui mettront en œuvre et qui utiliseront, à terme, ce nouveau système.

Ainsi, pour aider à prendre des décisions de priorisation relatives à la progression de ce projet, cette étude propose cinq critères pour ordonner et séquencer les tâches fonctionnelles présentes au sein du *Product Backlog* :

- **L'urgence :** ce critère permet de distinguer les tâches fonctionnelles qui doivent être réalisées, coûte que coûte. Le risque de ne pas réaliser ces tâches pourrait engendrer une fatalité au sein de l'entreprise. Bien que ce critère puisse être considéré comme extrême, Rep'Aero se doit de capitaliser sur toutes ses précédentes expériences et devenir une organisation proactive, engagée dans une démarche de prévention des situations d'urgence.

- **L'effort requis :** ce critère peut être considéré comme la somme du coût, du nombre d'heure, du niveau d'énergie à fournir estimé ou du nombre de personnes et/ou de spécialités nécessaire à la réalisation d'une tâche. Ce facteur permettra de distinguer les tâches les plus prenantes des autres.

- **L'impact du résultat :** ce critère peut se résumer en une seule question : quel est l'état amélioré attendu ? En définissant clairement la raison d'être et l'intérêt d'une tâche, il sera alors possible de définir celles qui auront le plus d'impact sur la vision produit.

- **Le contexte socio-environnemental favorable :** étant donné que l'objectif du projet est l'obtention d'un succès, le contexte socio-environnemental est un facteur pouvant énormément influer sur celui-ci. En effet, ce critère permet d'identifier les terrains glissants et les tâches à réaliser avec précaution, voire à ne pas réaliser pour le moment si le contexte situationnel ne le permet pas...néanmoins, ce n'est pas parce que ce n'est pas le moment de réaliser une tâche qu'il faut l'oublier ; dans ce contexte, il sera alors simplement utile d'en diminuer sa priorisation de réalisation.

- **Le plaisir :** ce critère peut paraître subjectif et il l'est ! Néanmoins, une entreprise est riche de son potentiel humain et la motivation en est le moteur. A contrario, la démotivation des employés peut avoir de graves incidences sur leur santé et sur la santé de l'entreprise elle-même. Aussi, fort de ce constat, ce critère devra prendre en considération les envies ou préférences des utilisateurs pour prioriser grandement les tâches associées aux demandes les plus fortes.

VI.B. KPI d’avancement

Tel qu’il a été mentionné plus haut, les KPI d’avancement utilisés seront associés à l’établissement d’un *Sprint Backlog*, recensant toutes les tâches techniques (*user’s stories*) à réaliser pour répondre à une ou plusieurs tâche(s) fonctionnelle(s) (*epic*).

Ainsi, les KPI préconisés ici seront associés aux quatre notions suivantes :

- la temporalité,
- le budget,
- la qualité traduite par les ressources utilisées,
- l’efficacité.

Ces derniers devront suivre le concept des objectif SMART, à savoir :

- Spécifique,
- Mesurable,
- Atteignable,
- Réaliste,
- Temporellement défini.

La mise en place de KPI permettra de jauger l’évolution du projet en identifiant rapidement les écarts entre les réalisations et les objectifs, permettant ainsi de prendre les mesures qui s’imposent.

VI.B.1. KPI de temporalité

Ce type de KPI englobe trois domaines :

- l’**écart de durée** : ce KPI permettra d’appréhender si la réalisation d’une tâche est plus longue que son estimation initiale. Il est obtenu par l’opération suivante :

  (durée réelle – durée initiale) / durée initiale

- l’**écart de délai** : ce KPI va exposer si le projet est en avance ou en retard sur le budget ou le temps estimé initialement. Il est obtenu par l’opération suivante :

  budget initialement prévu – budget actuellement utilisée

  Temps initialement prévu – Temps actuellement utilisée

  Si le résultat de ces opérations est négatif, c’est que le projet utilise moins de budget que prévu ou avance plus vite que les estimations

- le **taux de dépassement** : ce KPI donnera un pourcentage du dépassement de budget ou du temps pour une tâche donnée. Il est obtenu par l’opération suivante :

  (dépassement de budget / budget prévu) x 100

  (dépassement de durée / durée prévue) x 100

VI.B.2. KPI de coût

Ces KPI vont donner des indicateurs quant à l'utilisation du budget du projet. Deux types de KPI seront utilisés pour ce projet :

- l'**écart de coût du projet** : ce KPI permettra de mesurer si le coût effectif du projet est différent du coût estimé initialement. Il est obtenu par l'opération suivante :

  (coût réel – coût prévisionnel) / coût prévisionnel

- le **coût actuel du projet** : ce KPI permettra simplement d'estimer ce que le projet à coûter et il suffira donc d'ajouter toutes les dépenses liées au projet jusqu'à maintenant.

VI.B.3. KPI de ressources

Ce domaine de KPI utilisera un seul et unique indicateur :

- le **KPI de ressources** : cet indicateur permettra de calculer la productivité des ressources humaines travaillant sur le projet. Le *jour-homme* sera l'unité de mesure utilisée afin d'estimer la quantité de travail effectuée par une personne pendant une journée. Il est obtenu par l'opération suivante :

  (nombre de jours-hommes consacrés) x (% de réalisation de la tâche)

Il suffira alors de comparer le résultat obtenu au nombre de jours-hommes prévu pour atteindre ce même pourcentage de réalisation de la tâche. Ainsi, cela vous indiquera l'avance ou le retard des délais par rapport au planning.

VI.B.4. KPI d'efficacité

Ce type d'indicateurs sera représenté par deux KPI :

- le **pourcentage des tâches réalisées** : ce KPI permettra d'obtenir une vue d'ensemble des performances du projet en donnant le pourcentage des tâches réalisées.

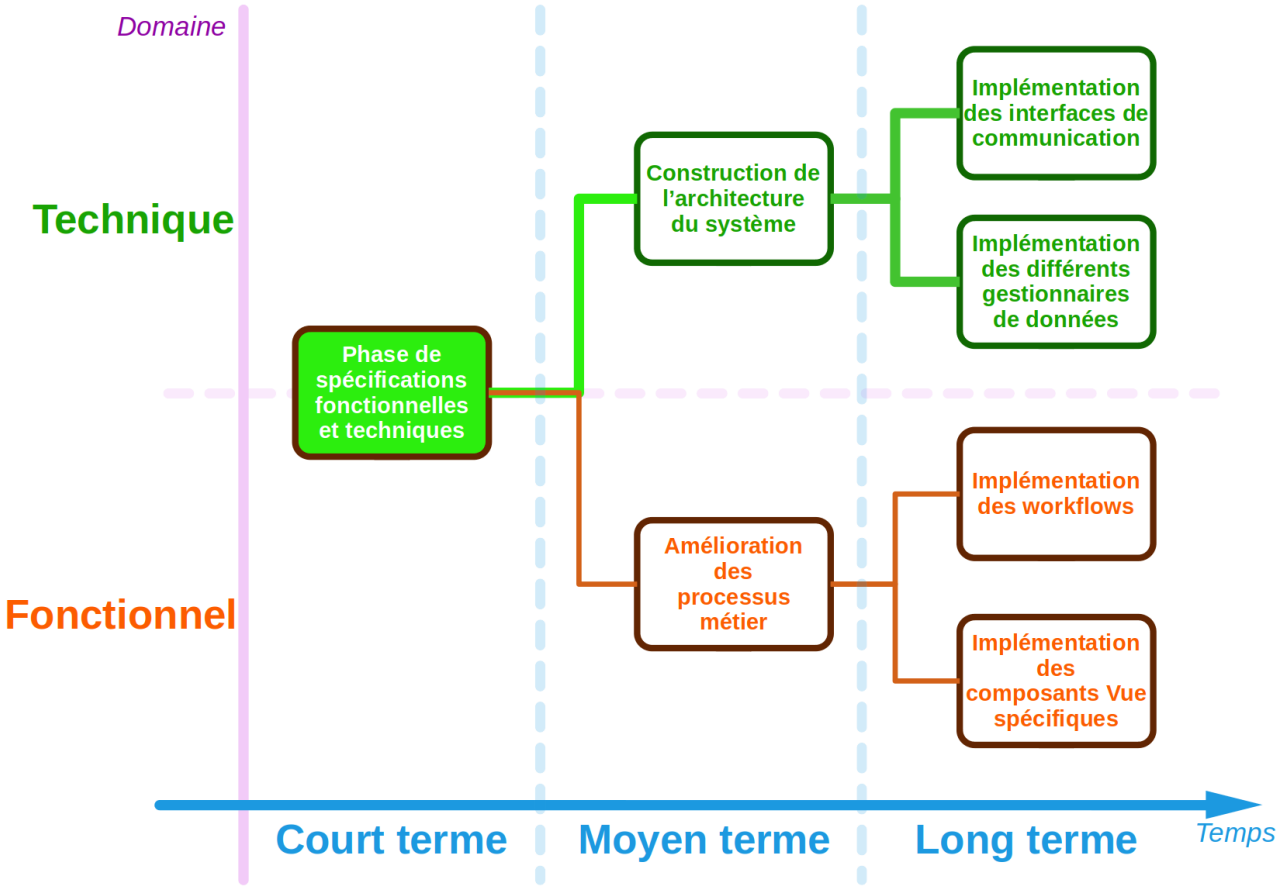
- les **heures de travail** : ce KPI comparera le nombre d'heures de travail planifiées pour le projet, au temps réellement passé sur le projet. Si la quantité d'heures passées dépasse les prévisions de durée initiale, il sera temps de réévaluer la durée totale du projet.

VII. Plan et calendrier du projet d'architecture

Relativement aux sections précédentes, cette étude préconise d’adopter une roadmap organiser en deux axes différents :

- un axe de domaine d’approche :
  - domaine fonctionnelle,
  - domaine technique.
- Un axe temporel selon trois périodes différentes :
  - **court terme** : période d’environ 3 semaines ;
  - **moyen terme** : période d’environ 3 mois ;
  - **long terme** :période d’environ 3 trimestre.

Ainsi, le roadmap pourra être représentée selon les deux axes sus-mentionnés, tels que schméatisé ci-dessous :




MedHead