[chapitre à supprimer]

Modèle de Dossier d'Architecture Projet

https://github.com/bflorat/modele-dap

Mises à jour du modèle			
Version	***Date***	***Auteurs***	***Changements***
1.0	*06 jan 2017*	*Bertrand FLORAT*	*Version initiale*

Légende
Ceci est une explication (à supprimer)
Ceci est un exemple (à supprimer)
Ceci est une notice à destination du lecteur final et pouvant être conservée

Informations diverses

- *ce modèle s'adresse à des architectes ;*
- *tous les diagrammes de ce modèle ont été générés avec l'excellent outil PlantUML (http://plantuml.com/). Les sources des diagrammes sont inclues dans le document et visibles avec l'extension PlantUML pour LibreOffice.*

Conseils d'utilisation de ce modèle :

- *les retours, remarques et suggestions sur ce modèle sont les bienvenues. Utiliser de préférence les tickets github à https://github.com/bflorat/modele-dap ;*
- *rester bref, tout doit être utile. Pas d'explication bateau type 'ceci est l'introduction', pas de redites d'autres documents, de l'historique de l'entreprise ou de concepts vagues ;*
- *Un lecteur doit comprendre le fonctionnement et les contraintes du projet sans être noyé de détails. Le document doit rester maintenable et donc à jour ;*
- *si le projet s'inscrit dans le fonctionnement nominal de l'organisation, ne pas se répéter et se référencer à un document commun (à faire figurer au chapitre 1.1).*
- *Si un chapitre n'est pas applicable, ne pas le laisser vide mais simplement mentionner « N/A » pour que le lecteur sache que le sujet a été traité. Ce modèle se veut suffisamment complet pour couvrir la plupart des projets.* ***Il est donc normal que de nombreux chapitres ne soient pas applicables dans leur contexte****. Un bon dossier d'architecture suivant ce modèle devrait faire entre 50 et 100 pages.*
- *que ne trouve-t-on* ***pas*** *dans ce document ?*
  - *la conception détaillée (diagrammes UML de classes, de séquences ...) sauf pour présenter un pattern général spécifique au projet ;*
  - *des éléments d'études (SWOT, scénarios…) : les choix doivent déjà avoir été faits ;*
  - *l'urbanisation du SI (nous sommes ici au niveau d'un projet) ;*
  - *des détails techniques (IP, logins) pouvant compromettre la sécurité du projet ;*
  - *le détail des environnements autres que la production (recette, développement...). Ces derniers sont en général trop mouvants pour figurer dans ce dossier et gagneront à plutôt être documentés par l'intégrateur dans d'autres dossiers, fiches ou mieux : wikis.*

Bibliographie :

- *Performance des architectures IT - 2e ed. - Pascal Grojean*
- *[GOF] Design Patterns: Elements of Reusable Object-Oriented Software de Erich Gamma, Richard Helm, Ralph Johnson et John Vlissides*
- *Christophe Longépé « Le projet d'Urbanisation du SI »*
- *Topologie de cluster HA : http://www.ibm.com/support/knowledgecenter/SSALK7_7.5.1/com.ibm.mbs.doc/gp_highavail/c_cluster_config_models.html*

Projet XXX

Dossier d’Architecture Projet

APPROBATION DU DOCUMENT			
Organisation	**Nom (fonction)**	**Date**	**Visa**

DIFFUSION	
Destinataire	**Organisation**

SUIVI DES MODIFICATIONS			
Version	**Date**	**Auteurs**	**Changements**

Sommaire

1. INTRODUCTION

1.1. DOCUMENTATION DE RÉFÉRENCE

N°	Version	Titre/URL du document	Détail
1		*https://references.modernisation.gouv.fr/rgaa-accessibilite/#menu*	*RGAA*

1.2. GLOSSAIRE

Note : par concis de concision, nous ne détaillons ici que les termes et acronymes spécifiques au projet. Pour les définitions générales, veuillez vous référer au glossaire d'entreprise.

1.2.1 Terminologie

Concept	Définition du concept
partition	*Pour un batch JSR352, une partition est l'exécution au sein d'un thread d'un sous ensemble des données d'entrée d'une étape du batch. La sélection des données se fait via une clé de partitionnement fournie par un* `PartitionMapper`*.*

1.2.2 Sigles et Acronymes

Sigles et Acronymes	Définition
JTA	*Java Transaction API*

2. POINTS À TRAITER

Lister ici les points en attente de décision ou d'études complémentaires (ils doivent être exceptionnels, sinon le DA est rédigé trop tôt)

N°	Détail
1	*Le choix technique de la solution d'ESB reste soumise à étude complémentaires*
2	*En fonction de l'avancement du projet YYY, ce composant pourrait se baser dessus ou s'accoster à l'ancien composant ZZZ*

3. CONTEXTE GÉNÉRAL DU PROJET

3.1. OBJECTIFS DU PROJET

Rappeler ici les objectifs du projet et mettre en évidence ceux qui sont structurants pour l'architecture.

Exemple 1 : Ce projet doit permettre la dématérialisation des factures reçues de nos fournisseurs et une consultation aisée de ces documents par les services comptables.

Exemple 2 : ce projet est la réécriture en technologies Web de l'application Cobol XXX.

3.2. ENJEUX DU PROJET

Larousse : « enjeu : Ce que l'on peut gagner ou perdre dans une entreprise quelconque »

Détailler ici ce qu'on gagne à faire le projet ou ce qu'on risque de perdre si on ne le fait pas. Ne pas reprendre les objectifs.

Exemple 1 : 1M€ seront économisés grâce à ce projet. Un retard de 6 mois coûterait 100K€.

Exemple 2 : Notre SI sera rattachée au système central en février prochain. Dans ce cadre, ce projet doit permettre de s'aligner sur les contrats de services attendus par les applications clientes du programme XXX. Ce projet est donc sur le chemin critique du programme.

3.3. PHASES ET ÉTAPES DU PROJET

Décrire les différentes phases du projet et leur couvertures fonctionnelles respectives, notamment les phases d'initialisation, migration, cible ainsi que les différents paliers du projet.

Décrire la trajectoire du projet et les éventuels composants transitoires.

Exemple 1 : un lot 1 prévu pour janvier 20XX proposera les fonctionnalités X, Y et Z. Un lot 2 finalisera le produit.

Exemple 2 : ce projet sera développé en Scrum suivant des sprints de 3 semaines. Actuellement, 10 sprints sont planifiés et devraient couvrir 80% des stories.

3.4. DESCRIPTION GÉNÉRALE DU PROJET

3.4.1 Description du projet

Décrire brièvement l'apport fonctionnel du projet sans reprendre les objectifs.

Exemple : Le projet XXX est l'un des composants principaux du programme YYY. Il s'adosse sur les référentiels Personne et Facturation pour enrichir le CMS en données clients temps réel.

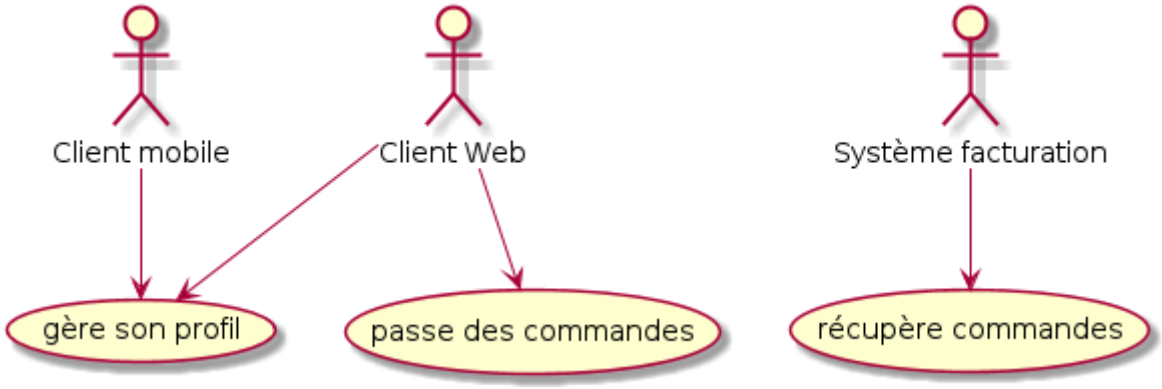
3.4.2 Positionnement dans le SI

Si le SI est urbanisé, reprendre le POS et préciser le bloc concerné

3.5. ACTEURS

3.5.1 Vision d'ensemble

Proposer ici un schéma (de use case de préférence) présentant de façon exhaustive les acteurs du projet. Un acteur peut être un humain ou un système informatique. Ne pas détailler à ce stade les interactions des acteurs avec le système, nous voulons simplement les lister.



3.5.2 Acteurs internes

On entend par 'internes' les acteurs appartenant à l'organisation. Il peut s'agir d'humains ou de composants applicatifs.

Acteur	Description	Population	Fréquence appels
Système de facturation	*Le système de facturation de l'application YYY*	*N/A*	*Appels en masse la nuit*
Employés facturation	*Les employés du service facturation*	*100*	*Plusieurs fois par minute sur les heures de bureau*

3.5.3 Acteurs externes

Acteur	Description	Population	Fréquence appels
Client Web	*Un client particulier depuis un PC*	*Max 1M*	*10 appels à l'IHM par session, une session par jour et par acteur*
Client mobile	*n client particulier depuis un mobile*	*Max 2M*	*5 appels de service par session, une session par jour et par acteur*

4. CONTRAINTES

On rappelle ici les contraintes auxquelles l'architecture de ce projet est soumise, par exemple les contraintes d'exploitation qui influent sur la disponibilité tenable, les normes à respecter… Bref, il s'agit des règles de niveau SI fixées par le DSI et les architectes des études et des opérations et auxquelles les projets doivent se soumettre. En général, ce chapitre fera simplement référence à des documents déjà existant et sera donc court.

4.1. COÛT

Donner l'enveloppe du projet (coût financier, charge en JH) pour justifier les choix techniques (méthodologie de développement, choix d'outils Open Source ou propriétaire...)

Dans certaines organisations, ce volet figure dans des études de choix préalables et pas dans ce dossier.

Les coûts peuvent être globalement catégorisés en :

- *Coûts de réalisation (études, développement, tests, intégration principalement)*
- *MCO (Maintien en Condition Opérationnelle) : correction des bugs, coût de support des éditeurs, coûts d'exploitation spécifiques...*
- *Maintenance évolutive (nouvelles fonctionnalités)*
- *Coûts de fonctionnement : frais de mise sous plis, de cloud, de sauvegarde externe ...*

Exemple : le coût de la version initiale du projet ne devra dépasser 100K€ en licences et 300JH de développement, tests et intégration dont la moitié en prestation externe.

Exemple 2 : le coût de la MCO annuel (support des éditeurs + maintenance corrective + coûts de fonctionnement) ne dépassera pas 300K€ par an.

Exemple 3 : Les frais de mise sous pli ne devront pas dépasser 2K€ / mois.

Exemple 4 de projet PaaS sous App Engine : les frais d'exécution de l'application ne dépasseront pas 5K€ de frais Google / mois.

4.2. FILIÈRE TECHNIQUE CHOISIE

Donner les technologies choisies parmi les technologies au catalogue de l'organisation. S'il existe des écarts au catalogue, le préciser et le justifier.

Exemple : ce projet suit le profil P3 : "Application Web Spring" avec utilisation exceptionnelle de la librairie JasperReport.

4.3. HÉBERGEMENT

- *Où sera hébergé ce projet ? (datacenter on premise ? Cloud ? IaaS ? PaaS ? …)*
- *Qui administrera cette application ? Administré en interne ? Sous-traité ? Pas d'administration (PaaS) … ?*

Exemple : Ce projet sera hébergé en interne dans le datacenter de Nantes Centre (seul à assurer la disponibilité de service exigée) et il sera administré par l'équipe XXX de Lyon.

4.4. NORMES

Lister les normes SI que le projet suit et les éventuels écarts et leur justification.

Exemple : ce projet respecte l'ensemble des normes d'architecture et de sécurité de l'organisation XXX mis à part :

- *l'utilisation d'hyperliens de validation dans les e-mails envoyés aux internautes (dérogation validée)*

4.5. DISPONIBILITÉ

Les éléments ici fournis pourront servir de base à l'OLA (Operationnal Level Agreement).

4.5.1 Présence nominale des exploitants et astreintes

Heures de présence des exploitants, astreintes normales sur le SI...

Comme toute application hébergée au datacenter XXX, le projet disposera de la présence d'exploitants de 7h à 20h jours ouvrés. Aucune astreinte n'est prévue.

4.5.2 Durées d'intervention externes

Lister ici les durées d'intervention des prestataires matériels, logiciels, électricité, telecom...

Le remplacement de support matériel IBM sur les lames BladeCenter est assuré en 4h de 8h à 17h, jours ouvrés uniquement.

4.5.3 Interruptions programmées

Lister ici les interruptions à prévoir pour raison d'exploitation (et non sur incident).

Exemple : suite aux mises à jour de sécurité de certains packages RPM (kernel, libc…), les serveurs RHEL seront redémarrés automatiquement la nuit du mercredi suivant la mise à jour. Ceci entraînera une indisponibilité de 5 mins en moyenne 4 fois par an.

4.5.4 Niveau de service du datacenter

Donner ici le niveau de sécurité du datacenter selon l'échelle Uptime Institute (Tier de I à IV).

La plupart des datacenters sont de niveau I ou II dans le meilleur des cas.

Choix	Tier	Caractéristiques	Taux de disponibilité	Indisponibilité statistique annuelle	Maintenance à chaud possible	Tolérance aux pannes
	Tier I	Non redondant	unité 99,671 %	28,8 h	Non	Non
X	Tier II	Redondance partielle	99,749 %	22 h	Non	Non
	Tier III	Maintenabilité	99,982 %	1,6 h	Oui	Non
	Tier IV	Tolérance aux pannes	99,995 %	0,4 h	Oui	Oui

Source : wikipedia

4.5.5 Plan de Reprise ou de Continuité d'Activité (PRA / PCA)

PRA comme PCA répondent à un risque de catastrophe sur le SI (catastrophe naturelle, accident industriel, incendie...).

Un PCA permet de poursuivre les activités critiques de l'organisation (en général dans un mode dégradé) sans interruption notable, voir norme ISO 22301. Le concept est réservé aux organisations très matures car il exige des dispositifs techniques coûteux et complexes (réplication des données au fil de l'eau par exemple).

Un PRA permet de reprendre l'activité suite à une catastrophe après une certaine durée de restauration. Il exige au minium un doublement du datacenter.

Décrire entre autres :

- *Les matériels redondés dans le second datacenter, nombre de serveurs de spare, capacité du datacenter de secours par rapport au datacenter nominal ;*
- *Pour un PRA les dispositifs de restauration (OS, données, applications) prévues ;*
- *Pour un PCA les dispositifs de réplication de données (synchrone ? fil de l'eau ? Combien de transactions peuvent-être perdues ?) ;*
- *Pour un PRA, donner le Recovery Time Objective (durée maximale admissible de rétablissement en h) et Recovery Point Objective (durée maximale admissible de données perdues en h) de l'organisation ;*
- *Présenter la politique de failback : doit-on rebasculer vers le premier datacenter ? Comment ?*
- *Comment sont organisés les tests de bascule à blanc ? Avec quelle fréquence ?*

4.6. SÉCURITÉ

Présenter les normes de sécurité de l'organisation qui ont dicté l'architecture de la solution, comme les règles d'échanges inter-zones.

Exemple : Comme exigé par la norme SI XXX, aucun appel direct depuis les composants front-office exposés sur Internet ne pourra se faire vers un composant back-office situé dans l'intranet. Seuls les appels dans l'autre sens sont autorisés. Cette contrainte a dicté l'architecture asynchrone à base de files JMS situées dans le FO et accédées par des EJB MDB en BO.

5. EXIGENCES NON FONCTIONNELLES

Contrairement aux contraintes qui fixaient le cadre auquel tout projet devait se conformer, les exigences non fonctionnelles sont données par les commendataires du projet (MOA en général). Prévoir des interviews pour les déterminer. Si certaines exigences ne sont pas réalistes, le mentionner au chapitre 2.

5.1. PLAGES DE FONCTIONNEMENT

On liste ici les plages de fonctionnement principales (ne pas trop détailler, ce n'est pas un plan de production).

Les informations données ici serviront d'entrants au SLA du projet.

No plage	Détail	Intervalle de temps
1	*Ouverture Intranet aux employés*	*De 8H00-19H30, 5J/7 jours ouvrés*
2	*Plage batch*	*De 21h00 à 7h00*
3	*Ouverture Internet aux usagers*	*24 / 7 / 365*

5.2. TEMPS DE RÉPONSE

5.2.1 Temps de Réponse des IHM

Si les clients accèdent au système en WAN (Internet, VPN, LS …), préciser que les exigences de TR sont données hors transit réseau car il est impossible de s'engager sur la latence et le débit de ce type de client. Dans le cas d'accès LAN, il est préférable d'intégrer le temps réseau, d'autant que les outils de test de charge vont déjà le prendre en compte.

Les objectifs de TR sont toujours donnés avec une tolérance statistique (90eme centile par exemple) car la réalité montre que le TR est très fluctuant car affecté par un grand nombre de facteurs.

Inutile de multiplier les types de sollicitations (en fonction de la complexité de l'écran par exemple , ce type de critère n'a plus grand sens aujourd'hui, particulièrement pour une application RIA).

Types de sollicitations :

Type de sollicitation	Bon niveau	Niveau moyen	Niveau insuffisant
Chargement d'une page	*< 0,5 s*	*< 1 s*	*> 2 s*
Opération métier	*< 2 s*	*< 4 s*	*> 6 s*
Édition, Export, Génération	*< 3 s*	*< 6 s*	*> 15 s*

Acceptabilité des TR :

Le niveau de respect des exigences de temps de réponse est :

Bon si :	• *au moins 90 % des temps de réponse sont bons* • *au plus 2% des temps de réponse sont insuffisants*
Acceptable si :	• *au moins 80 % des temps de réponse sont bons* • *au plus 5 % des temps de réponse sont insuffisants*

En dehors de ces intervalles, l'application devra être optimisée et repasser en recette puis être soumise à nouveau aux tests de charge.

5.2.2 Durée d'exécution des batchs

Préciser ici dans quel intervalle de temps les traitements par lot doivent s'exécuter.

Exemple 1 : La fin de l'exécution des batchs étant un pré-requis à l'ouverture du TP, ces premiers doivent impérativement se terminer avant la fin de la plage batch définie en 5.1.

Exemple 2 : le batch mensuel XXX de consolidation des comptes doit s'exécuter en moins de 4 J.

Exemple 3 : les batchs et les IHM pouvant fonctionner en concurrence, il n'y a pas de contrainte majeure à la durée d'exécution des batchs mais pour assurer une optimisation de l'infrastructure matérielle, on favorisera la nuit pendant laquelle les sollicitations IHM sont moins nombreuses.

5.3. SÉCURITÉ

Présenter les exigences, pas les solutions. Celles-ci seront détaillées au chapitre Sécurité.

Pour les projets particulièrement sensibles, prévoir un dossier d'analyse de risque. Pour cela, utiliser par exemple la méthode EBIOS (Expression des Besoins et Identification des Objectifs de Sécurité à https://www.ssi.gouv.fr/guide/ebios-2010-expression-des-besoins-et-identification-des-objectifs-de-securite/)

Les exigences les plus importantes telles que définies dans la norme ISO/CEI 27001 sont la triade Disponibilité, Intégrité et Confidentialité.

5.3.1 Disponibilité

Nous listons ici les exigences de disponibilité. Les mesures techniques permettant de les atteindre seront données dans l'architecture technique de la solution.

Les informations données ici serviront d'entrants au SLA du projet.

Attention à bien cadrer ces exigences car une MOA a en général tendance à demander une disponibilité très élevée sans toujours se rendre compte de ce que cela implique. Le coût et la complexité de la solution augmente exponentiellement avec le niveau de disponibilité exigé. L'architecture physique, technique voire logicielle change complètement en fonction du besoin de disponibilité (clusters d'intergiciels voire de bases de données, redondances matériels coûteuses, architecture asynchrone, caches de session, failover ...). Ne pas oublier également les coûts d'astreinte très importants si les exigences sont très élevées. De la pédagogie et un devis permettent en général de modérer les exigences.

A titre d'ordre de grandeur, la haute disponibilité (HA) commence à deux neufs (99%) ;

5.3.1.1 Disponibilité exigée par plage de fonctionnement

La liste des plages de fonctionnement est disponible en 5.1.

La disponibilité exigée ici devra être en cohérence avec les contraintes de disponibilités du SI données en 4.5.

No Plage	Disponibilité attendue	Indisponibilité programmée	Indisponibilité non programmée
1	*99.72 %*	*0 %*	*0.28% (2 h/mois)*
2	*94.72 %*	*5% d'interruption programmée - (8,2 h / semaine pour sauvegarde à froid) - 0.2 h / semaine en moyenne pour mise à jour système*	*0.28% (2 h/mois) d'interruption non programmée*

5.3.1.2 Mode dégradé acceptable

Préciser l'impact maximal accepté sur les temps de réponse lors d'une panne

Exemple 1 (perte d'un nœud d'un cluster) : Les serveurs devront être dimensionnés pour être chacun en mesure d'assurer le fonctionnement de l'application et de limiter l'augmentation des temps de réponse à 20 %.

Préciser les modes dégradés applicatifs envisagés.

Exemple2 (perte d'un service) : Le site monsite.com devra pouvoir continuer à prendre les commandes en l'absence du service de gestion de l'historique des commandes.

5.3.2 Robustesse

La robustesse du système indique sa capacité à ne pas produire d'erreurs lors d'événement exceptionnels comme une surcharge ou la panne d'un des composants.

Cette robustesse s'exprime en valeur absolue par unité de temps : nombre d'erreurs (techniques) par mois, nombre de messages perdus par an...

Attention à ne pas être trop exigent sur ce point car une grande robustesse peut impliquer la mise en place de système à tolérance de panne complexes, coûteux et pouvant aller à l'encontre des capacités de montée en charge voire de disponibilité (pour plus de détail, voir 6.1).

Exemple 1 : pas plus de 0.001% de requêtes en erreur

Exemple 2 : l'utilisateur ne devra pas perdre son panier d'achat même en cas de panne
-> attention, ce type d'exigence impacte l'architecture en profondeur, voir 6.1

Exemple 3 : le système devra pouvoir tenir une charge trois fois supérieure à la charge moyenne (voir 5.9.2) avec un temps de réponse de moins de 10 secondes au 95éme centile.

5.3.3 Sauvegardes

Donner ici le Recovery Point Objective (RPO) du projet. Il peut être utile de restaurer suite (par exemple) à :

- *une perte de données matérielle (peu probable avec des systèmes de redondance type RAID)*
- *une fausse manipulation d'un power user ou d'un administrateur*
- *un bug applicatif*
- *une destruction de donnée volontaire (attaque de type ransomware par exemple).*

Exemple : on ne doit pas pouvoir perdre plus d'une journée (de 08h à 18h00) de données applicatives

5.3.4 Intégrité

L'intégrité concerne la durabilité, la justesse et le niveau de confiance dans les données du projet. Gérer l'intégrité des données consiste à vérifier qu'elle ne peuvent être altérées ou supprimées (involontairement, suite à un crash disque par exemple) ou volontairement dans le cadre d'une attaque de type man in the middle ou par une personne s'étant octroyé des droits indus par exemple.

Attention à ne pas multiplier les classes de données. Il est possible de ne définir qu'une seule classe de donnée pour l'ensemble du projet (cas courant). Les dispositifs permettant de répondre à ces exigences seront définies au chapitre 6.2.

L'intégrité exigée pour chaque classe de donnée du projet est la suivante :

Classe de données	Niveau « Non intègre » : la donnée peut ne pas être intègre	Niveau « Détectable » : la donnée peut ne pas être intègre si l'altération est identifiée	Niveau « Maîtrisé » : la donnée peut ne pas être intègre si l'altération est identifiée et corrigée en moins d'une heure	Niveau « Intègre » : la donnée doit être rigoureusement intègre
Données de la base métier				*X*
Données archivées		*X*		
Données calculées XXX			*X*	
Silo NoSQL des données Big Data avant consolidation	*X*			
Sources du projet				*X*
Avis 'imposition en PDF				*X*

5.3.5 Confidentialité

« La confidentialité est le fait de s'assurer que l'information n'est accessible qu'à ceux dont l'accès est autorisé. » (ISO 27018) .

Attention à ne pas multiplier les classes de données. Il est possible de ne définir qu'une classe de donnée pour l'ensemble du projet (cas courant). Les dispositifs permettant de répondre à ces exigences seront définies au chapitre 6.2.

La confidentialité exigée pour chaque classe de donnée du projet est la suivante :

Classe de données	Niveau « Public » : tout le monde accéder à la donnée	Niveau « Limité » : la donné n'est accessible qu'aux personnes habilitées	Niveau « Réservé » : la donné n'est accessible qu'au personnel interne habilité	Niveau « Privé » : la donnée n'est visible que par l'intéressé(e)
Contenu éditorial du projet	*X*			
Profil du compte du site Web		*X*		
Historique du compte			*X*	
Logs des activités de l'internaute			*X*	
Données du projet RH "aides sociales aux employés"				*X*

5.3.6 Identification

L'identification est l'ensemble des dispositifs permettant de différentier un utilisateur d'un autre (sans vérifier qu'il est bien celui qu'il prêtent être).

Exemple 1 : Un utilisateur ne peut avoir qu'un identifiant et un identifiant ne peut être partagé par plusieurs utilisateurs.

Exemple 2 : l'identité d'un internaute fera l'objet d'un test d'existence avant tout appel de service.

Exemple 3 : un ID est non supprimable, non modifiable et non réutilisable

Le cas échéant, préciser également si le projet a besoin de fédération d'identité.

La fédération d'identité est l'utilisation d'une même identité géré par un « identity provider » depuis plusieurs entités différentes . Par exemple, France Connect très utilisé par les administrations permet par exemple de réutiliser son compte La Poste pour se loguer sur le compte de la CNAM. Il y a aussi les « Connect with [Google|Twitter|...] » en technologie OpenId Connect. Contrairement au SSO, il n'assure pas un login automatique à une application mais permet simplement de réutiliser le même login/mot de passe.

Exemple : L'identification et l'authentification seront externalisés à un fournisseur d'identité pour simplifier la gestion de la sécurité et réduire les coûts de développement et d'exploitation.

5.3.7 Authentification

L'authentification est des dispositifs permettant de vérifier la cohérence entre l'identité de l'utilisateur et la personne physique se connectant.

L'authentification peut être à un ou plusieurs facteurs (dans ce dernier cas, on parle d'authentification forte). Ces facteurs peuvent être : quelque chose que l'on connaît (mot de passe), quelque chose qu'on est (biométrie), quelque chose qu'on a (token, générateur de mot de passe unique, pièce d'identité avec photo...).

Penser à décrire le système d'authentification une fois inscrit mais également lors de l'inscription.

La délégation d'authentification s'appuie sur une technologie de fédération d'identité pour authentifier l'utilisateur.

Il est bien sûr possible d'ajouter des facteurs d'authentification spécifiques à votre organisation au besoin.

Les facteurs d'authentification requis en fonction des situations sont (on peut exiger plusieurs occurrences du même facteur, utiliser autant de croix) :

Cas d'authentification	Mot de passe respectant la politique de mot de passe XXX	Clé publique ssh connue	OTP par Token	Biométrie	Carte magnétique	Connaissance de données métier	E-mail d'acticvation	Délégation d'authentification
Utilisateur déjà inscrit	*X*							
Création d'un compte						*X X*	*X*	
Modification du mot de passe	*X*						*X*	
Accès au datacenter				*X*	*X*			
Accès aux logs		*X*						
Ajout d'un bénéficiaire de virement	*X*		*X*					
Application mobile YYY								*X*

5.3.8 SSO, SLO

Décrire les besoin en terme de Single Sign On et Single Log Out.

*Nous entendons ici SSO dans son sens complet : une authentification **automatique** à une application d'un utilisateur déjà authentifié depuis une autre application du même domaine de confiance.*

Attention, la mise en place de SSO sur un projet est souvent complexe, surtout si l'infrastructure (ID provider…) n'existe pas encore et nécessite souvent une adaptation des applications. L'exigence en SSO doit être justifiée. Une application périphérique ou un outil rarement utilisé n'ont en général pas besoin de SSO (une simple authentification centralisée au sein d'un annuaire LDAP suffit souvent). Attention également à évaluer l'impact qu'aurait une authentification faible (mauvais mot de passe par exemple) sur la sécurité de l'ensemble du SI.

Exemple 1 : aucun SSO n'est exigé puisque toutes les IHM du projet sont exposées au sein d'un portail JSR352 qui gère déjà l'authentification.

Exemple 2 : aucun besoin de SSO ou SLO n'est identifié

Exemple 3 : cette application Web métier devra fournir une authentification unique mutualisée avec celle des autres applications de l'intranet : une fois authentifié sur l'une des applications, l'agent ne doit pas avoir à se reconnecter (sauf bien sûr suite à l'expiration de sa session). De même, une déconnexion depuis l'une des applications doit assurer la déconnexion de toutes les applications de l'intranet .

5.3.9 Preuve

Lister ici les données à conserver car pouvant servir de preuve en cas de contestation ainsi que leur durée de rétention :

Donnée	Objectif	Durée de rétention
Log complet (IP, heure GMT, détail) des commandes passées sur le site	*Prouver que la commande a bien été passée*	*1 an*
Date et contenu du mail de confirmation	*Prouver que le mail de confirmation a bien été envoyé*	*2 ans*
Contrat d'assurance signé et numérisé en PDF	*Prouver que le contrat a bien été signé*	*5 ans*
Avis d'imposition primitif avec signature numérique	*Conserver le montant et de l'impôt.*	*5 ans*

5.3.10 Non répudiation

Lister ici les actions métiers possédant une exigence de non-répudiation, c'est à dire un dispositif permettant de rendre impossible la remise en cause d'un contrat en prouvant l'identité des deux parties et l'intégrité du document par signature numérique comme décrit dans la loi n°2000-230 du 13 mars 2000.

Donnée signée	Origine du certificat client	Origine du certificat serveur
Déclaration d'impôt sur le revenu (données X, Y et Z)	*PKI de l'administration fiscale*	*Verisign*

5.3.11 Anonymat et vie privée

Lister les contraintes d'anonymat et de vie privée légale (exigée par la CNIL) + les contraintes supplémentaires de l'organisation. Voir aussi https://www.cnil.fr/fr/comprendre-vos-obligations .

Exemple 1 : Aucun consolidation de donnée de pourra être faite entre les données du domaine PERSONNE et du domaine SANTE.

Exemple 2 : Par soucis de sécurité en cas d'intrusion informatique, certaines données des personnes seront expurgées avant réplication vers la zone publique : le taux de cholestérol et le poids.

Exemple 3 : aucune donnée raciale ou d'orientation sexuelle ne pourra être stockée sous quelque forme que ce soit dans le SI.

Exemple 4 : Les données OpenData issues du domaine « logement » ne contiendront que des données consolidées de niveau commune, pas plus précise.

Exemple 5 : En application de la directive européenne « paquet telecom », un bandeau devra informer l'usager de la présence de cookies.

Exemple 6 : Les données de santé seront hautement sécurisées (chiffrement fort) et tout accès tracé.

5.3.12 Autorisations

Une autorisation (ou habilitation) permet de donner l"accès à une fonction applicative (= « privilège », « permission ») à un utilisateur ou un groupe d'utilisateur.

Par exemple :

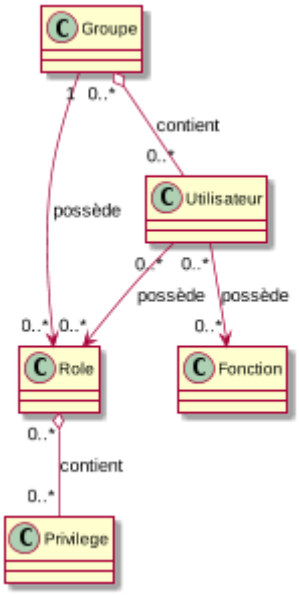
- *Fonction F1 : faire un virement inter-bancaire*
- *Fonction H2 : voir l'historique de son compte*
- *Fonction E2 : supprimer un utilisateur*

Attention à ne pas multiplier le nombre de fonctions et de rôles pour éviter une explosion combinatoire et des coûts de gestion associés.

Pour simplifier la gestion des autorisations par factorisation, on peut :

- *regrouper les utilisateurs dans des groupes (G_chef_service = Untel, Lui, Elle)*
- *regrouper les fonctions dans des rôles (R_Administrateur, R_banquier_niv1, R_chef_service…) qu'on peut affecter à une personne ou un groupe*

Exemple de gestion complète des autorisations :



Penser à spécifier les rôles des éventuels pseudos-utilisateurs comme :

- *@anonyme : les personnes non connectées*
- *@connecte : les personnes connectées*

Préciser si le projet doit utiliser de la délégation d'autorisation (type OAuth2) et si oui, le projet est-il fournisseur ou consommateur d'autorisations ? Quelles sont les autorisations concernées ?

Exemple 1 : les personnes non connectées auront accès à tous les privilèges en lecture seule, c'est à dire : ….

Exemple 2 : Le projet s'appuiera sur une gestion des autorisations matricielle de type rôles → groupes et utilisateurs . Le détail des autorisations sera donnée dans les SFD.

Groupe / utilisateur	Rôle « suppression »	Rôle administration	Rôle consultation des données de base
Groupe g_usagers			*X*
@anonyme			
Groupe g_admin	*X*	*X*	*X*
Utilisateur XXX	*X*		*X*

Exemple 3 : L'application aura besoin de façon obligatoire du carnet d'adresse Facebook de l'usager

5.3.13 Traçabilité, auditabilité

Lister ici les besoins en traces permettant de détecter par exemple :

- *un usage abusif des applications Back Office par des employés*
- *des intrusions informatiques*
- *des modifications de données*

Les traces sont des données nominatives et complètes pour permettre l'audit. Elles sont donc elle-même sensibles et nécessitent souvent un bon niveau de confidentialité (voir 5.3.5)

Différentier :

- *les traces métier (bilan d'un acte de gestion complet comme « l'agent XXX a consulté le dossier de Mme YYY »)*
- *et les traces applicatives (logs) comme dans un fichier de log : « [INFO] 2016/12/23 11:14 Appel par XXX du service consulter » qui sont de niveau technique.*

Pour les données les plus sensibles, penser à la traçabilité à deux niveaux (tracer la consultation des traces) pour éviter une traçabilité hiérarchique abusive.

La traçabilité des données des référentiels (PERSONNE typiquement) nécessite une historisation complète, ce qui est de toute façon une bonne pratique (voir par exemple Longépé « Le projet d'Urbanisation du SI », règles applicatives 1, 2 et 3). Pour cela, prévoir un MCD permettant d'ajouter un enregistrement à chaque changement de la donnée avec une date de modification et une date d'effet.

Exemple 1 : pour le module XXX, toute action métier (en mise à jour comme en consultation) devra faire l'objet d'une trace métier contenant a minima l'agent, la date et en cas de modification l'ancienne et la nouvelle valeur.

Exemple 2 : Toute intrusion dans le SI devra être détectée (dans la mesure du possible).

Exemple 3 : nous devons pouvoir reconstituer l'historique du dossier de tout patient à n'importe quelle date.

5.4. CONCURRENCE

Préciser ici si des composants internes ou externes au projet peuvent interférer.

Exemple 1 : Tous les composants de ce projet peuvent fonctionner en concurrence. En particulier, la concurrence batch/TP doit toujours être possible car les batchs devront pouvoir tourner de jour en cas de besoin de rattrapage

Exemple 2 : le batch X ne devra être lancé que si le batch Y s'est terminé correctement sous peine de corruption de données.

5.5. HÉBERGEMENT

Donner ici les exigences spécifiques d'hébergement.

Exemple : Étant donné le niveau de sécurité très élevé du projet, la solution devra être exploitée uniquement en interne par des agents assermentés et hors sous-traitance. Pour la même raison, les solutions de cloud sont exclues.

Exemple 2 : Étant donné le nombre d'appels très important de cette application vers le référentiel PERSONNE, elle sera colocalisé avec le projet PERSONNE dans le VLAN XXX.

5.6. ACCESSIBILITÉ

Ce projet doit-il être accessible aux non/mal voyants ? malentendants ? Si oui, quelle niveau d'accessibilité ? Se référer de préférence au Référentiel Général d'Accessibilité (RGAA) à https://references.modernisation.gouv.fr/rgaa-accessibilite/#menu qui vise un niveau WCAG 2.0 AA :

Les niveaux d'accessibilité WCAG2.0

Niveau	Objectif	Faisabilité	Exemple
A	Atteindre un niveau d'accessibilité minimal.	Critères de succès essentiels pouvant raisonnablement s'appliquer à toutes les ressources Web.	La couleur n'est pas le seul moyen visuel de véhiculer l'information.
AA	Améliorer le niveau d'accessibilité	Critères de succès supplémentaires pouvant raisonnablement s'appliquer à toutes les ressources Web.	Les textes de petite taille ont un ratio de contraste au moins égal à 4.5
AAA	Atteindre un niveau supérieur d'accessibilité.	Critères de succès ne s'appliquant pas à toutes les ressources Web.	Les textes de petite taille ont un ratio de contraste au moins égal à 7

Source : Wikipedia

Il existe d'autres normes d'accessibilté (WCAG, AccessiWeb …) . Attention à correctement évaluer le niveau visé :

- *atteindre un niveau d'accessibilité très élevé est coûteux, contraignant technologiquement et limite les possibilités ergonomiques. Il demande également de bonnes compétences (accessibilité, HTML5/CSS3 en particulier) et des profils rares ;*
- *pour un niveau AAA, le plus simple est en général de prévoir en général deux sites Web : un accessible et un classique ;*
- *la loi est de plus en plus stricte pour les administrations qui doivent respecter un niveau d'accessibilité suffisant (loi n°2005-102 du 11 février 2005 pour l'égalité des droits et des chances, la participation et la citoyenneté des personnes handicapées). « Tous les sites publics européens doivent atteindre le double A (AA) du W3C/WAI ».*

5.7. ERGONOMIE

5.7.1 Site Web adaptatif

Lister les contraintes d'affichage multi-support. Utiliser quand possible les fonctionnalités venants avec les frameworks modernes (type AngularJS ou ReactJS). Il y a plusieurs niveaux d'adaptation :

- *statique (largeur de page fixe)*
- *dynamique (redimensionnement automatique car tout est exprimé en %)*
- *adaptatif (les distances sont exprimées en unités sont la taille dépend du support)*
- *responsive (le contenu et son agencement dépend du support).*

Attention, un design responsive vient avec ses contraintes (duplication de code, augmentation du volume du site à télécharger par le client, complexité, plus de tests à prévoir…).

5.7.2 Charte ergonomique

En général, on se réfère ici à la charte ergonomique de l'organisme. Lister néanmoins d'éventuelles spécificités. Ne pas reprendre les contraintes d'accessibilité listées plus haut.

5.7.3 Spécificités sur les widgets

Des comportements ergonomiques très précis peuvent impacter assez fortement l'architecture et imposer une librairie de composants graphiques ou une autre. Il est fortement déconseillé de personnaliser des librairies existantes (coût de maintenance très élevé, grande complexité). Mieux vaut bien choisir sa librairie ou restreindre ses besoins.

Exemple 1 : les tableaux devront être triables suivant plusieurs colonnes.
Exemple 2 : de nombreux écrans seront pourvus d'accordéons

5.7.4 Navigateurs supportés

Préciser quels sont les navigateurs supportés s'il s'agit d'une IHM Web.

Attention : supporter d'anciens navigateur (IE en particulier) peut engendrer des surcoûts rédhibitoires (sauf à utiliser une librairie qui masque la complexité et en espérant qu'elle n'est pas buggée).

Dans tous les cas, il convient d'évaluer les surcoûts de tester sur plusieurs plate-formes. Il existe de bons outils (payants) comme Litmus ou Emailonacid permettant de générer un rendu des sites Web et des courriels HTML sur une combinatoire d'OS / type de lecteur (PC/tablette/mobile) /navigateur très vaste (de l'ordre de 50). Ce type de site est incontournable pour une application grand public.

Exemple 1 : L'application intranet XXX devra fonctionner sur les navigateurs habités en interne (cf norme XXX)

Exemple 2 : L'application YYY étant une application internet visant le public le plus large possible, y compris des équipements de pays en voie de développement. Il devra supporter Firefox 3+, IE 8+, Opera 6+.

Exemple 3 : L'application ZZZ vise le public le plus large doté de systèmes raisonnablement anciens et devra donc supporter : Firefox 6+, Chrome 8+, Opera 8+, IE 10, Edge.

5.7.5 Contrainte de déploiements et de mise à jour

5.7.5.1 Coté serveurs

Préciser ici comment l'application devra être déployée coté serveur et avec quelles contraintes.

Coté serveur, cela peut être :

- *Doit-on prévoir l'installation d'une image complète de l'OS par boot réseau (solution type Cobbler sous Linux ou Windows Deployment Toolkit sous Windows) ?*
- *Comment sont déployés les composants ? Sous forme de paquets ? Utilise-t-on un dépôt de paquets (type yum ou apt) ?*
- *Comment sont appliquées les mises jour ?*

5.7.5.2 Coté clients

Coté client, pour une application Web, cela peut être :

- *Prévoit-on un déploiement de type blue/green ? Si oui, préciser les proportions et la trajectoire visée*
- *si l'application est volumineuse (beaucoup de JS ou d'images par exemple), risque-t-on un impact sur le réseau ?*
- *Une mise en cache de proxy locaux est-elle à prévoir ?*
- *Des règles de firewall ou QoS sont-elles à prévoir ?*

Coté client, pour une application Java :

- *Quel version du JRE est nécessaire sur les clients ?*
- *S'il s'agit d'une application JavaWebStart, prévoit-on une mise à jour incrémentale des jars ?*

Coté client, pour une application client lourd :

- *Quel version de l'OS est supportée ?*
- *Si l'OS est Windows, l'installation passe-t-elle par un outil de déploiement (Novell ZENWorks par exemple) ? l'application vient-elle avec un installeur type Nullsoft ? Affecte-t-elle le système (variables d'environnements, base de registre…) ou est-elle en mode portable (simple zip) ?*
- *Si l'OS est Linux, l'application doit-elle fournie en tant que package ? Ce package est-il dans un dépôt d'entreprise pour être installé par un gestionnaire de paquet type yum ou apt ?*
- *Comment sont appliquées les mises jour ?*

5.8. INTERNATIONALISATION

Préciser les contraintes du projet en terme d'i18n : localisation des libellés, direction du texte, mise en page adaptable, code couleur spécifique, format de dates, affichage des séparateurs décimaux, etc.

Exemple 1 : L'IHM XXX sera traduite en 25 langues dont les langues asiatiques principales et l'arabe.

Exemple 2 : les formats de dates et autres champs de saisie devront être parfaitement localisés pour un confort maximal de l'utilisateur.

5.9. VOLUMÉTRIE

La volumétrie ici décrite permettra le dimensionnement initial de la solution. Il est crucial de récupérer un maximum d'informations issues de la production plutôt que des estimations car ces dernières se révèlent souvent loin de la réalité. C'est d'autant plus difficile s'il s'agit d'un projet sans précédent, prévoir alors une large marge.

Les informations données ici serviront d'entrants au SLA du projet.

5.9.1 Volumétrie statique

Lister ici les besoins en stockage de chaque composant une fois le projet rendu en vitesse de croisière (volumétrie à deux ans par exemple).

Prendre en compte :

- *la taille des bases de données;*
- *la taille des fichiers produits;*
- *la taille des files;*
- *la taille des logs ;*
- *...*

Ne pas prendre en compte :

- *le volume lié à la sauvegarde : elle est gérée par les opérationnels ;*
- *le volume des binaires (OS, intergiciels...) qui est à considérer par les opérationnels comme une volumétrie de base d'un serveur (le ticket d'entrée) et qui est de leur ressort ;*
- *les données archivées qui ne sont donc plus en ligne.*

Fournir également une estimation de l'augmentation annuelle en % du volume pour permettre aux opérationnels de commander ou réserver suffisamment de disque de façon proactive.

Pour les calculs de volumétrie, penser à prendre en compte les spécificités de l'encodage (nombre d'octets par caractère, par date, par valeur numérique...). Pour une base de donnée, prévoir l'espace occupé par les index et qui est très spécifique à chaque application. Une (mauvaise) estimation préliminaire est de doubler l'espace disque (à affiner ensuite).

N'estimer que les données dont la taille est non négligeable.

Exemple : Volumétrie statique du composant C :

Donnée	**Description**	**Taille unitaire**	**Nombre d'éléments à 2 ans**	**Taille totale**	**Augmentation annuelle**
Table `Article`	Les articles du catalogue	2Ko	100K	200 Mo	5 %
Table `Commande`	Les commandes clients	10Ko	3M	26.6 Go	10 %
Logs	Les logs applicatifs (niveau INFO)	200 o	300M	56 Go	0 % (archivage)

5.9.2 Volumétrie dynamique

Il s'agit ici d'estimer le nombre d'appels aux composants et donc le débit cible (en Tps = Transactions par seconde) que devra absorber chacun d'entre eux. Un système bien dimensionné devra présenter des temps de réponse moyen à peu près similaires en charge nominale et en pic.

Toujours estimer le "pic du pic", c'est à dire le moment où la charge sera maximale suite au cumul de tous les facteurs (par exemple pour un système de comptabilité : entre 14 et 15h d'un jour de

semaine de fin décembre). Ne pas considérer que la charge est constante mais prendre en compte :

- *les variations journalières. Pour une application de gestion avec des utilisateurs travaillant sur des heures de bureau, on observe en général des bosses du double de la charge moyenne à 8h-9h, 11h-12h et 14h-15h. Pour une application Internet grand public, ce sera plutôt en fin de soirée. Encore une fois, se baser sur des mesures d'applications similaires quand possible plutôt que des estimations.*
- *les éléments de saisonnalité. La plupart des métiers en possèdent : Noël pour l'industrie du chocolat, le samedi soir pour les admissions aux urgences, juin pour les centrales de réservation de séjours etc. La charge peut alors doubler ou bien plus. Il ne faut donc pas négliger cette estimation.*

Si le calcul du pic pour un composant en bout de chaîne de liaison est complexe (par exemple, un service central du SI exposant des données référentiel et appelé par de nombreux composants qui ont chacun leur pic), on découpera la journée en intervalles de temps suffisamment fins (une heure par exemple) et on calculera sur chaque intervalle la somme mesurée ou estimée des appels de chaque appelant (batch ou transactionnel) pour ainsi déterminer la sollicitation cumulée la plus élevée.

Si l'application tourne sur un cloud de type PaaS, la charge sera absorbée dynamiquement mais veiller à estimer le surcoût et à fixer des limites de consommation cohérentes pour ne pas se ruiner tout en assurant un bon niveau de service.

Exemple d'estimation de charge dynamique pour un composant applicatif d'un site de e-commerce:

Opération REST GET `DetailArticle`	***Estimation***
Nombre d'utilisateurs potentiels	*1M*
Éléments de saisonnalité	*- pic journalier de 20h à 21h00* *- pic annuel de décembre*
Taux maximal d'utilisateurs connectés en même temps de 20h00 à 21h00 en décembre *Nombre maximal d'utilisateurs connectés concurrents*	*5%* *50K*
Durée moyenne d'une session utilisateur	*15 mins*
Nombre d'appel moyen du service par session	*10*
Charge (Transaction / seconde)	*50K * 10/15 / 60 = 556 Tps*

Pour un composant technique (comme une instance de base de donnée) en bout de chaîne et sollicité par de nombreux services, il convient d'estimer le nombre de requêtes en pic en cumulant les appels de toutes les clients et de préciser le ratio lecture /écriture quand cette information est pertinente (elle est très importante pour une base de donnée).

Le niveau de détail de l'estimation dépend de l'avancement de la conception du projet et de la fiabilité des hypothèses. Dans l'exemple plus bas, nous avons déjà une idée du nombre de requêtes pour chaque opération. Dans d'autres cas, on devra se contenter d'une estimation très large sur le nombre de requêtes total à la base de données et un ratio lecture /écriture basée sur des abaques d'applications similaires : dans ce cas, ne pas perdre son temps à vouloir détailler plus à ce stade.

Enfin, garder en tête qu'il s'agit simplement d'estimation à valider lors de campagnes de performances puis en production. Prévoir un ajustement du dimensionnement peu après la MEP (relativement aisé si les ressources matérielles sont virtualisées et/ou si l'architecture est scalable horizontalement).

Exemple : la base de donnée Oracle BD01 est utilisée en lecture par les appels REST GET sur `DetailArticle` mais également en mise à jour par les appels POST et PUT sur `DetailArticle` issus du batch d'alimentation B03 la nuit entre 01:00 et 02:00.

Nombre de requêtes SQL en pic vers l'instance BD01 de 01:00 à 02:00 en décembre	**Estimation**
Taux maximal d'utilisateurs connectés en même temps *Nombre maximal d'utilisateurs connectés concurrents*	*0.5%* *5K*
Durée moyenne d'une session utilisateur	*15 mins*
Nombre d'appel moyen du service par session	*10*
Charge usagers GET DetailArticle (Transaction / seconde)	*(10/15)*5K / 60 = 55 Tps*
Nombre de requête en lecture / écriture par appel de service	*2 / 0*
Nombre journalier d'articles créés par le batch B03 *Nombre de requêtes INSERT / SELECT*	*4K* *3 / 2*
Nombre journalier d'articles modifiés par le batch B03 *Nombre de requêtes SELECT / UPDATE*	*10K* *1 / 3*
Nombre de requêtes par secondes : *SELECT* *INSERT* *UPDATE*	 *55*2 + 2*4K/3600 + 1* 10K/3600= 115 Tps* *0 + 3*4K/3600 = 3.4 Tps* *0 + 3*10K/3600 = 8.3 Tps*

Coupe-circuits

Dans certains rares cas, le pic est extrême et imprévisible (effet 'slashdot'). Si ce risque est identifié, prévoir un système de fusible avec déport de toute ou partie de la charge sur un site Web statique avec message d'erreur par exemple. Ce dispositif peut également servir en cas d'attaque de type DDOS. Ceci permet de montrer qu'on gère le problème sans le subir (on évite les erreurs HTTP 404 ou 50x ou bout d'un long timeout) et permet d'assurer un bon fonctionnement à une partie des utilisateurs.

Qualité de Service

Il est également utile de prévoir des systèmes de régulation applicatifs dynamiques, par exemple :

- *via du throttling (limite de nombre de requêtes par origine et unité de temps). A mettre en amont de la chaîne de liaison ;*
- *des systèmes de jetons (qui permettent en outre de favoriser tel ou tel client en leur accordant un quota de jetons différents).*

Exemple : Le nombre total de jetons d'appels aux opérations REST sur la ressource DetailArticle sera de 1000. Au delà de 1000 appels simultanés, les appelants obtiendront une erreur d'indisponibilité 429 qu'ils devront gérer (et faire éventuellement des rejeux à écarter progressivement dans le temps). La répartition des jetons sera la suivante par défaut :

Opération sur DetailArticle	**Proportion des jetons**
GET	*80%*
POST	*5%*
PUT	*15%*

Exemple 2 : un troddling de 100 requêtes par source et par minute sera mis en place au niveau du reverse proxy.

Augmentation prévisionnelle de la charge

Pour faciliter le dimensionnement et éviter d'avoir à changer de serveurs trop souvent, il est important de donner une estimation de l'augmentation annuelle de la charge. Une façon de faire

dans certaines organisations est de prendre la charge cible estimée à cinq ans et de choisir le matériel en fonction puisqu'il s'agit de sa durée de vie moyenne.

5.10. DURÉE DE RÉTENTION ET ARCHIVAGE

L'archivage concerne les sauvegardes au delà d'un an en général pour raison légales. Les archives sont souvent conservées trente ans ou plus.

Préciser si des données du projet doivent être conservées à long terme. Préciser les raisons de cet archivage (légales le plus souvent, voir https://www.service-public.fr/professionnels-entreprises/vosdroits/F10029).

Exemple : comme exigé par l'article L.123-22 du code de commerce, les données comptables devra être conservées dix ans.

Il est néanmoins crucial de prévoir des purges régulières pour éviter une dérive continue des performances et de l'utilisation disque (par exemple liée à un volume de base de données trop important).

Il est souvent judicieux d'attendre la MEP voire plusieurs mois de recul pour déterminer précisément les durées de rétention (âge ou volume maximal par exemple) mais il convient de prévoir le principe même des purges ou des non-purges dès la définition de l'architecture du projet. Cela peut avoir des conséquences importantes y compris sur le fonctionnel (exemple : s'il n'y a pas de rétention ad vitam aeternam de l'historique, certains patterns à base de listes chaînées ne sont pas envisageables).

Exemple 1 : Les pièces comptables doivent être conservées en ligne (en base) au moins deux ans puis peuvent être archivées pour conservation au moins dix ans de plus.

6. SÉCURITÉ

On présente ici les solutions retenues pour répondre aux exigences de sécurité. Il traite de sujets de niveau applicatif, technique ou logiciel.

6.1. DISPONIBILITÉ

Donner ici les dispositifs permettant d'atteindre la disponibilité exigée au chapitre 5.3.1.

Les mesures permettant d'atteindre la disponibilité exigée sont très nombreuses et devront être choisies par l'architecte en fonction de leur apport et de leur coût (financier, en complexité, …).

Nous regroupons les dispositifs de disponibilité en quatre grandes catégories :

- *dispositifs de supervision (technique et applicative) permettant de détecter au plus tôt les pannes et donc de limiter le MDT (temps moyen de détection) ;*
- *dispositifs organisationnels :*
  - *la présence humaine (astreintes, heures de support étendues, CDD dédiées au projet...) qui permet d'améliorer le MTTR (temps moyen de résolution) et sans laquelle la supervision est inefficiente ;*
  - *la qualité de la gestion des incidents (voir les bonnes pratiques ITIL), par exemple un workflow de résolution d'incident est-il prévu ? si oui, quel est sa complexité ? sa durée de mise en œuvre ? si elle nécessite par exemple plusieurs validations hiérarchiques, la présence de nombreux exploitants n'améliore pas forcement le MTTR ;*
- *dispositifs de redondance technique (clusters, RAID...) qu'il ne faut pas surestimer si les dispositifs précédents sont insuffisants ;*
- *dispositifs de restauration de données (sauvegardes, procédures de restauration) : la procédure de restauration est-t-elle bien définie ? testée ? d'une durée compatible avec les exigences de disponibilité ? C'est typiquement utile dans le cas de perte de données causée par une fausse manipulation ou bug dans le code : il faut alors arrêter l'application et pouvoir restaurer rapidement la dernière sauvegarde permet d'améliorer grandement la disponibilité de l'application.*

Rappels sur les principes de disponibilité :

- *la disponibilité d'un ensemble de composants en série : D = D1 * D2 * … * Dn, exemple : la disponibilité d'un projet utilisant un serveur Tomcat à 98 % et une base Oracle à 99 % sera de 97.02 %;*
- *la disponibilité d'un ensemble de composants en parallèle : D = 1 – (1-D1) * (1- D2) * ..* (1-Dn). Exemple : la disponibilité de trois serveurs Nginx en cluster dont chacun possède une disponibilité de 98 % est de 99.999 % ;*
- *il convient d'être cohérent sur la disponibilité de chaque maillon de la chaîne de liaison : rien ne sert d'avoir un cluster actif/actif de serveurs Websphere si tous ces serveurs attaquent une base de donnée Oracle localisée sur un unique serveur physique avec disques sans RAID.*
- *on estime un système comme hautement disponible (HA) à partir de 99 % de disponibilité ;*
- *on désigne par «spare» un dispositif (serveur, disque, carte électronique...) de rechange qui est dédié au besoin de disponibilité mais qui n'est pas activé en dehors des pannes. En*

fonction du niveau de disponibilité recherché, il peut être dédié au projet ou mutualisé au niveau SI.

- *le niveau de redondance d'un dispositif peut s'exprimer avec la notion suivante si N est le nombre de dispositifs assurant un fonctionnement correct en charge :*

N	*aucune redondance (ex : il faut deux alimentation pour le serveur, si une tombe, le serveur s'arrête)*
N+1	*un composant de rechange est disponible (mais pas forcement actif), on peut supporter la panne d'un matériel (ex : on a une alimentation de spare disponible).*
2N	*le système est entièrement redondé (mais les composants de remplacement ne sont pas forcement actifs) et peut supporter la perte de la moitié des composants (ex : on dispose de quatre alimentations)*

Clustering

- *un cluster est un ensemble de nœuds (machines) hébergeant la même application ;*
- *le failover (bascule) est la capacité d'un cluster de s'assurer qu'en cas de panne, les requêtes ne sont plus envoyées vers le nœud tombé mais vers un nœud opérationnel ;*
- *en fonction du niveau de disponibilité recherché, chaque nœud peut être :*
  - *actif : le nœud traite les requêtes (ex : un serveur Apache parmi dix et derrière un répartiteur de charge). Temps de failover : 0.*
  - *passif en mode «hot standby» : le nœud est installé et démarré mais ne traite pas les requêtes (ex : une base MySql slave qui devient master en cas de panne de ce dernier via l'outil mysqlfailover). Temps de failover : de l'ordre de quelques secondes (temps de la détection de la panne).*
  - *passif en mode «warm standby» : le nœud est démarré et l'application est installée mais n'est pas démarrée (ex : un serveur avec une instance Tomcat hébergeant notre application en mode inactif en plus d'une autre application -elle- démarrée). En cas de panne, notre application est démarrée automatiquement. Temps de failover : de l'ordre de la minute (temps de la détection de la panne et de démarrage de l'application) ;*
  - *passif en mode «cold standby» : le nœud est un simple spare. Pour l'utiliser, il faut installer l'application et le démarrer. Temps de "failover" : de l'ordre de dizaines de minutes avec solutions de virtualisation (ex : KVM live migration) et/ou de containers (Docker) à une journée lorsqu'il faut installer/restaurer et démarrer l'application.*
- *Il existe deux architectures de clusters actif/actif :*
  - *les clusters actifs/actifs à couplage faible dans lesquels un nœud est totalement indépendant des autres, soit parce que l'applicatif est stateless (le meilleur cas), soit parce que les données de contexte (typiquement une session HTTP) sont gérées isolément par chaque nœud. Dans le dernier cas, le répartiteur de charge devra assurer une affinité de session, c'est à dire toujours router les requêtes d'un client vers le même nœud et en cas de panne de ce nœud, les utilisateurs qui y sont routés perdent leurs données de session et doivent se reconnecter. Bien entendu, les nœuds partagent tous les mêmes données persistées en base, les données de contexte sont uniquement des données transitoires en mémoire.*
  - *les clusters actifs/actifs à couplage fort (clusters à tolérance de panne) dans lesquels tous les nœuds forment en quelque sorte une super-machine logique partageant la*

même mémoire. Dans cette architecture, toute donnée de contexte doit être répliquée dans tous les nœuds (ex : cache de sessions HTTP répliqué avec JGroups).

Failover

Le failover (bascule) est la capacité d'un cluster à basculer les requête d'un nœud vers un autre en cas de panne.

Sans failover, c'est au client de détecter la panne et de rejouer sur un autre. Dans les faits, ceci est rarement praticable et les clusters disposent presque toujours de dispositifs de failover.

Une solution de failover peut être décrite par les attributs suivants :

- *automatique ou manuel ? (dans une solution HA, le failover est en général automatique à moins de disposer d'astreintes 24/7/365 et d'une exploitation extrêmement organisée) ;*
- *quelle stratégie de failover et de failback ?*
  - *dans un cluster dit "N+1", on bascule vers un nœud passif qui devient actif et le restera (le nœud en panne une fois réparé pourra devenir le nouveau serveur de secours). Si un serveur cible ne tiendrait pas seul la charge, on prévoit plusieurs serveurs passifs (cluster dit "N+M") ;*
  - *dans un cluster "N-to-1", on rebasculera (failback) sur le serveur qui était tombé en panne une fois réparé et le serveur basculé redeviendra le serveur de secours ;*
  - *dans un cluster N-to-N (architecture en voie de démocratisation avec le cloud de type PaaS comme App-Engine ou CaaS comme Kubernetes ou Rancher) : on distribue les applications du nœud en panne vers d'autres nœuds actifs et qui auront été dimensionnés en prévision de cette éventuelle surcharge.*
- *transparent via à vis de l'appelant ou pas ? en général, les requêtes pointant vers un serveur en panne non encore détectée tombent en erreur (en timeout la plupart du temps). Certains dispositifs ou architectures de FT (tolérance de panne) permettent d'assurer la transparence ;*
- *quelle solution de détection de panne ?*
  - *les répartiteurs de charge utilise des health checkers (tests de santé) très variés (requêtes bouchonnées, analyse du CPU, des logs à distance, etc...) vers les nœuds qu'ils contrôlent ;*
  - *les détections de panne des cluster actifs/passifs fonctionnent la plupart du temps par écoute des palpitations (heartbeat) du serveur passif vers le serveur actif, par exemple via des requêtes multicast UDP dans le protocole VRRP utilisé par keepalived.*
- *quelle durée de détection de la panne ? il convient de paramétrer correctement (le plus court possible sans dégradation de performance) les solutions de détection de panne pour limiter la durée de failover ;*
- *quelle pertinence de la détection ? le serveur en panne est-il *vraiment* en panne ? un mauvais paramétrage peut provoquer une indisponibilité totale d'un cluster alors que les nœuds sont sains.*

Quelques mots sur les répartiteurs de charge :

- *un répartiteur de charge (Load Balancer = LB) est une brique obligatoire pour un cluster actif/actif.*

- *dans le cas des clusters, une erreur classique est de créer un SPOF au niveau du répartiteur de charge. On va alors diminuer la disponibilité totale du système au lieu de l'améliorer. Dans la plupart des clusters à vocation de disponibilité (et pas seulement de performance), il faut redonder le répartiteur lui-même en mode actif/passif (et évidemment pas actif/actif sinon, il faudrait un "répartiteur de répartiteurs"). Le répartiteur passif doit surveiller à fréquence élevée le répartiteur actif et le replacer dès qu'il tombe (les requêtes arrivant au LB en panne avant la bascule sont en erreur) ;*
- *il est crucial de configurer correctement et à fréquence suffisante les tests de vie (heathcheck) des nœuds vers lesquels le répartiteur distribue la charge car sinon, le répartiteur va continuer à envoyer des requêtes vers des nœuds tombés ou en surcharge ;*
- *certains LB avancés (ex : option redispatch de HAProxy) permettent de configurer des rejeux vers d'autres nœuds en cas d'erreur ou timeout et donc d'améliorer la tolérance de panne puisqu'on évite de retourner une erreur à l'appelant pendant la période de pré-détection de la panne.*
- *penser à lisser la charge vers les nœuds et ne pas forcement se contenter de round robin. Un algorithme simple est le LC (Least Connection) permettant au répartiteur de privilégier les nœuds les moins chargés, mais il existe bien d'autres algorithmes plus ou moins complexes (systèmes de poids par nœud ou de combinaison charge + poids par exemple) ;*
- *dans le monde Open Source, voir LVS + keepalived ou HAProxy + keepalived ;*

La tolérance de panne

La tolérance de panne (FT=Fault Tolerance) ne doit pas être confondue avec la disponibilité, elle concerne la capacité d'un système à passer outre les pannes sans perte de données. Par exemple, un disque RAID 1 assure une tolérance de panne transparente ; en cas de panne, le processus écrit ou lit sans erreur après le failover automatique vers le disque sain.

Pour permettre la tolérance de panne d'un cluster, il faut obligatoirement partir sur un cluster actif/actif avec fort couplage dans lequel les données de contexte sont répliquées à tout moment. Une autre solution (la meilleure) est d'éviter tout simplement les données de contexte (en gardant les données de session dans la navigateur via une architecture RIA par exemple) ou de les stocker en base (SQL/NoSQL) ou en cache distribué (mais attention aux performances). Pour disposer d'une tolérance de panne transparente (le niveau de disponibilité le plus élevé), il faut en plus prévoir un répartiteur de charge assurant les rejeux.

Attention à bien qualifier les exigences avant de construire une architecture FT car en générales ces solutions :

1. *complexifient l'architecture et la rendent donc moins robustesse et plus coûteuse à construire, tester, exploiter ;*
2. *peuvent dégrader les performances : les solutions de disponibilité et de performance vont en général dans le même sens (par exemple, un cluster de machines stateless va diviser la charge par le nombre de nœuds et dans le même temps, la disponibilité augmente. Mais quelque fois, disponibilité et performance peuvent être antagonistes : dans le cas d'une architecture stateful, typiquement gérant les sessions HTTP avec un cache distribué (type Infinispan répliqué en mode synchrone) contenant les données de session, toute mise à jour de la session ajoute un surcoût lié à la mise à jour et la réplication des caches, ceci pour assurer le failover (en cas de plantage d'un des nœuds, l'utilisateur conserve sa session à la requête suivante et n'a pas à se reconnecter, mais à quel coût ?) ;*
3. *elles peuvent même dégrader la disponibilité car tous les nœuds sont fortement couplés. Une mise à jour logicielle par exemple peut imposer l'arrêt de l'ensemble du cluster.*

Quelques solutions de disponibilité (hors disponibilité du datacenter) :

Solution	**Coût**	**Complexité de mise en œuvre**	**Amélioration de la disponibilité**
Disques en RAID 1 ou RAID 1+0	*XXX*	*X*	*XXX*
Disques en RAID 5	*X*	*X*	*XX*
Redondance des alimentations et autres composants,	*XX*	*X*	*XX*
Bonding des cartes Ethernet	*X*	*X*	*X*
Cluster actif/passif	*XX*	*XX*	*XX*
Cluster actif/actif (donc avec LB)	*XXX*	*XXX*	*XXX*
Serveurs de spare	*XX*	*X*	*X*
Bonne supervision système	*X*	*X*	*XX*
Bonne supervision applicative	*XX*	*XX*	*XX*
Systèmes de test de vie depuis un site distant	*X*	*X*	*XX*
Astreintes dédiées au projet, 24/7/365	*XXX*	*XX*	*XXX*
Copie du backup du dernier dump de base métier sur baie SAN (pour restauration express)	*XX*	*X*	*XX*

Exemple 1 : Pour atteindre la disponibilité de 98 % exigée, les dispositifs de disponibilité envisagés sont les suivants :

- *tous les serveurs en RAID 5 + alimentations redondées ;*
- *répartiteur HAProxy + keepalived actif/passif mutualisé avec les autres projets ;*
- *cluster actif /actif de deux serveurs Apache + mod_php ;*
- *serveur de spare pouvant servir à remonter la base MariaDB depuis le backup de la veille en moins de 2h.*

Exemple 2 : Pour atteindre la disponibilité de 99.97% exigée, les dispositifs de disponibilité envisagés sont les suivants (pour rappel, l'application sera hébergée dans un datacenter de niveau tiers III, voir §4.5.4) :

- *tous les serveurs en RAID 1+0 + alimentations redondées + interfaces en bonding ;*
- *répartiteur HAProxy + keepalived actif/passif dédié au projet ;*
- *cluster actif /actif de 4 serveurs (soit une redondance 2N) Apache + mod_php ;*
- *instance Oracle en RAC sur deux machines (avec interconnexion FC dédiée).*

6.2. INTÉGRITÉ

Décrire ici les mesures répondant aux exigences exprimées au chapitre 5.3.4.

Classe de données	Niveau exigé	Mesures
Données de la base métier	*Intègre*	*- Utilisation du SGBDR SGBD PostgreSQL avec un niveau d'isolation transactionnelle SERIALIZABLE - Les entités seront référencées uniquement par des ID techniques issues de séquences PostgreSQL.*
Données archivées	*Détecté*	*Génération de checksums SHA-256 des backups*
Données calculées XXX	*Maîtrisé*	*Stockage d'un checksum SHA1, relance du calcul automatiquement par batch dans les 24H.*
Silo NoSQL des données Big Data avant consolidation	*Non intègre*	*Pas de mesure particulière, pas de backup*
Sources du projet	*Intègre*	*Utilisation du SCM Git*
Avis d'imposition PDF	*Intègre*	*Signature numérique par la clé privée de l'administration des données D de l'avis au format PKCS#7 (RSA, SHA256) avec horodatage. D= base64(montant net + date+nom). La signature résultante sera intégrée a posteriori au format hexa en pied de page du PDF.*

6.3. CONFIDENTIALITÉ

Décrire ici les mesures répondant aux exigences exprimées au chapitre 5.3.5.

Classe de données	Niveau exigé	Mesures
Contenu éditorial du projet	*Public*	*Aucune, contenu en HTTP et HTTPS, pas d'authentification.*
Profil du compte du site Web	*Limité*	*L'accès à ce contenu nécessite une authentification réussie par login/mot de passe.*
Historique du compte	*Réservé*	*L'accès à ce contenu est réservé aux exploitants habilités, uniquement via des requêtes PL/SQL de la base de données.*
Logs des activités de l'internaute	*Réservé*	*L'accès aux fichiers de log est réservé aux exploitants habilités (accès SSH à la machine XXX et mot de passe Unix)*
Données du projet RH " aides sociales aux employés"	*Privé*	*Ces données sont chiffrées en AES 256 sous forme d'un BLOB en base, remontées au client Web via le service REST YYY puis déchiffrées au sein du navigateur dans l'application AngularJS via un mot de passe complémentaire de l'utilisateur (non stocké coté serveur). Il s'agit donc d'un chiffrement client uniquement. Une perte de mot de passe rend les données irrécupérables. Les données modifiées sur le client sont chiffrées et enregistrées à nouveau dans le BLOB via le service REST XXX.*

Penser aussi à la confidentialité de données dérivées :

- *Chiffrement des backups*
- *Chiffrement des données clientes pour les applications lourdes. Cela peut être un chiffrement matériel en SED (Self Encryption Disk), un chiffrement logiciel de niveau partition (SafeGuard, dm-crypt) ou de niveau fichier (encfs, TrueCrypt,…)*

6.4. IDENTIFICATION

Décrire ici les mesures répondant aux exigences exprimées au chapitre 5.3.6.

En cas d'utilisation de fédération d'identité, renvoyer le lecteur au chapitre 6.6.

Exemple 1 : L'Id des usagers de l'application sera l'attribut uid des DN `cn=XXX,ou=service1,dc=entreprise,dc=com` dans l'annuaire LDAP central. Un filtre sera également appliqué sur l'appartenance au groupe `ou=monapplication,dc=entreprise,dc=com`

Exemple 2 : Pour assurer la non réutilisation des ID des comptes supprimés, une table d'historique sera ajouté dans l'application et requêtée avant toute création de nouveau compte.

6.5. AUTHENTIFICATION

Décrire ici les mesures répondant aux exigences exprimées au chapitre 5.3.7.

Pour les authentifications par mot de passe, décrire le mode de stockage et de vérification. Penser également à décrire les solutions de changement de mot de passe.

En cas d'utilisation de fédération d'identité, renvoyer le lecteur au chapitre 6.6.

Exemple 1 : L'authentification des internautes inscrits se fera par login/mot de passe (respectant la politique de mot de passe XXX)

Exemple 2 : L'authentification des internautes à l'inscription se fera par la saisie du code internaute figurant sur les factures + la valeur de la dernière facture puis par l'activation du compte via un lien figurant dans un e-mail de vérification.

Exemple 3 : lors de la création d'un nouveau bénéficiaire de virement dans l'espace internet, l'utilisateur devra fournir un mot de passe unique issu de son token OTP en plus d'être authentifié.

Exemple 4 : Les mots de passe ne seront en aucun cas conservés mais stockés sous la forme de digest bcrypt.

6.6. FÉDÉRATION D'IDENTITÉ

En cas de fédération d'identité, détailler la technologie choisie et son intégration dans l'architecture du projet. Les solutions les plus courantes sont actuellement : France Connect, OpenId Connect, SAML ou Oauth 2.0 (pseudo-authentification seulement). Pour les applications Web, préciser les contraintes navigateur (activation des cookies en particulier).

Exemple : L'IHM grand public permettra une identification et authentification France Connect de sorte que les utilisateurs puissent utiliser leur compte DGFiP ou CNAM pour s'identifier et s'authentifier. La cinématique d'authentification sera la suivante : <schéma>

6.7. SSO, SLO

Décrire ici les mesures répondant aux exigences exprimées au chapitre 5.3.8.

En cas de SSO/SLO, détailler la technologie choisie et son intégration dans l'architecture du projet. Les solutions les plus courantes sont actuellement : CAS, OpenAM, LemonLDAP::NG. Pour les applications Web, préciser les contraintes navigateur (activation des cookies en particulier).

Exemple 1 : L'IHM XXX intégrera un client CAS spring-security pour le SSO. Le serveur CAS utilisé sera YYY. Son royaume d'authentification (realm) sera l'annuaire AD XXX. <schéma>

Exemple 2 : Comme toutes les applications du portail métier, l'IHM XXX devra gérer les callbacks de déconnexion provenant du serveur CAS suite à une demande de SLO. <schéma>

6.8. PREUVE

Décrire ici les mesures répondant aux exigences exprimées au chapitre 5.3.9.

Exemple 1 : tous les documents servant de preuve seront archivés dans la GED Alfresco.

Exemple 2 : Les logs contenant le terme [PREUVE] et issu de l'ensemble des composants seront centralisés via le système de centralisation de log Logstash + Elastic Search puis insérés de façon journalière vers la base MongoDB « preuves ».

6.9. NON-RÉPUDIATION

Décrire ici les mesures répondant aux exigences exprimées au chapitre 5.3.10.

Exemple : La déclaration d'impôt sera signée par le certificat client de l'usager (certificat X509, RSA, SHA-256) qui lui a été fourni par le composant XXX suivant l'architecture suivante : <schéma>.

6.10. ANONYMAT ET VIE PRIVÉE

Décrire ici les mesures répondant aux exigences exprimées au chapitre 5.3.11.

Exemple 1 : un audit interne sera mené une fois par an sur le contenu des données en base et les extractions à destination des partenaires.

Exemple 2 : les données à destination de la zone public sera exporté partiellement via un `COPY (SELECT …) TO <fichier>`. Les colonnes sensibles seront ainsi exclues de la réplication.

Exemple 3 : le bandeau d'acceptation des cookies sera mis en ouvre sur toutes les pages de l'application AngularJS via le module angular-cookie-law.

6.11. AUTORISATIONS

Décrire ici les mesures répondant aux exigences exprimées au chapitre 5.3.12.

Exemple 1 : la gestion des autorisations sera gérée applicativement et stockée au sein de tables `Utilisateur`, `Groupe`, `Role` et `Privilege` de la base de donnée PostgreSQL. Ces tables seront décrites dans le dossier de spécification.

Exemple 2 : L'obtention du carnet d'adresse Facebook sera en OAuth2. Le client Oauth utilisera sera l'API Java Google Oauth2.

6.12. TRACABILITÉ, AUDITABILITÉ

Décrire ici les mesures répondant aux exigences exprimées au chapitre 5.3.13.

Exemple 1 : à la fin de chaque action métier, l'application ReactJS appellera dans une action asynchrone un service REST de trace métier. Ce service stockera les traces dans une base Elastic Search pour consultation en Kibana. <schéma>

Exemple 2 : l'outil d'IDS hybride (réseau + host) OSSEC sera installé au bureau du reverse-proxy HA Proxy servant d'entrée dans le SI. Il supervisera les intrusions de l'ensemble des machines impliquées dans le projet.

Exemple 3 : Les tables XXX, YYY, .. seront historisées suivant le principe suivant : … <diagramme de classe>

6.13. AUTO-CONTRÔLE DES VULNÉRABILITÉS

La gestion des vulnérabilités dépasse largement le cadre de ce document mais il est bon de s'auto-controler pour s'assurer que les failles les plus courantes sont bien prises en compte et comment. Cette liste est en partie basée sur le TOP 10 OWASP. Pour le TOP 10 des application mobiles, adapter cette liste avec le TOP 10 mobile .

Vulnérabilité	Pris en compte ?	Mesure technique entreprises
Accès à des ports privés	*X*	*Configuration du firewall iptables sur la machine exposée à Internet. Seul les ports 80 et 443 sont ouverts. Le firewall sera configuré en mode stateful (avec extension conntrack) .*
Attaque de mot de passe par force brute	*X*	*Utilisation de fail2ban, mise en prison de 1h au bout de 3 tentatives de connexion ssh.*
Visibilité des URLs directes	*X*	*Centralisation de tous les accès depuis Internet via un reverse proxy Apache + mod_proxy. Réécriture d'URLs pour masquer les URL internes.*
Contournement du contrôle d'accès	*X*	*Utilisation du SSO CAS, voir 6.7.*
Injection SQL	*X*	*Utilisation de `PreparedStatement` uniquement, audit des requêtes SQL.*
Injection OS	*X*	*Vérification qu'il n'y a aucun appel de commandes systèmes dans le code (type `Runtime.exec()`)*
Violation de gestion d'authentification et de session	*X*	*Traité avec le dispositif anti-CSRF, voir plus bas. On logue l'IP à fin d'audit.*

XSS	*X*	*- Utilisation de librairie d'échappement. Pour les modules Java, nous utiliserons `StringEscapeUtils.escapeHtml4()` de commons-lang.*
		- Utilisation des headers HTTP :
		`X-Frame-Options SAMEORIGIN`
		`X-XSS-Protection 1;mode=block`
		`X-Content-Type-Options nosniff`
		`Content-Security-Policy`
		`X-XSS-PROTECTION` *(pour parer les détournements de dispositifs anti-XSS des navigateurs)*
		- Spécification systématique de l'encoding dans le header de réponse Content-Type (ex : `text/html;charset=UTF-8`*) pour parer les attaques basées sur des caractères spéciaux concournant l'anti-XSS.*

ReDOS	*X*	*Vérification que les expressions régulières utilisées par les dispositifs anti-XSS ne sont pas eligibles à ce type d'attaque, voir https://www.owasp.org/index.php/Regular_expression_Denial_of_Service_-_ReDoS .*
Référence directe à un objet	*X*	*Vérification à chaque requête que les arguments passés correspondent bien à la personne identifiée. Par exemple, toute requête contient son ID et on vérifie par une requête que le dossier qu'elle tente de consulter lui appartient bien avant de poursuivre la requête initiale.*
Planification des mises à jour de sécurité	*X*	*- Les mises à jour Centos seront planifiées tous les premier mercredi du mois.* *- Les mises à jour Wildfly sont appliqué au plus deux semaines après leur sortie.*
Exposition de données sensibles	*X*	*- Tous les algorithmes de sécurité sont à jour : au minimum SHA-256, AES 256* *- Le SSL V2 et V3 est désactivé coté Apache suite à la faille DROWN (* ***SSLProtocol all -SSLv2 -SSLv3****)* *- L'application ne fonctionne qu'en HTTPS*
CSRF	*X*	*- Utilisation du dispositif anti-CSRF d'AngularJS (https://docs.angularjs.org/api/ng/service/$http)*
Manque de contrôle d'accès au niveau fonctionnel	*X*	*- Mise en place de la politique d'autorisation décrite en 5.3.12* *- Campagne de tests fonctionnels*
Log injection	*X*	*- Échappement des logs avant de les transmettre à log4j* *- Vérification des outils de consultation de logs*
Attaques HTTPS + compression CRIME/BREACH	*X*	*- Désactivation de la compression HTTPS au niveau de l'Apache : SSLCompression off* *- Dispositif anti-CSRF*
Upload de fichiers malicieux	*X*	*- Validation des pieces jointes par l'anti-virus clamav*
...		

7. POINT DE VUE APPLICATIF

Le grain est ici l'application dans son ensemble ou le composant applicatif (partie d'application correspondant à un livrable). On décrit les composants applicatifs, leurs interactions et les données qu'ils échangent ou stockent.

7.1. PRINCIPES DE L'ARCHITECTURE APPLICATIVE

Rappeler ici :

- *le type d'architecture (client-serveur, Web monolithique, SOA, micro-service...) ;*
- *le découpage en couches (présentation, service, métier, persistance ...) ;*
- *les principes d'architecture applicatives (normalement définis par les urbanistes) ;*
- *d'éventuelles dérogations.*

Exemple 1 : les appels inter-services sont interdits sauf les appels de services à un service de nomenclature.

Exemple 2 : tous les batchs doivent pouvoir fonctionner en concurrence des IHM sans verrouillage des ressources.

Exemple 3 : les services ne peuvent être appelés directement. Les appels se feront obligatoirement via une route exposées au niveau du bus d'entreprise qui appellera à son tour le service. Il est alors possible de contrôler, prioriser, orchestrer ou piloter les appels.

Exemple 4 : Les composants de ce projet suivent l'architecture SOA telle que définie dans le document de référence XXX.

7.2. ARCHITECTURE APPLICATIVE GÉNÉRALE

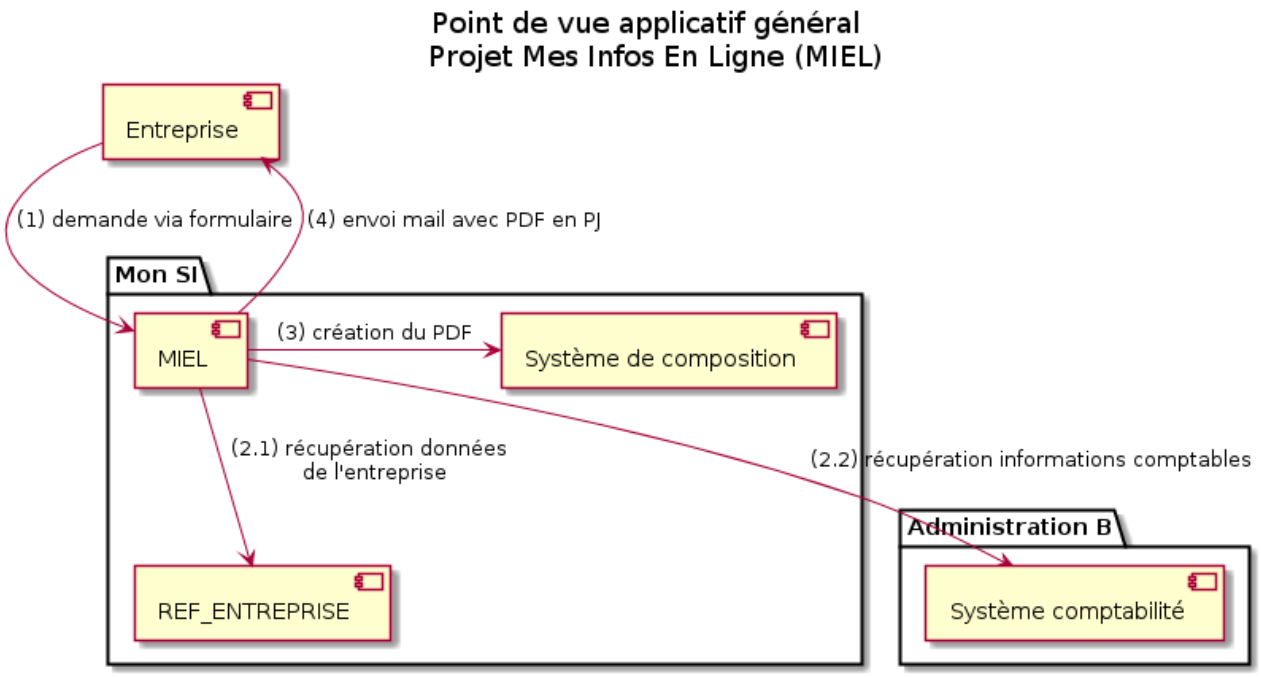
Présenter ici l'application dans son ensemble (sans détailler ses sous-composants) en relation avec les autres applications du SI. Présenter également les macro-données échangées ou stockées.

Le choix de la représentation est libre mais un diagramme de composant UML2 semble le plus adapté.

Numéroter les étapes par ordre chronologique assure une meilleure compréhension du schéma. Grouper les sous étapes par la notation x, x.y, x.y.z, ...

Ne pas faire figurer les nombreux systèmes d'infrastructure (serveur SMTP, dispositif de sécurité, reverse proxy, annuaires LDAP, ...) qui sont du domaine de l'architecture technique. Mentionner en revanche les éventuels bus d'entreprises qui ont un rôle applicatif (orchestration de service par exemple).

Exemple de vision applicative générale d'une application MIEL = "MesInfosEnLigne" permettant à une entreprise de récupérer par mail un document listant ce que l'administration connait sur elle. L'administration va compléter ses données par celle d'une autre administration.



7.3. ARCHITECTURE APPLICATIVE DÉTAILLÉE

Détailler ici tous les composants du projet, leurs flux entre eux et avec les autres applications du SI.

7.3.1 Schéma global

Proposer un ou plusieurs schémas (diagramme UML2 de composant de préférence) avec flux numérotés.

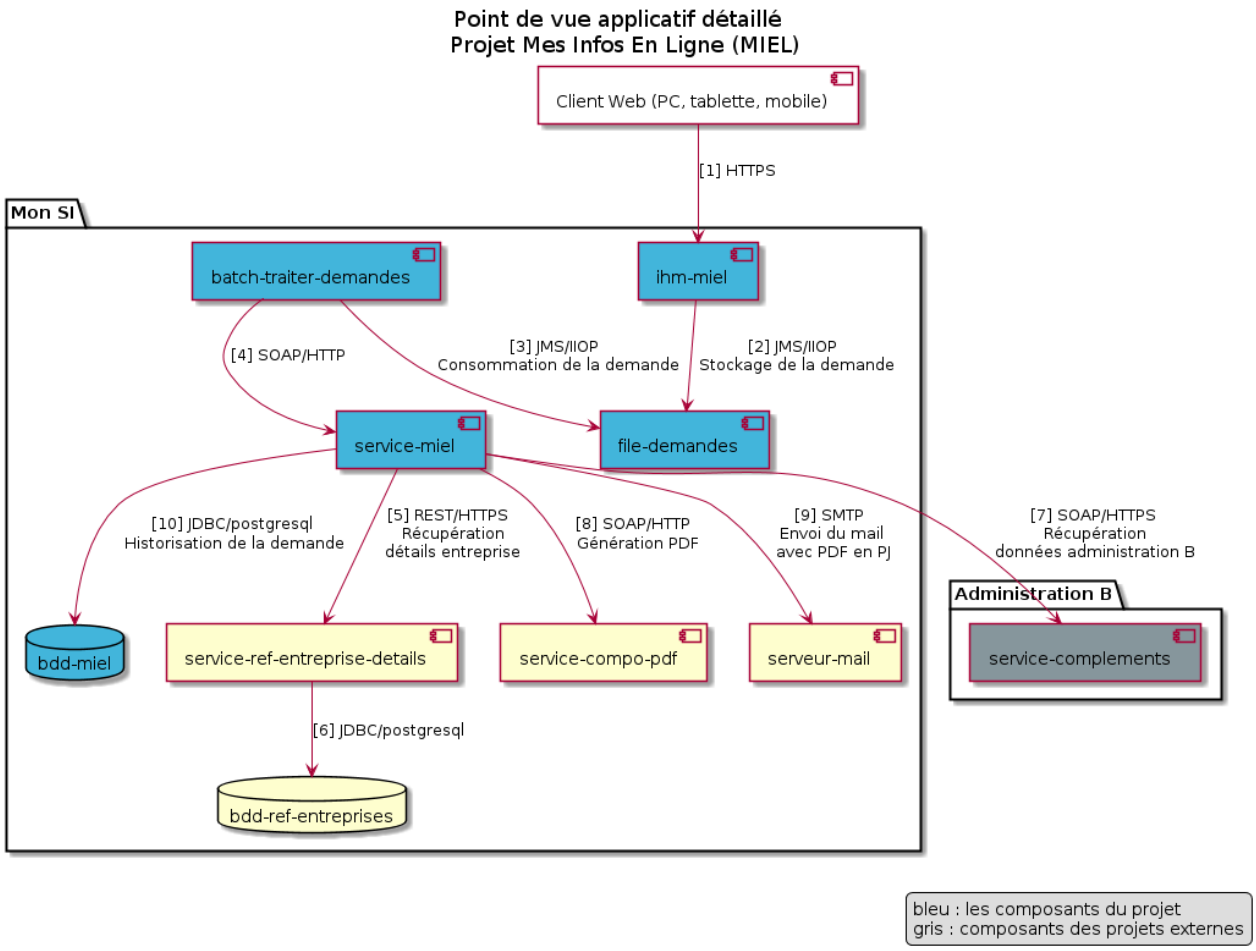
Idéalement, le schéma tiendra sur une page A4 et sera autoporteur. Il devrait devenir un des artefacts documentaire les plus importants du projet et figurer dans la war room d'un projet agile par exemple ou être imprimé par chaque développeur. Il convient donc de particulièrement le soigner.

Si l'application est particulièrement complexe, faire un schéma par chaîne de liaison.

Utiliser comme ID des flux une simple séquence non signifiante (1, 2, ..., n).

Les flux sont logiques et non techniques (par exemple, on peut représenter un flux HTTP direct entre deux composants alors qu'en réalité, il passe par un répartiteur de charge intermédiaire). Ce niveau de détail sera donné dans le point de vue technique en 9.

Exemple d'application :



7.3.2 Matrice des flux applicatifs

ID	De	vers	Protocole	Réseau	Description	(S)ynchrone/(A)synchrone	(L)ecture/(E)criture
1	Navigateur de l'usager	ihm-miel	HTTPS	Internet	Interactions avec l'IHM JSF2 du projet	S	L
2	ihm-miel	file-demandes	JMS/ IIOP	LAN	Production de la demande de renseignement	S	E
3	batch-traiter-demandes	file-demandes	JMS/ IIOP	LAN	Consommation de la demande de renseignement au fil de l'eau (intervalles de l'ordre de 5 mins)	S	E
4	batch-traiter-demandes	service-miel	SOAP/ HTTP	LAN	Appel du service de demande de notre application	S	L
5	service-miel	service-ref-entreprise-details	REST/ HTTPS	LAN	Appel sécurité pour récupérer les données de l'entreprise dans le référentiel	S	L

6	service-ref-entreprise-details	Base bdd-ref-entreprises	JDBC/ pg	LAN	Accès à la base	S	L
7	service-miel	service-complements	SOAP/ HTTPS	WAN sur LS privée	Appel sécurisé TLS (double certificats X509) à la administration B pour récupérer les données complémentaires de l'entreprise	S	L
8	service-miel	service-compo-pdf	SOAP/ HTTP	LAN	Appel au service de composition JasperReport de PDF	S	L
9	service-miel	serveur-mail	SMTP	LAN	Envoi du mail avec le PDF en PJ	S	E
10	service-miel	Base bdd-miel	JDBC /PG	LAN	Historisation de la demande	S	E

7.3.3 Détails

Optionnellement, clarifier certains points de façon textuelle si le schéma et la table des flux applicatifs ne suffit pas (ne pas reprendre tout le schéma, il s'agit de points précis uniquement).

Exemple : Le courriel ne devra pas dépasser 5 Mo pour passer les filtres du relais XXX. Il convient de prévoir un contrôle dans service-miel en retour du service de composition.

8. POINT DE VUE LOGICIEL

Le point de vue logiciel concerne le développement des composants eux-mêmes, indépendamment de leur environnement d'exécution.

8.1. PILE LOGICIELLE

Lister ici pour chaque composant les principales librairies et frameworks utilités ainsi que leur version. Ne pas lister les librairies fournies (provided) par les serveurs d'application ou les frameworks. Inutile de trop détailler, donner uniquement les composants structurants.

Exemple :
Composant ihm-monappli :

Librairie	Rôle	Version
Framerwork Angular2	Framework JS de présentation	2.1.1
JasperReport	Editique transactionnelle, composition des factures au format PDF	6.3.0

8.2. DÉROGATIONS TECHNOLOGIQUES

Préciser si on utilise des composantes avec dérogation (composants non supportés par l'organisation).
Exemple : comme valisé en comité d'architecture du 12/01/2016,, nous utilisons ReactsJS à titre expérimental au sein de l'organisation XXX.

8.3. SPÉCIFICITÉS D'USINE LOGICIELLE

Sans reprendre le fonctionnement de la PIC (Plate-forme d'Intégration Continue) de l'organisation, préciser si ce projet nécessite une configuration particulière.

Exemple : Les jobs Jenkins produiront le logiciel sous forme de containers Docker si tous les TU sont passants. Les tests d'intégration seront ensuite exécutés sur ce container. Si tous les tests d'intégration et BDD sont passants, le container est releasé dans Nexus.

8.4. SPÉCIFICITÉS DES TESTS

Une méthodologie ou une technologie particulière est-elle en jeu dans ce projet ?
Exemple 1 : ce projet sera couvert en plus des TU et tests d'intégration car des tests d'acceptance BDD (Behavioral Driven Development) en technologie JBehave + Serenity.

Exemple 2 : ce projet sera développé en TDD (test first)

8.5. PATTERNS NOTABLES

Préciser si ce projet a mis en œuvre des patterns (GoF, JEE ou autre) structurants) l'intérieur des composants logicielles. Inutile de reprendre les patterns déjà supportés par les langages ou les serveurs d'application (par exemple, l'IoC avec CDI dans un serveur JEE 6).

Exemple 1 : pour traiter l'explosion combinatoire des contrats possibles et éviter de multiplier les niveaux d'héritage, nous utiliserons massivement la pattern décorateur [GoF] dont voici un exemple d'utilisation : <schéma>..

9. POINT DE VUE TECHNIQUE

Le point de vue technique concerne l'infrastructure : serveurs, réseaux, systèmes d'exploitation, bases de données, intergiciels (middleware) ... , bref ce qui fait tourner l'application et qui n'est pas l'application elle-même.

Ne pas pas répéter ici les solutions de sécurité (clustering en particulier) présentées au chapitre 6, Y faire seulement référence au besoin.

9.1. PRINCIPES DE L'ARCHITECTURE TECHNIQUE

Quels sont les grands principes techniques de notre projet ?

Exemple :

- *Les composants applicatifs exposés à Internet dans une DMZ protégé derrière un reverse-proxy et sur un VLAN isolé.*
- *Concernant les interactions entre la DMZ et l'intranet, un pare-feu PIX ne permet les communications que depuis l'intranet vers la DMZ*
- *Comme décrit au chapitre 6, les clusters actifs/actifs seront exposés derrière un LVS + Keepalived avec direct routing pour le retour.*

9.2. DÉPLOIEMENT EN PRODUCTION

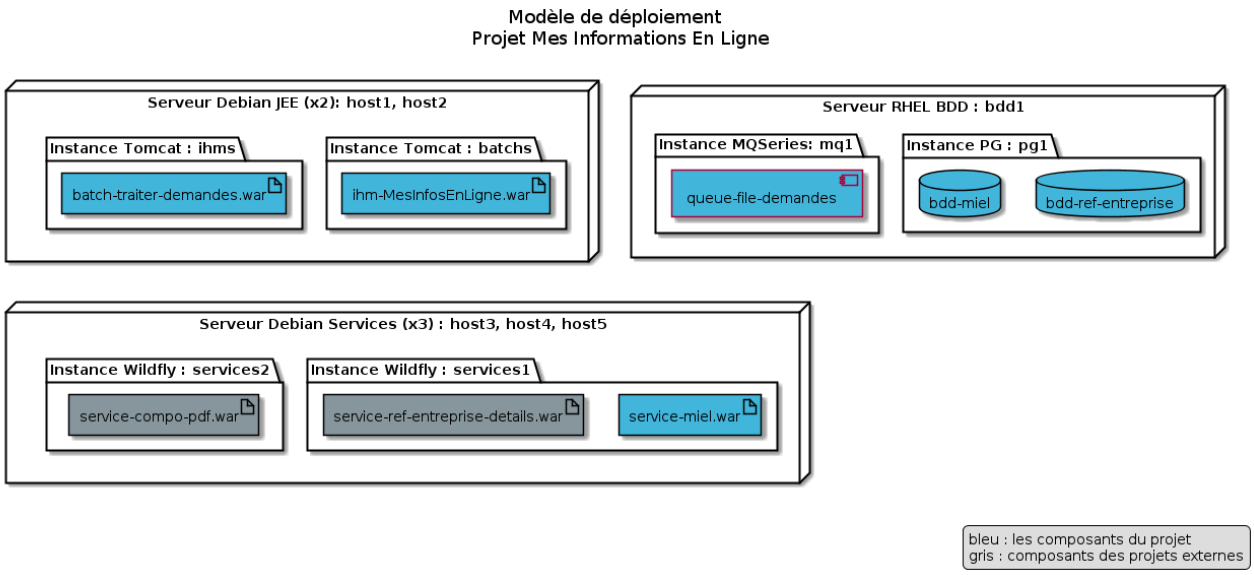
Fournir ici le modèle de déploiement des composants (ici représentés en tant qu'artefacts) sur les différents intergiciels et nœuds physiques (serveurs). Attention : ne pas détailler ici les composants d'infrastructure. Cette section doit permettre de répondre à la question « où tourne le composant x ? ».

Tout naturellement, on le documentera de préférence avec un diagramme de déploiement UML2.

Pour les clusters, donner le facteur d'instanciation en attribut du nœud.

Mentionner optionnellement les composants qui tournent sur le même serveur ou le même serveur d'application que le projet pour en rappeler la proximité, en particulier s'ils sont des dépendances du projet.

Exemple :



9.3. SCHÉMA GLOBAL

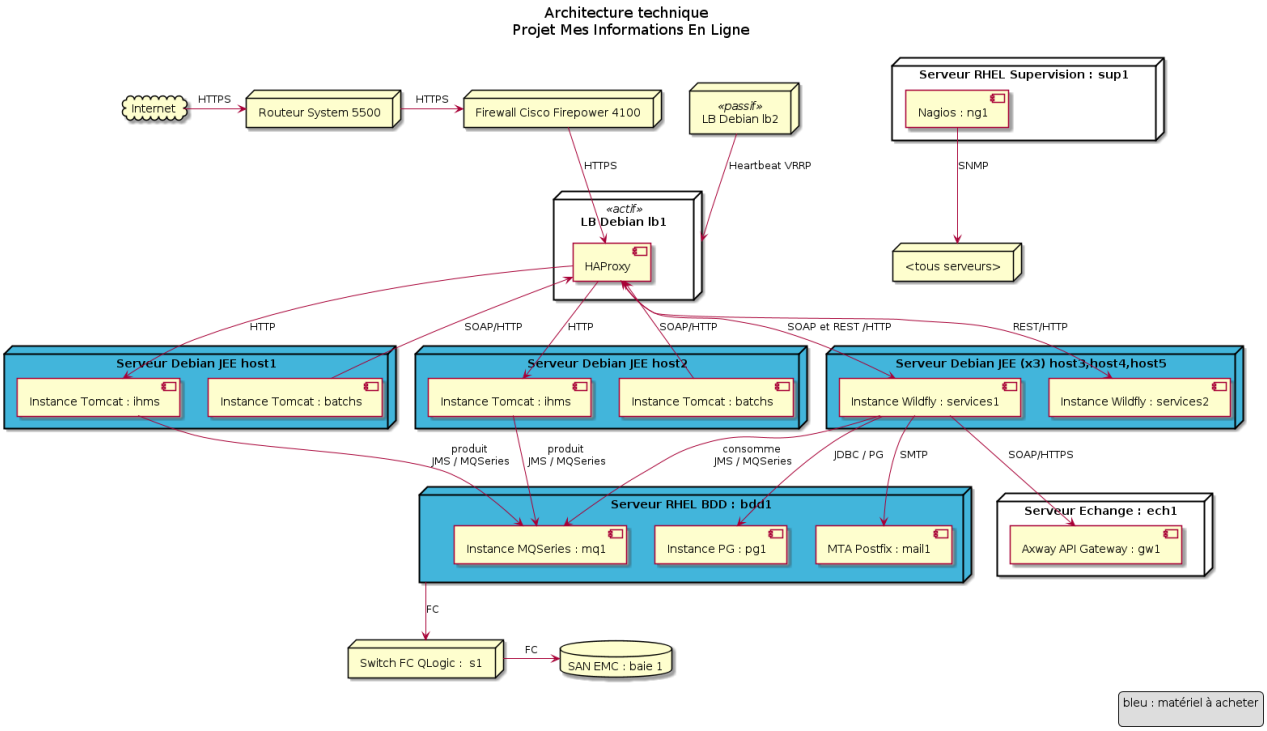
On donne ici un schéma d'architecture technique global avec les OS, les composants d'infrastructure logiciels (serveurs d'application, bases de données...) voire réseau (pare-feu, appliances, routeurs...) uniquement quand pertinents. Attention néanmoins à ne pas aller trop dans le détail, surtout concernant les comptants systèmes ou réseau de niveau SI au risque de dupliquer la documentation entre projets et de rendre le document inmaintenable.

Identifier clairement le matériel dédié au projet (et éventuellement à acheter).

Ce schéma sera de préférence un diagramme de déploiement UML2.

On donne ici les véritables hostnames ou IP des machines (sauf si cela pose des problèmes de sécurité).

Exemple :



9.4. VERSIONS DES COMPOSANTS D'INFRASTRUCTURE

OS, bases de données, MOM, serveurs d'application, etc...

Composant	Rôle	Version	Environnement technique
CFT	Transfert de fichiers sécurisé	X.Y.Z	RHEL 6
Wildfly	Serveur d'application JEE	9	Debian 8, OpenJDK 1.8.0_144
Tomcat	Container Web pour les IHM	7.0.3	CentOS 7, Sun JDK 1.8.0_144
Nginx	Serveur Web	1.11.4	Debian 8
PHP + `php5-fpm`	Pages dynamiques de l'IHM YYY	5.6.29	nginx
PostgreSQL	SGBDR	9.3.15	CentOS 7

9.5. MATRICE DES FLUX TECHNIQUES

On liste ici l'intégralité des flux techniques utilisés par l'application. Les ports d'écoute sont précisés. On détaille aussi les protocoles d'exploitation (JMX ou SNMP par exemple). Dans

certaines organisions, cette matrice sera trop détaillée pour un dossier d'architecture et sera maintenue dans un document géré par les intégrateurs ou les exploitants.

Il n'est en général pas nécessaire de faire référence aux flux applicatifs tels que décrits en 7.3.2 car les lecteurs ne recherchent pas les mêmes infirmations. Ici, les exploitants ou les intégrateurs recherchent l'exhaustivité des flux à fin d'installation et de configuration des pare-feu par exemple.

Exemple partiel :

ID	Source	Destination	Protocole	Port d'écoute
1	lb2	lb1	VRRP sur UDP	IP multicast 224.0.0.18
2	lb1	host1, host2	HTTP	80
3	host3, host4, host5	bdd1	PG	5432
4	sup1	host[1-6]	SNMP	199

9.6. DIMENSIONNEMENT DES MACHINES

Mémoire, disque et CPU de chaque nœud. A affiner après campagne de performance ou MEP.

Pour les VM, on considère le disque des partitions système comme interne même si elles sont physiquement sur un SAN.

Le disque SAN concerne des partitions monté sur SAN depuis un serveur physique ou virtuel.

Exemple :

Machines virtuelles lb1, host2 (Reverse proxy)

CPU	Mémoire	Disque interne	Disque SAN
2 VCPU	4 Go	20 Go	-

Machine physique bdd1 (BDD PostgreSQL)

CPU	Mémoire	Disque interne	Disque SAN
16 cœurs Xeon 3Ghz	24 Go	20 Go	3 To

10. EXPLOITATION ET SUPERVISION

Lister ici les grands principes d'exploitation de la solution, pour plus de détail (filesystems sauvegardés, plan de production, planification des traitements...), prévoir un DEX (Dossier d'EXploitation). Si le projet reste dans le standard de l'organisation, ce référer à un dossier commun.

10.1. ORDRE D'ARRÊT/DÉMARRAGE

Préciser ici l'ordre de démarrage des machines et composants entre eux ainsi que l'ordre d'arrêt. En fonction des situations, on peut faire figurer les composants hors projet ou non.

Le DEX contiendra une version plus précise de ce chapitre (notamment avec un numéro d'ordre SystemV ou un "Wants" SystemD précis), ce sont surtout les principes généraux des ordres d'arrêt et de démarrage qui doivent ici être décrits.

Le démarrage se fait en général dans le sens inverse des chaînes de liaison et l'arrêt dans le sens de la chaîne de liaison.

Préciser si éventuelles problématiques si le démarrage est incomplet (par exemple, le pool de connexions du serveur d'application va-t-il retenter de se connecter à la base de donnée si elle n'est pas démarrée ? combien de fois ? quel est le degré de robustesse de la chaîne de liaison ?)

Exemple :

Démarrage

Ordre	Composant	Type
1	pg1 sur serveur bdd1	Instance de donnée PG
2	mq1 sur bdd1	Instance MQSeries (Queue Manager)
3	services1 sur serveurs host3, host4 et host5	Instance Wildfly
4	services2 sur serveurs host3, host4 et host5	Instance Wildfly
5	batchs sur serveurs host1, host2	Instance Tomcat
5	ihms sur serveurs host1, host2	Instance Tomcat

Arrêt
Inverse exact du démarrage

10.2. OPÉRATIONS PROGRAMMÉES

Lister de façon macro (le DEX détaillera le plan de production précis) :

- *les batchs ou famille de batchs et leurs éventuelles inter-dépendances. Préciser si un ordonnanceur sera utilisé.*

- *les traitements internes (tâches de nettoyage / bonne santé) du système qui ne remplissent uniquement des rôles techniques (purges, reconstruction d'index, suppression de données temporaires...)*

Exemple 1 : le batch traiter-demande fonctionnera au fil de l'eau. Il sera lancé toutes les 5 mins depuis l'ordonnanceur JobScheduler.

Exemple 2 : le traitement interne ti_index est une classe Java appelant des commandes REINDEX en JDBC lancées depuis un scheduler Quartz une fois par mois.

10.3. SAUVEGARDES ET RESTAURATIONS

Donner la politique générale de sauvegarde. Elle doit répondre aux exigences du chapitre 5.3.3. De même les dispositifs de restauration doivent être compatibles avec les exigences de disponibilité exprimés en 5.3.1 :

- *Quels sont les backups à chaud ? à froid ?*
- *Que sauvegarde-t-on ? (bien sélectionner les données à sauvegarder car le volume total du jeu de sauvegardes peut facilement atteindre dix fois le volume sauvegardé).*
  - *des images/snapshots systèmes pour restauration de serveur ou de VM ?*
  - *des systèmes de fichiers ou répertoires ?*
  - *des bases de données sous forme de dump ? sous forme binaire ?*
  - *le contenu de files ?*
  - *les logs ? les traces ?*
- *Les sauvegardent sont-elles chiffrées ? si oui, préciser l'algorithme de chiffrement symétrique utilisé et comment sera gérée la clé ;*
- *Les sauvegardes sont-elles compressées ? si oui, avec quel algorithme ? (gzip, bz2, lzma ? xv ? ...) quel paramétrage (indice de compression) ? attention à trouver le compromis entre durée de compression / décompression et gain de stockage ;*
- *Quel outillage est mis en ouvre ? (peut aller de backup-manager à IBM TSM)*
- *Quelle technologie est utilisée pour les sauvegardes ? (bandes magnétiques type LTO ou DLT ? disques externes ? cartouches RDX ? cloud de stockage comme Amazon S3 ? support optique ? NAS ? ...)*
- *Quelle est la périodicité de chaque type de sauvegarde ? (ne pas trop détailler ici, ceci sera dans le DEX)*
- *Quelle est la stratégie de sauvegarde ?*
  - *complètes ? incrémentales ? différentielles ? (prendre en compte les exigences en disponibilité. La restauration d'une sauvegarde incrémentale sera plus longue qu'une restauration de sauvegarde différentielle, elle même plus longue qu'une restauration de sauvegarde complète)*
  - *quel roulement ? (si les supports de sauvegarde sont écrasés périodiquement)*
- *Comment se fait le bilan de la sauvegarde ? par courriel ? où sont les logs ?*

- *Où sont stockées les sauvegardes ? (idéalement le plus loin possible du système sauvegardé tout en permettant une restauration dans un temps compatible avec les exigences de disponibilité) ;*
- *Qui a accès physique ou informatique aux sauvegardes ? à la clé de chiffrement ? (penser aux exigences de confidentialité)*
- *Des procédures de contrôle de sauvegarde et de test de restauration sont-il prévus ? (prévoir un test de restauration une fois par an minimum)*

Exemple de roulement : jeu de 21 sauvegardes sur un an :

- *6 sauvegardes journalières incrémentales ;*
- *1 sauvegarde complète le dimanche et qui sert de sauvegarde hebdomadaire ;*
- *3 sauvegardes hebdomadaires correspondant aux 3 autres dimanches. Le support du dernier dimanche du mois devient le backup mensuel ;*
- *11 sauvegardes mensuelles correspondant aux 11 derniers mois.*

10.4. ARCHIVAGE

Décrire ici les dispositifs permettant de répondre aux besoins d'archivage exprimés au chapitre 5.10

La plupart du temps, la dernière sauvegarde mensuelle de l'année devient une archive dont les modalités de stockage sont :

- *la technologie : idéalement, on dupliquera par sécurité l'archive sur plusieurs supports de technologies différentes (bande + disque dur par exemple);*
- *un lieu de stockage spécifique et distinct des sauvegardes classiques (coffre en banque par exemple);*

Exemple : la sauvegarde annuelle RDX deviendra une archive dupliquée sur bande LTO. Les deux supports seront stockés en coffre dans deux banques différentes.

10.5. PURGES

Donner ici les dispositifs techniques répondant aux exigences du chapitre 5.10.

Exemple : l'historique des consultations sera archivé par un dump avec une requête SQL de la forme `COPY (SELECT * FROM matable WHERE ...) TO '/tmp/dump.tsv'` puis purgé par une requete SQL DELETE après validation par l'exploitant de la complétude du dump.

10.6. LOGS

Sans être exhaustif sur les fichiers de logs (à prévoir dans le DEX), présenter la politique générale de production et de gestion des logs :

- *Quelles sont les politiques de roulement des logs ? le roulement est-il applicatif (via un `DailyRollingFileAppender` log4j par exemple) ou système (typiquement par le démon logrotate) ?*
- *Une centralisation de logs est-elle prévue ? (presque indispensable pour les architectures SOA ou micro-services). Voir par exemple la solution logstash ;*

- *Quel est le niveau de prolixité prévu par type de composant ? le débat en production est en général entre les niveaux WARN et INFO. Si les développeurs ont bien utilisé le niveau INFO pour des informations pertinentes (environnement au démarrage par exemple) et pas du DEBUG, fixer le niveau INFO.*
- *Des mesures anti-injection de logs sont-elles prévues (échappement XSS) ?*
- *Penser aux sauvegardes des logs au chapitre 10.3.*

Exemple 1 : les logs applicatifs du composant service-miel seront en production de niveau INFO avec roulement journalier et conservation deux mois.

Exemple 2 : les logs seront échappés à leur création via la méthode `StringEscapeUtils.escapeHtml()` de Jakarta commons-lang.

10.7. SUPERVISION

Comme évoqué au chapitre 5.3.1, la supervision est un pilier central de la disponibilité en faisant diminuer drastiquement le MTTD (temps moyen de détection de la panne). Idéalement, elle ne sera pas uniquement réactive mais proactive.

Les métriques sont des mesures brutes (% CPU, taille FS, taille d'un pool...) issues de sondes système, middleware ou applicatives. Les indicateurs sont des combinaisons logiques de plusieurs métriques et sont porteurs de sens (ex : niveau critique si l'utilisation de CPU sur le serveur xxx reste au delà de 95% pendant plus de 5 minutes).

10.7.1 Supervision technique

Lister les métriques :

- *Système (% d'utilisation de file system, load, volume de swap in/out, nombre de threads total ...)*
- *Middleware (% de Used HEAP sur une JVM, nb de threads sur la JVM, % utilisation d'un pool de threads ou de connexions JDBC ..)*

Exemple : on mesura le % de wait io.

10.7.2 Supervision applicative

Lister les métriques applicatifs (développés en interne). Ils peuvent être techniques ou fonctionnels :

- *Nombre de requêtes d'accès à un écran ;*
- *Nombre de contrats traités dans l'heure ;*
- *...*

Il est également possible de mettre en place des outils de BAM (Business Activity Monitoring) basées sur ces métriques pour suivre des indicateurs orientés processus.

Exemple : l'API REST de supervision applicative proposera une ressource `Metrique` contenant les métriques métier principaux : nombre de colis à envoyer, nombre de préparateurs actifs ...

10.7.3 Outil de pilotage de la supervision

Un tel outil (comme Nagios, Hyperic HQ dans le monde Open Source) :

- *collecte les métriques (en SNMP, JMX, HTTP ...) de façon périodique ;*
- *persiste les métriques dans un type de base de données spécifique (comme RRD) ;*
- *consolide les indicateurs depuis les métriques ;*
- *affiche les tendances dans le temps de ces indicateurs ;*
- *permet de fixer des seuils l'alerte basé sur les indicateurs et de notifier les exploitants en cas de dépassement.*

Exemple : la pilotage de la supervision se basera sur la plate-forme Nagios de l'organisation.

10.7.4 Suivi des opérations programmées

Indiquer l'ordonnanceur ou le scheduleur utilisé pour piloter les batchs et consolider le plan de production (exemple : VTOM, JobScheduler, Dollar Universe, Control-M,...). Détailler les éventuelles spécificités du projet :

- *degré de parallélisme des batchs*
- *plages de temps obligatoires*
- *rejeux en cas d'erreur*
- *...*

Exemple : les batchs du projet seront ordonnancés par l'instance JobScheduler de l'organisation.

- *les batchs ne devront jamais tourner les jours féries ;*
- *leur exécution sera bornée aux périodes 23h00 - 06h00. Leur planification devra donc figurer dans cette plage ou ils ne seront pas lancés ;*
- *on ne lancera pas plus de cinq instances du batch XXX en parallèle.*

10.7.5 Supervision des performances

Suit-on les performances de l'application en production ? Cela permet :

- *de disposer d'un retour factuel sur les performances in vivo et d'améliorer la qualité des décisions d'éventuelles redimensionnement de la plate-forme matérielle ;*
- *de détecter les pannes de façon proactive (suite à une chute brutale du nombre de requêtes par exemple) ;*
- *de faire de l'analyse statistique sur l'utilisation des composants ou des services afin de favoriser la prise de décision (pour le décommissionnement d'une application par exemple).*

Il existe trois grandes familles de solutions :

- *les APM (Application Performance Monitoring) : outils qui injectent des sondes sans impact applicatifs, qui les collectent et les restituent (certains reconstituent même les chaînes de liaison complètes via des identifiants de corrélations injectés lors des appels distribués). Exemple : Oracle Enterprise Manager, Oracle Mission Control, Radware, BMC*

APM, Dynatrace , Pinpoint en OpenSource ...). Vérifier que l'overhead de ces solutions est négligeable ou limité et qu'on ne met en péril la stabilité de l'application.

- *la métrologie «maison» par logs si le besoin est modeste ;*
- *les sites de requêtage externes (voir aussi les tests de vie en 10.7.6) qui appellent périodiquement l'application et produisent des dashboards. Ils ont l'avantage de prendre en compte les temps WAN non disponibles via les outils internes. A utiliser couplés aux tests de vie (voir plus loin).*

Exemple : les performances du site seront supervisées en continu par pingdom.com. Des analyses de performances plus poussées seront mises en ouvre par Pinpoint en fonction des besoins.

10.7.6 Test de vie

Il est également fortement souhaitable et peu coûteux de prévoir un système de test de vie (via des scénarios déroulés automatiquement).

En général, ces tests sont simples (requêtes HTTP depuis un curl croné par exemple). Ils doivent être lancés depuis un ou plusieurs sites distants pour détecter les coupures réseaux.

Il est rarement nécessaire qu'ils effectuent des actions en mise à jour. Si tel est le cas, il faut prévoir dans toute l'architecture d'identifier les données issues de ce type de requêtes pour ne pas polluer les données métier et les systèmes décisionnels.

Exemple pour un site Internet : des tests de vie seront mis en œuvre via des requêtes HTTP lancées via l'outil uptrends.com. En cas de panne, un mail est envoyé aux exploitants.

11. DÉCOMMISSIONNEMENT

Ce chapitre est à remplir dès sa conception pour un projet transitoire ou quand le projet arrive en fin de vie et doit être supprimé ou remplacé. Il décrit entre autres :

- *l'architecture d'une bascule complexe avec l'ordre de remplacement des composants, la trajectoire… ;*
- *un éventuel dispositif de retour arrière ;*
- *les données à archiver ou au contraire à détruire avec un haut niveau de confiance ;*
- *les composants physiques à évacuer ou détruire ;*
- *les procédures de désinstallation coté serveur et/ou client (il est courant de voir des composants obsolètes toujours s'exécuter sur des serveurs et occasionner des problèmes de performance et de sécurité passant sous le radar)*
- *les contraintes de sécurité associées au décommissionnement (c'est une étape sensible souvent négligée, on peut retrouver par exemple des disques durs remplis de données très sensibles suite à un don à une association par exemple) ;*

Exemple 1 : Le composant X sera remplacé par les services Y. Ensuite les données Oracle Z du silo seront migrées en one-shot via un script PL/SQL + DBLink vers l'instance XX avec le nouveau format de base du composant T.

Exemple 2 : en cas de problème sur le nouveau composant, un retour arrière sera prévu : les anciennes données seront restaurées dans les deux heures et les nouvelles données depuis la bascule seront reprise par le script S1 qui...

Exemple 3 : Les serveurs X, Y et Z seront transmis au service d'action sociale pour don caritatif après avoir effacer intégralement les disques durs via shred, 3 passes et contrôle humain.

12. ANNEXES

Isoler dans des annexes les informations d'architecture importantes mais concernant des points précis n'intéressant que peu de lecteurs.

12.1. ANNEXE 1

12.2. ANNEXE 2