




Modèle de DA Architecture et urbanisation



Github
L'équipe d'architecture

Document généré le 2022-05-03

Table des matières

| | | |
|----------|---|----------|
| 1 | Modèle de dossier d'architecture | 5 |
| 1.1 | Principes du modèle | 5 |
| 1.2 | Utilisation de ce modèle | 6 |
| 1.2.1 | Présentation générale | 6 |
| 1.2.2 | Conseils sur la rédaction de votre dossier d'architecture | 7 |
| 1.2.3 | Que ne trouve-t-on PAS dans ce document ? | 7 |
| 1.3 | FAQ | 7 |
| 1.4 | Licence | 8 |
| 1.5 | Remerciements | 8 |
| 1.6 | Bibliographie partielle | 8 |
| 1.7 | Termes en cours de discussion / à valider | 8 |
| 1.8 | Termes métier et organisationnels | 9 |
| 1.9 | Termes applicatifs | 9 |
| 1.10 | Termes techniques | 9 |
| 1.11 | Sources | 9 |
| 1.12 | Introduction | 9 |
| 1.12.1 | Documentation de Référence | 10 |
| 1.13 | Non statué | 10 |
| 1.13.1 | Points soumis à étude complémentaire | 10 |
| 1.13.2 | Hypothèses | 10 |
| 1.14 | Contexte général | 10 |
| 1.14.1 | Objectifs | 10 |
| 1.14.2 | Existant | 11 |
| 1.14.3 | Positionnement dans le SI | 11 |
| 1.14.4 | Acteurs | 11 |
| 1.15 | Contraintes | 12 |
| 1.15.1 | Budget | 12 |
| 1.15.2 | Planning | 12 |
| 1.15.3 | Urbanisation | 12 |
| 1.16 | Exigences | 13 |
| 1.17 | Architecture cible | 13 |
| 1.17.1 | Architecture applicative générale | 13 |
| 1.17.2 | Architecture applicative détaillée | 15 |
| 1.17.3 | Principes ayant dicté les choix | 15 |
| 1.17.4 | Vision statique | 15 |
| 1.17.5 | Vision dynamique | 17 |

| | | |
|----------|--|-----------|
| 1.17.6 | Matrice des flux applicatifs | 18 |
| 2 | Volet développement | 20 |
| 2.1 | Introduction | 20 |
| 2.1.1 | Documentation de Référence | 20 |
| 2.2 | Non statué | 20 |
| 2.2.1 | Points soumis à étude complémentaire | 20 |
| 2.2.2 | Hypothèses | 21 |
| 2.3 | Contraintes | 21 |
| 2.4 | Exigences non fonctionnelles | 21 |
| 2.4.1 | Accessibilité | 22 |
| 2.4.2 | Ergonomie | 22 |
| 2.4.3 | Exigences de SEO | 24 |
| 2.5 | Architecture cible | 24 |
| 2.5.1 | Pile logicielle | 24 |
| 2.5.2 | Performances | 25 |
| 2.5.3 | Spécificités d'usine logicielle | 26 |
| 2.5.4 | Normes de développement et qualimétrie | 26 |
| 2.5.5 | Patterns notables | 27 |
| 2.5.6 | Spécificités des tests | 27 |
| 2.6 | Éco-conception | 29 |
| 2.6.1 | Gestion de la robustesse | 30 |
| 2.6.2 | Gestion de la configuration | 31 |
| 2.6.3 | Politique de gestion des branches | 32 |
| 2.6.4 | Versioning | 32 |
| 2.6.5 | Gestion de la concurrence | 32 |
| 2.6.6 | Encodage | 33 |
| 2.6.7 | Fuseaux horaires | 33 |
| 2.6.8 | Gestion des logs | 33 |
| 2.6.9 | Outils d'administration | 35 |
| 2.6.10 | Tri et Pagination | 35 |
| 2.6.11 | Provisioning et mises à jour des DDL | 36 |
| 3 | Volet dimensionnement | 37 |
| 3.1 | Introduction | 37 |
| 3.1.1 | Documentation de Référence | 37 |
| 3.2 | Non statué | 37 |
| 3.2.1 | Points soumis à étude complémentaire | 37 |
| 3.2.2 | Hypothèses | 38 |
| 3.3 | Contraintes | 38 |
| 3.3.1 | Contraintes stockage | 38 |
| 3.3.2 | Contraintes CPU | 38 |
| 3.3.3 | Contraintes mémoire | 38 |
| 3.3.4 | Contraintes réseau | 38 |
| 3.4 | Exigences | 38 |
| 3.4.1 | Volumétrie statique | 39 |
| 3.4.2 | Volumétrie dynamique | 40 |
| 3.4.3 | Exigences de temps de réponse | 43 |

| | | |
|----------|--|-----------|
| 3.5 | Dimensionnement cible | 44 |
| 3.5.1 | Estimation des besoins en ressources par composant technique | 44 |
| 3.5.2 | Dimensionnement des machines | 45 |
| 4 | Volet infrastructure | 47 |
| 4.1 | Introduction | 47 |
| 4.1.1 | Documentation de Référence | 47 |
| 4.2 | Non statué | 47 |
| 4.2.1 | Points soumis à étude complémentaire | 47 |
| 4.2.2 | Hypothèses | 48 |
| 4.3 | Contraintes | 48 |
| 4.3.1 | Contraintes sur la disponibilité | 48 |
| 4.3.2 | Hébergement | 51 |
| 4.3.3 | Contraintes réseau | 52 |
| 4.3.4 | Contraintes de déploiement | 52 |
| 4.3.5 | Contraintes de logs | 52 |
| 4.3.6 | Contraintes de sauvegardes et restaurations | 52 |
| 4.3.7 | Coûts | 52 |
| 4.4 | Exigences | 53 |
| 4.4.1 | Plages de fonctionnement | 53 |
| 4.4.2 | Exigences de disponibilité | 53 |
| 4.4.3 | Modes dégradés | 54 |
| 4.4.4 | Exigences de robustesse | 54 |
| 4.4.5 | Exigences de RPO | 54 |
| 4.4.6 | Exigences d'archivage | 55 |
| 4.4.7 | Exigences de purges | 55 |
| 4.4.8 | Exigences de déploiements et de mise à jour | 55 |
| 4.4.9 | Exigences de gestion de la concurrence | 56 |
| 4.4.10 | Exigences d'écoconception | 57 |
| 4.5 | Architecture cible | 57 |
| 4.5.1 | Principes | 57 |
| 4.5.2 | Disponibilité | 57 |
| 4.5.3 | Déploiement en production | 62 |
| 4.5.4 | Versions des composants d'infrastructure | 63 |
| 4.5.5 | Matrice des flux techniques | 64 |
| 4.5.6 | Environnements | 64 |
| 4.6 | Écoconception | 65 |
| 4.6.1 | Régulation de la charge | 65 |
| 4.6.2 | Gestion des timeouts | 66 |
| 4.6.3 | Exploitation | 67 |
| 4.6.4 | Migration | 72 |
| 4.6.5 | Décommissionnement | 72 |
| 5 | Volet architecture sécurité | 74 |
| 5.1 | Introduction | 74 |
| 5.1.1 | Documentation de Référence | 74 |
| 5.2 | Non statué | 74 |
| 5.2.1 | Points soumis à étude complémentaire | 74 |

| | | |
|--------|---|----|
| 5.2.2 | Hypothèses | 75 |
| 5.3 | Contraintes | 75 |
| 5.4 | Exigences | 75 |
| 5.4.1 | Exigences d'intégrité | 75 |
| 5.4.2 | Exigences de confidentialité | 76 |
| 5.4.3 | Exigences d'identification | 77 |
| 5.4.4 | Exigences d'authentification | 77 |
| 5.4.5 | Exigences de fédération d'identité | 78 |
| 5.4.6 | Exigences de SSO et SLO | 78 |
| 5.4.7 | Exigences de non répudiation | 79 |
| 5.4.8 | Exigences d'anonymat et de respect de la vie privée | 79 |
| 5.4.9 | Exigences sur les habilitations | 80 |
| 5.4.10 | Exigences de traçabilité et d'auditabilité | 81 |
| 5.5 | Mesures de sécurité | 82 |
| 5.5.1 | Intégrité | 82 |
| 5.5.2 | Confidentialité | 83 |
| 5.5.3 | Identification | 84 |
| 5.5.4 | Authentification | 84 |
| 5.5.5 | Fédération d'identité | 85 |
| 5.5.6 | SSO, SLO | 85 |
| 5.5.7 | Non-répudiation | 86 |
| 5.5.8 | Anonymat et vie privée | 86 |
| 5.5.9 | Habilitations | 86 |
| 5.5.10 | Tracabilité, auditabilité | 86 |
| 5.6 | Auto-contrôles | 87 |
| 5.6.1 | Auto-contrôle des vulnérabilités | 87 |
| 5.6.2 | Auto-contrôle RGPD | 88 |

Chapitre 1

Modèle de dossier d'architecture

Ce modèle de Dossier d'Architecture (DA) est applicable à la plupart des projets d'informatique de gestion, indépendamment de l'architecture générale retenue (monolithe, SOA, micro-service, n-tiers, ...). Ce modèle a déjà été utilisé sur plusieurs projets importants y compris au sein de grandes organisations. Il s'enrichit régulièrement.

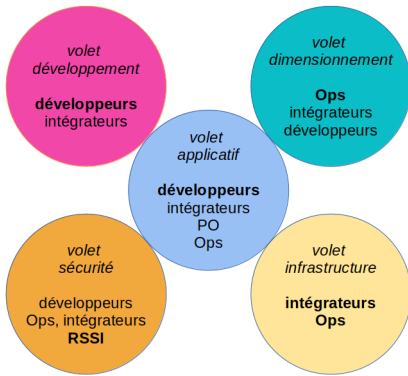
Nous utilisons le terme 'DA' et pas 'DAT' ou termes similaires car chaque organisation possède sa terminologie mais surtout le terme 'technique' (comme 'fonctionnel' d'ailleurs) est équivoque (qu'est ce qui n'est pas "technique" dans l'IT ?).

Autres langues : Anglais.

1.1 Principes du modèle

Nous avons découpé l'architecture en cinq volets (applicatif, sécurité, dimensionnement, infrastructure et développement), **chaque volet étant auto-porteur**.

L'idée est de proposer un **ensemble de vues d'architecture alignées sur les rôles que l'on trouve le plus fréquemment dans les organisations** et sur leurs préoccupations respectives. Par exemple, un architecte d'infrastructure ou un ingénieur DevOps a rarement besoin de connaître le détail de l'architecture logicielle (le détail des frameworks utilisés ou façon de traiter les erreurs). De même, un PO ou un architecte d'entreprise va s'intéresser à la vision macroscopique des modules applicatifs et de leurs interactions principales ("le batch B appelle le service S") mais rarement du détail de l'infrastructure sous-jacente (choix de la base de donnée du service, dimensionnement des machines, ...).



Un dossier suivant ce modèle sera ainsi constitué :

- d'un volet applicatif présentant le contexte général et l'architecture applicative ;
- d'un volet développement présentant l'architecture logicielle et son environnement ;
- d'un volet dimensionnement présentant les aspects liés aux performances et au dimensionnement de l'infrastructure ;
- d'un volet infrastructure présentant les serveurs, les middlewares, l'exploitation, etc. ;
- d'un volet sécurité ;

Dans chaque volet, on retrouvera le triptyque :

- **Contraintes** (juridiques, budgétaires, technologiques, normatives...) applicables au projet;
- **Exigences** non fonctionnelles (ENF) exprimées par les porteurs du projet dans la limite des contraintes ;
- **Solution** (description de l'architecture retenue et répondant aux ENF).

Le dossier comprend également un exemple de glossaire pouvant servir de support à l'Ubiquitous Language, élément fondamental d'une architecture.

Pour en savoir plus, voir ce papier publié dans Applied Computing and Informatics.

1.2 Utilisation de ce modèle

1.2.1 Présentation générale

- Ce modèle est au format asciidoc. Vous pouvez le convertir dans le format de votre choix pour vos DA même si nous préconisons un **format textuel et lisible** (type Markdown) facile à suivre et à modifier par merge requests dans un outil de gestion de version et ainsi transformer votre DA en une documentation vivante ;
- Ce modèle est perfectible, c'est pourquoi tous les retours, critiques (constructives), contributions et suggestions sont appréciés (faire une pull request ou utiliser les issues) ;
- De plus, il est volontairement riche en **explications et exemples** car il a également une (modeste) prétention éducative à destination des étudiants et jeunes architectes.
- Le texte en italique contient des exemples ;
- Chaque chapitre dispose de notes aidant à le remplir ;
- Des modèles vierges (sans exemples) sont fournis pour votre confort. **Il est fortement préconisé de partir des modèles vierges tout en ayant dans une autre fenêtre le modèle avec exemples et explications sous les yeux ;**

- Le script `export` joint permet de produire une archive avec tout votre dossier au format HTML et/ou PDF.

1.2.2 Conseils sur la rédaction de votre dossier d'architecture

- **Rester bref**, chaque mot doit avoir son utilité. 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 de l'application sans être noyé de détails. Le document doit **rester maintenable et à jour** ;
- Si l'application suit une architecture standardisée par l'organisation, **ne jamais la répéter** (principe DRY) et se référer à un document commun ;
- 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 ou TODO s'il reste à compléter ;
- Ce modèle se veut **suffisamment complet pour couvrir la plupart des applications**. Il est donc normal que de nombreux chapitres ne soient pas applicables dans votre contexte ;
- Lister les **hypothèses d'architecture** et études en cours dans le chapitre "Points non statués" de chaque volet (leur nombre doit être limité, sinon c'est le signe que le DA est rédigé trop tôt) ;
- **Isoler dans des annexes** en fin de document les informations d'architecture importantes mais concernant des points précis n'intéressant que peu de lecteurs ;

1.2.3 Que ne trouve-t-on PAS dans ce document ?

- la **conception détaillée** du projet (diagrammes UML de classes, de séquences...) sauf pour présenter un pattern général spécifique à l'application ;
- des éléments d'**études** (SWOT, scénarios...) : les choix doivent déjà avoir été faits (en revanche, nous préconisons à cette fin l'écriture d'ADRs en annexe du DA) ;
- l'**urbanisation du SI** (nous nous positionnons ici au niveau d'une application ou d'un ensemble de composants cohérents) ;
- les **règles d'architecture de référence** (communes à toutes les applications) ;
- des détails techniques (IP, logins) pouvant compromettre la sécurité ;
- l'**architecture physique** (détails des serveurs et datacenters, architecture réseau, architecture de stockage, provisioning...). Il s'agit de sujets très pointus et en général traités par les architectes d'infrastructure à un niveau SI ;
- le détail des **environnements** autres que la production (recette, développement...). Ces derniers sont en général trop fluctuants pour figurer dans ce dossier et gagneront à plutôt être documentés par l'intégrateur dans d'autres dossiers, fiches, wikis ou outils de CMDB.

1.3 FAQ

- **A partir de quelle taille de projet ce modèle est-il éligible ?** Ce modèle a été utilisé avec succès sur un projet d'une seule personne. Il peut être utilisé pour toute taille de projet. Un grand projet remplira probablement plus de rubriques mais la plupart concernent tous les projets. Par exemple, les questions de disponibilité ou d'internationalisation ne sont pas liées à la taille d'un projet nous semble-il.
- **Ce modèle peut-il servir de base à un référentiel d'architecture ?** Même si de nombreuses idées peuvent être reprises, non, ce n'est pas l'objet de ce modèle.

- **Ce modèle convient-il à un programme complet ?** Nous préconisons pour un programme complet une approche type TOGAF avec les livrables associés. En revanche, les phases C et D pourront être documentées par un DA au sein de chaque projet de ce programme.

1.4 Licence

- Copyright (c) 2017-2021 Bertrand Florat et contributeurs
- Ce modèle est en licence CC BY-SA 4.0 : Creative Commons Attribution - Partage à l'identique V4.0
- Vous pouvez créer votre propre modèle à condition qu'il conserve la licence CC BY-SA 4.0 et qu'il contienne donc ces trois éléments:
 - Le nom du créateur (Bertrand Florat) ;
 - Un lien vers <https://creativecommons.org/licenses/by-sa/4.0/> ;
 - Une notice de non-responsabilité et un lien vers <https://github.com/bflorat/modele-da>.
- Les dossiers d'architecture issus de ce modèle n'ont pas à appliquer cette licence. Il est néanmoins recommandé d'y inclure un lien vers <https://github.com/bflorat/modele-da>.

1.5 Remerciements

- Contributeurs/eurs
- Relecture : Frédérique Lefranc
- Retours : Antoine Parra Del Pozo, Pascal Bousquet, Philippe Mayjonade, Nicolas Chahwekilian, Steven Morvan, Dr. Christophe Gaie
- Tous les diagrammes de ce modèle ont été générés avec l'excellent outil PlantUML Les diagrammes C4 utilisent la personnalisation C4 de plantuml.

1.6 Bibliographie partielle

- *Site Reliability Engineering* - Google
- *Living documentation* - Cyril Martraire
- *Clean Code* - Robert Martin
- *Performance des architectures IT - 2e ed.* - Pascal Grojean
- *Design Patterns: Elements of Reusable Object-Oriented Software* de Erich Gamma, Richard Helm, Ralph Johnson et John Vlissides (GOF)
- *Le projet d'Urbanisation du SI* - Christophe Longépé
- *Sécurité de la dématérialisation* - Dimitri Mouton ## Glossaire des termes utilisés

Merci de respecter l'ordre alphabétique lors des éditions du document

1.7 Termes en cours de discussion / à valider

| Terme | Signification | Commentaire |
|--------------|------------------------------|------------------------------|
| <i>Canal</i> | <i>Origine de la demande</i> | <i>Canal ou provenance ?</i> |

1.8 Termes métier et organisationnels

| Terme | Signification | Commentaire |
|-------|---------------------|---|
| CA | <i>Cour d'appel</i> | <i>"Alors que les juridictions de première instance rendent un « jugement », une cour d'appel rend un « arrêt »"</i> [Wikipedia] |

1.9 Termes applicatifs

| Terme | Signification | Commentaire |
|-------|--|--|
| EIDAS | <i>Electronic IDentification Authentication and trust Services</i> | <i>Règlement européen sur la signature électronique</i> |
| SPA | <i>Single Page Application</i> | <i>Application Web dont les pages sont générées en javascript dans le navigateur</i> |

1.10 Termes techniques

| Terme | Signification | Commentaire |
|-------|----------------------|--|
| RP | <i>Reverse Proxy</i> | <i>Mantataire inverse effectuant des appels HTTP pour le compte d'un client (en général sur Internet). Permet entre autres d'augmenter la sécurité par rupture protocolaire.</i> |

1.11 Sources

— <http://km.mondomaine.fr#> Volet architecture applicative :sectnumlevels: 4 :toclevels: 4 :sectnums: 4 :toc: left :icons: font :toc-title: Sommaire

Dernière modification : 2022-05-03

1.12 Introduction

Ceci est le point de vue applicatif du projet. Il décrit les modules applicatifs en jeu et leurs échanges.

Les autres volets du dossier sont accessibles d'ici.

Le glossaire du projet est disponible ici. Nous ne redéfinirons pas ici les termes fonctionnels ou techniques utilisés.

1.12.1 Documentation de Référence

Mentionner ici les documents d'architecture de référence (mutualisés). Ce dossier ne doit en aucun cas reprendre leur contenu sous peine de devenir rapidement obsolète et inmaintenable.

TABLE 1.5: Références documentaires

| N° | Version | Titre/URL du document | Détail |
|----|---------|-----------------------|-----------|
| 1 | 2.0.4 | XX_Urba_POS.pdf | POS du SI |

1.13 Non statué

1.13.1 Points soumis à étude complémentaire

TABLE 1.6: Points soumis à étude complémentaire

| Sujet | Détail | Statut | Porteur du sujet | Échéance |
|----------------------------|---|------------|------------------|------------|
| Utilisation des services Y | En fonction de l'avancement du projet Y, ce composant pourrait appeler les services de ce dernier ou ceux de l'ancien composant Z | EN_ATTENTE | Projet Y | AVANT 2040 |

1.13.2 Hypothèses

TABLE 1.7: Hypothèses

| ID | Détail |
|-----|--|
| HA1 | Même si la décision de généralisation de l'annuaire centralisé n'est pas totalement entérinée, l'application s'appuiera dessus et non sur un annuaire local. |

1.14 Contexte général

1.14.1 Objectifs

Décrire succinctement le projet et en rappeler les objectifs. Mettre en évidence ceux qui sont structurants pour l'architecture.

Exemple 1 : Cette application 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 X. Elle doit en faciliter la maintenance.

Exemple 3: l' application X est l'un des composants principaux du programme Y. Il s'adosse sur les référentiels Personne et Facturation pour enrichir le CMS en données clients temps réel.

1.14.2 Existant

Si ce document présente un projet de refonte ou migration, décrire a minima l'application existante. Ne pas reprendre la documentation, y faire simplement référence et pointer vers son éventuel dossier d'architecture. Mentionner néanmoins toute information ayant un impact fort sur la migration ou la conception du nouveau projet.

Exemple 1 : L'application VENIR2 est une application Client-Server en FORMS 4 pointant vers une base Oracle 9i. Son dossier d'architecture est donné en [REFxyz].

Exemple 2 : L'application existante se base et alimente un annuaire LDAP pour ses autorisations. Le nouveau projet devant fonctionner un temps avec l'ancienne, il convient de prendre en compte les accès concurrents et la cohérence du LDAP pendant la période de tuilage.

1.14.3 Positionnement dans le SI

Si le SI est urbanisé, reprendre le plan d'occupation au sol et préciser le bloc concerné

1.14.4 Acteurs

1.14.4.1 Acteurs internes

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

TABLE 1.8: Liste des acteurs internes

| Acteur | Description | Population | Localisation |
|--------------------------------------|---|------------|-----------------------|
| <i>Système de l'administration B</i> | <i>fournit les données comptables des entreprises</i> | <i>N/A</i> | <i>Site de Berlin</i> |
| <i>Agent</i> | <i>Agent back-office</i> | <i>100</i> | <i>Site de Paris</i> |

1.14.4.2 Acteurs externes

TABLE 1.9: Liste acteurs externes

| Acteur | Description | Population | Localisation |
|-------------------|------------------------------------|---------------|--|
| <i>Client Web</i> | <i>Une entreprise depuis un PC</i> | <i>Max 1M</i> | <i>10 appels à l'IHM par session, une session par jour et par acteur</i> |

| Acteur | Description | Population | Localisation |
|----------------------|--|---------------|---------------------|
| <i>Client mobile</i> | <i>Une entreprise depuis un mobile</i> | <i>Max 2M</i> | <i>Monde entier</i> |

1.15 Contraintes

1.15.1 Budget

Donner les contraintes budgétaires du projet

Exemple 1: Enveloppe globale de 1 M€

Exemple 2: Coûts d'infrastructure cloud < 20K€ / mois

1.15.2 Planning

Sans reprendre dans le détail les plannings du projet, donner les éléments intéressants pour l'architecture.

Exemple 1: MEP avant fev 2034, prérequis au programme HEAVY en mai 2034.

1.15.3 Urbanisation

Lister ici les contraintes relatives à l'urbanisation, ceci inclut par exemple mais pas seulement :

- Les règles applicables dans les appels entre composants (SOA)
- Les règles d'appels entre zones réseau
- Les règles concernant la localisation des données (MDM)
- Les règles concernant la propagation des mises à jours par événements (EDA)

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

Exemple 2 : pour en assurer la fraîcheur, il est interdit de répliquer les données du référentiel PERSONNE. Ce dernier devra être interrogé au besoin en synchrone.

Exemple 3 : Lors de la modification d'une commande, les zones comptabilité et facturation seront mises à jour de façon asynchrone via un événement.

Exemple 4 : tous les batchs doivent pouvoir fonctionner en concurrence des IHM sans verrouillage des ressources.

Exemple 5 : les services ne peuvent être appelés directement. Les appels se feront obligatoirement via une route exposée 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 6 : Les composants de cette application suivent l'architecture SOA telle que définie dans le document de référence X.

Exemple 7 : Les composants en zone Internet ne peuvent appeler les composants en zone Intranet pour des raisons de sécurité.

1.16 Exigences

Donner ici les exigences d'architecture applicative pouvant s'appliquer au projet.

Exemple 1 (projet de migration) : Les modules legacy devront faire l'objet d'aussi peu d'adaptations que possible.

Exemple 2 : Les modules devront pouvoir s'interfacer avec le partenaire XYZ via leurs API.

Exemple 3 : Le développement devra pouvoir se faire au sein d'équipes distribuées, chacune travaillant sur des modules distincts.

1.17 Architecture cible

1.17.1 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.

Rappeler :

- Le type d'architecture (client-serveur, Web monolithique, SOA, micro-service...).
- Les grands flux entre les composants ou entre les applications dans le cas des monolithes.
- D'éventuelles dérogations aux règles d'architecture du SI.

Le choix de la représentation est libre mais un diagramme C4 de System Landscape ou 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'entreprise qui ont un rôle applicatif (orchestration de service par exemple).

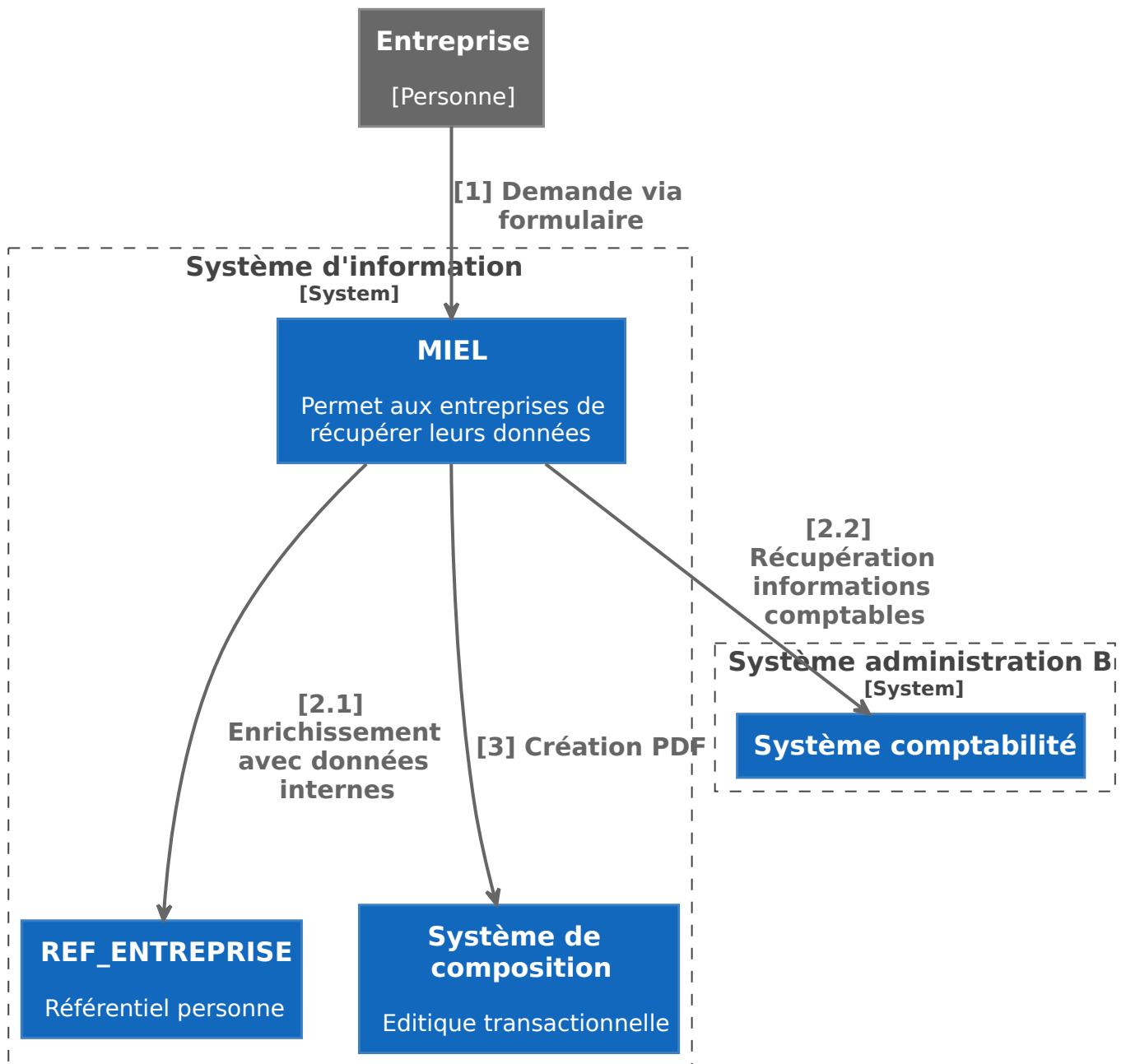
Exemple 1 : MesInfosEnLigne permet à une entreprise de récupérer par mail un document récapitulant toutes les informations dont l'administration dispose sur elle. L'administration peut compléter ses données par celles d'une autre administration.

Exemple 2 : MesInfosEnLigne est constituée de plusieurs microservices indépendants (composants IHM, batchs ou services REST)

Exemple 3 : Suite à la dérogation du DSI le 03 aout 20xx, l'IHM sera en architecture SPA (Single Page Application)

Diagramme de contexte système
Point de vue applicatif général
du projet Mes Infos En Ligne (MIEL)

archi-applicative-generale



| | Type |
|--|-----------------|
| | person |
| | external person |
| | system |
| | external system |
| | container |

1.17.2 Architecture applicative détaillée

Détailler ici tous les composants de l'application, leurs flux entre eux et avec les autres applications du SI.

Proposer un ou plusieurs schémas (de préférence des diagrammes C4 de type containers ou diagramme UML2 de composant).

Idéalement, le schéma tiendra sur une page A4, sera autoporteur et compréhensible par un non-technicien. Il devrait devenir l'un des artefacts documentaires les plus importants et figurer dans la war room d'un projet agile ou être imprimé par chaque développeur.

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 volet infrastructure).

Pour chaque flux, donner le protocole, un attribut synchrone/asynchrone, un attribut lecture/écriture/exécution et une description pour que le schéma soit auto-porteur.

1.17.3 Principes ayant dicté les choix

Donner ici l'intention dans la construction de l'architecture.

Exemple : nous utiliserons une approche monolithique et non micro-service par manque d'expertise.

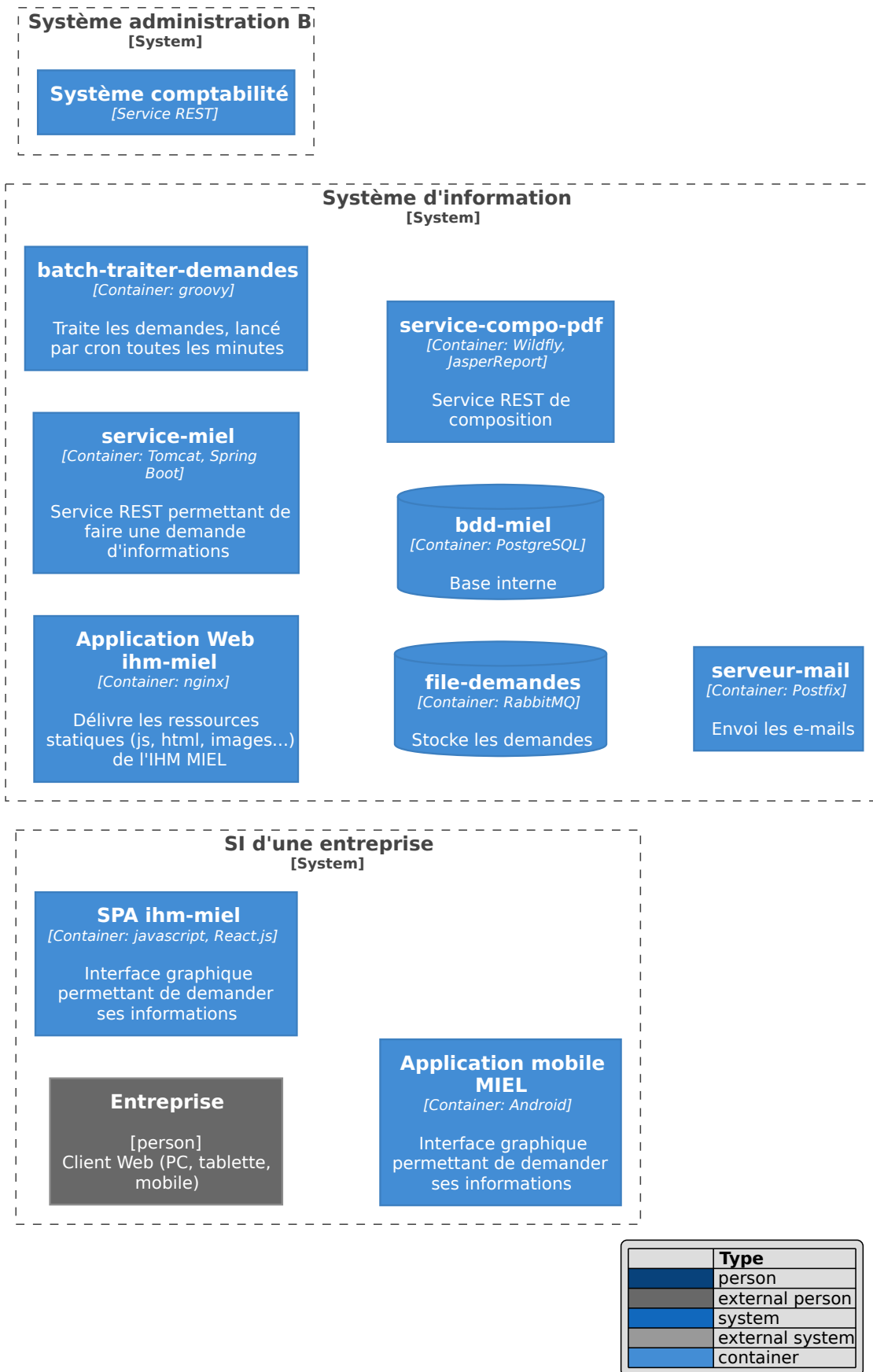
1.17.4 Vision statique

Exposer les modules applicatifs dans leurs différentes zones ou domaines.

Exemple: module X, Y et Z dans le domaine GED. Modules A, B dans le domaine PERSONNE.

Diagramme de conteneurs
Point de vue applicatif détaillé vue statique
Projet Mes Infos En Ligne (MIEL)

archi-applicative-detaillée-statique



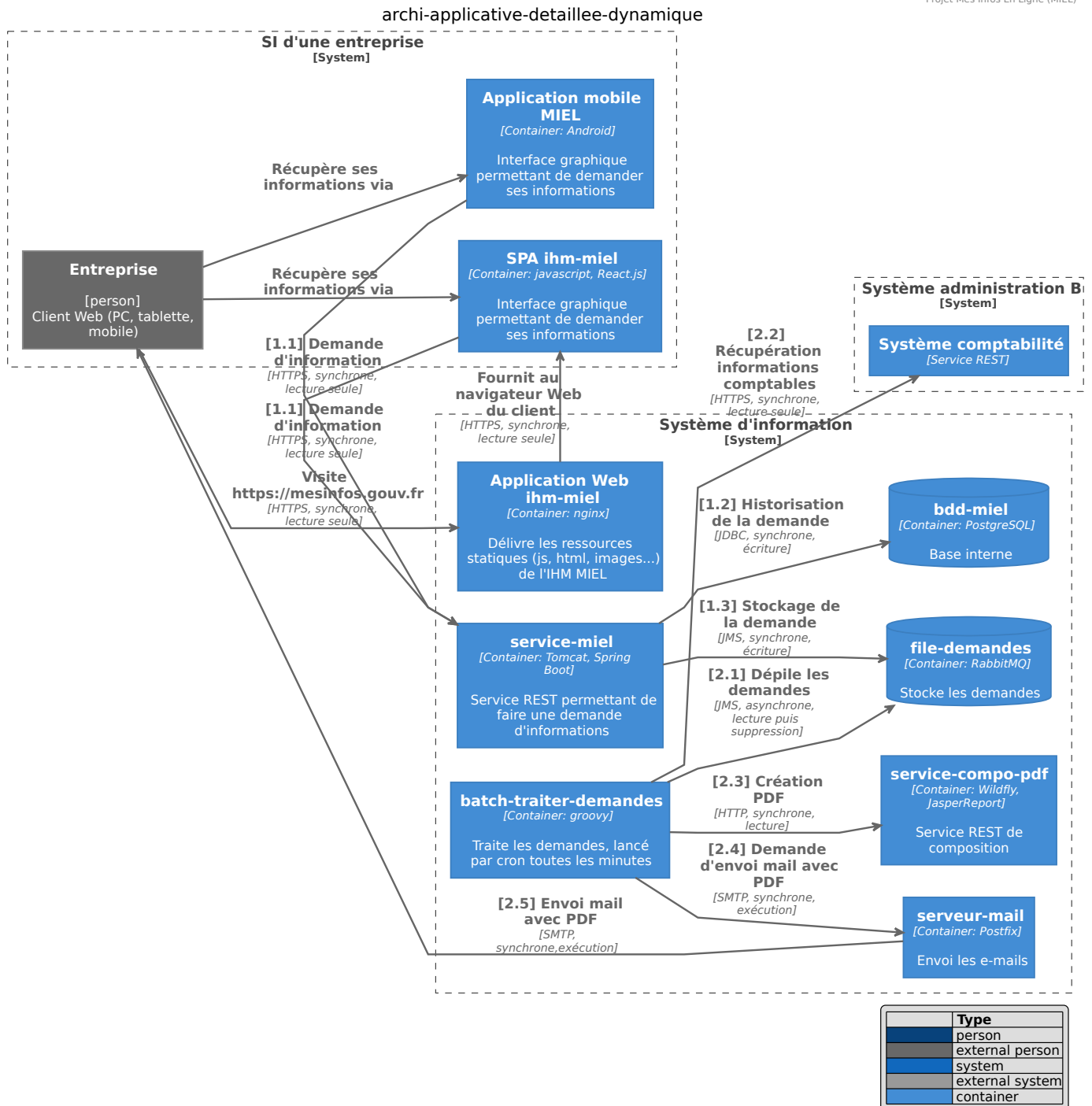
1.17.5 Vision dynamique

Exposer les modules applicatifs dans leurs différentes zones ou domaines avec leurs flux applicatifs principaux.

Ne pas détailler les flux techniques (comme les flux liés à la supervision ou au clustering).

Si l'application est complexe, proposer un schéma global exposant tous les flux applicatifs puis un schéma par chaîne de liaison principale en numérotant les échanges (utiliser un diagramme de séquence ou (mieux) un Dynamic Diagram C4). Il est possible également de détailler les chaînes de liaison par fonctionnalité principale.

Exemple:



1.17.6 Matrice des flux applicatifs

Lister ici les flux principaux de l'application.

Ne pas détailler les flux techniques de supervision ou liés au clustering par exemple. Mentionner le type de réseau (LAN, WAN).

TABLE 1.10: Exemple partiel de matrice de flux applicatifs

| Source | Destination | Type de réseau | Protocole | Mode. ¹ |
|------------------------------|-----------------------------------|-----------------|------------|--------------------|
| <i>Entreprise</i> | <i>PC/tablette/mobile externe</i> | <i>ihm-miel</i> | <i>WAN</i> | <i>LE</i> |
| <i>batch-traiter-demands</i> | <i>service-compo-pdf</i> | <i>HTTP</i> | <i>LAN</i> | <i>A</i> |

1. (L)ecture, (E)criture ou Lecture/Ecriture (LE), (A)ppel (vers un système stateless)

Chapitre 2

Volet développement

Dernière modification : 2022-05-03

2.1 Introduction

Ceci est le point de vue développement de l'application. Il décrit le code à produire et comment l'écrire.

Les autres volets du dossier sont accessibles d'ici.

Le glossaire du projet est disponible ici. Nous ne redéfinirons pas ici les termes fonctionnels ou techniques utilisés.

2.1.1 Documentation de Référence

Mentionner ici les documents d'architecture de référence (mutualisés). Ce document ne doit en aucun cas reprendre leur contenu sous peine de devenir rapidement obsolète et inmaintenable.

TABLE 2.1: Références documentaires développement

| N° | Version | Titre/URL du document | Détail |
|----|---------|---|--------|
| 1 | 1.0 | https://references.modernisation.gouv.fr/rgaa-accessibilite/#menu | RGAA |

2.2 Non statué

2.2.1 Points soumis à étude complémentaire

TABLE 2.2: Points soumis à étude complémentaire

| ID | Détail | Statut | Porteur du sujet | Échéance |
|-----|---|----------|---------------------------|------------|
| ED1 | <i>Le choix Angular ou React.JS pour le frontend est encore soumis à étude. Ceci n'impacte pas la partie back des services REST</i> | EN_COURS | Equipe méthodes et outils | AVANT 2040 |

2.2.2 Hypothèses

TABLE 2.3: Hypothèses

| ID | Détail |
|-----|---|
| HD1 | <i>Même si ce point n'est pas encore totalement validé, l'application nécessitera un JRE 9 + pour tirer profil de librairies et frameworks Java indispensables au projet.</i> |

2.3 Contraintes

Lister ici les contraintes relatives à l'architecture logicielle, ceci inclut par exemple mais pas seulement :

- L'obligation d'utiliser un framework ou une filière technologique précise
- Les budgets maximums de licence ou de développement
- L'outillage (IDE, ...)
- L'intégration continue
- Les normes et seuils de qualité de code applicables
- Les tests (taux de couverture, répartition par type de test, ...)

Exemple 1 : La couverture de code devra être d'au moins 60%

Exemple 2 : Le module devra se baser sur le framework Hibernate pour la persistance et CDI pour l'injection de dépendance

Exemple 3 : l'application sera construite, testée et déployée en continu à chaque push via la plateforme Gitlab-ci

2.4 Exigences non fonctionnelles

Contrairement aux contraintes qui fixaient le cadre auquel toute application devait se conformer, les exigences non fonctionnelles sont données par les porteurs du projet (MOA en général). Prévoir des interviews pour les déterminer. Si certaines exigences ne sont pas réalistes, le mentionner dans le référentiel des points à statuer.

2.4.1 Accessibilité

Cette application doit-elle être accessible aux non/mal voyants ? malentendants ?

Si oui, quelle niveau d'accessibilité ? Se référer de préférence au Référentiel Général d'Accessibilité (RGAA) qui préconise un niveau WCAG 2.0 AA :

Il existe d'autres normes d'accessibilité (WCAG, AccessiWeb ...) . Attention à correctement évaluer le niveau visé (ni sur-qualité, ni sous-qualité) :

- Atteindre un niveau d'accessibilité très élevé peut être coûteux et contraignant technologiquement. Il demande également de bonnes compétences (accessibilité, HTML5/CSS3 en particulier) et des profils rares.
- 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 ».

2.4.2 Ergonomie

2.4.2.1 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.

2.4.2.2 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). 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

2.4.2.3 Polices de caractère

Décrire ici les polices de caractère à utiliser pour les pages Web, les applications ou les documents composés.

Le choix des polices suit des contraintes de licences. Afin d'assurer une sécurité juridique au projet, attention aux polices commerciales soumises à royalties (en particulier les polices appartenant à Microsoft comme Times New Roman, Courier, Verdana, Arial) et qui ne permettent pas de produire gratuitement des documents sans passer par leurs éditeurs (Word, ...).

Voir par exemple la police Marianne préconisée par le gouvernement en tant que police à chasse variable.

Redhat propose quatre familles de polices Liberation Mono en licence Open Source sécurisante sur un plan juridique et compatible métriquement avec le Monotype, le Courier New, l'Arial et le Times New Roman.

2.4.2.4 Site Web adaptatif

Lister les contraintes d’affichage multi-support. Utiliser quand c’est possible les frameworks modernes (type AngularJS ou React.js). Il existe plusieurs niveaux d’adaptation des pages Web :

- Statique (largeur de page fixe).
- Dynamique (redimensionnement automatique, les tailles sont exprimées en %).
- Adaptatif (les distances sont exprimées en unités dont la taille dépend du support).
- Responsive (le contenu et son agencement dépend du support).

Un design responsive vient avec ses contraintes (duplication de code, augmentation du volume du site à télécharger par le client, complexité, plus de tests end-to-end à prévoir...).

2.4.2.5 Progressive Web Apps (PWA)

Spécifier si l’application est progressive. Les applications PWA sont des applications Web HTML5 possédant tous les attributs des applications natives (mode déconnecté, rapide, adaptatif, accessible depuis l’OS, ...)

Exemple : L’application X sera totalement PWA. Des tests devront démontrer que le site continuer à fonctionner sans réseau et que les pages se chargent en moins de 5 secs en 4G.

2.4.2.6 Navigateurs supportés

Préciser quels sont les navigateurs supportés si votre projet contient une IHM Web.

Lorsqu’on s’adresse à un public dont on ne gère pas le parc de navigateurs (comme un site Web sur Internet), la meilleure option pour rendre les choses intelligibles et expliciter les enjeux est de négocier avec les parties prenantes du projet un pourcentage de public supporté en se basant sur des statistiques. Par exemple : "Support de 95 % des navigateurs".

Supporter d’anciens navigateur (IE en particulier) peut engendrer des surcoûts rédhibitoires et des risques sur la sécurité. Dans tous les cas, il convient d’évaluer les surcoûts de tester sur plusieurs plateformes. 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 X devra fonctionner sur les navigateurs qualifiés en interne (cf norme xyz)

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

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

2.4.2.7 Internationalisation (i18n)

Préciser les contraintes de l’application en terme d’i18n : localisation des libellés, direction du texte, mise en page adaptable, code couleur spécifique, format de dates, devises, affichage des séparateurs

décimaux, etc.

Exemple 1 : L'IHM X sera traduite en 25 langues dont certaines langues asiatiques 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.

2.4.2.8 Mode déconnecté

Préciser si l'application doit pouvoir continuer à fonctionner sans accès à Internet ou au LAN (très courant pour les applications utilisées par les professionnels en déplacement par exemple).

Il peut s'agir de clients lourds classiques (Java, C, ...) possédant leur base locale pouvant être synchronisée de retour au bureau. Il peut également s'agir d'applications PWA (voir plus haut) utilisant un service worker pour les ressources statiques et du stockage navigateur (local storage, base de données IndexedDB).

Exemple 1 : L'application sera développée en Java Swing avec stockage local basé sur une base H2 synchronisées avec la base commune par appels REST.

Exemple 2 : L'application mobile sera en mode PWA, entièrement écrite en HTML5 avec local storage pour stocker les données de la journée dans le navigateur.

2.4.3 Exigences de SEO

Le SEO (Search engine optimization) concerne la visibilité d'un site Web au travers des moteurs de recherches (comme Google ou Quant).

Exemple 1 : Aucune indexation nécessaire ni désirée (site interne)

Exemple 2 : Les pages statiques du site devront suivre les bonnes pratiques SEO pour optimiser sa visibilité.

2.5 Architecture cible

2.5.1 Pile logicielle

2.5.1.1 Filière technique retenue

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

Exemple : cette application est de profil P3 : "Application Web Spring" avec utilisation exceptionnelle de la librairie JasperReport.

Exemple : Utilisation de Reacts.js à titre expérimental au sein de l'organisation. Validé en comité architecture le ...

2.5.1.2 Composants logiciels

Lister ici pour chaque composant les principales librairies et frameworks utilisés ainsi que leur version. Ne pas lister les librairies fournies au runtime par les serveurs d'application ou les frameworks. Inutile

de trop détailler, donner uniquement les composants structurants.

Exemple :

TABLE 2.4: Exemple de pile logicielle

| Librairie | Rôle | Version |
|---------------------------|--|--------------|
| <i>Framework Angular2</i> | <i>Framework JS de présentation</i> | <i>2.1.1</i> |
| <i>JasperReport</i> | <i>Editique transactionnelle, composition des factures au format PDF</i> | <i>6.3.0</i> |

2.5.2 Performances

Voir les exigences sont dans le volet dimensionnement.

Même si des campagnes de performance sont prévues, l'expérience montre que la plupart des problèmes de performance auraient pu être détectés dès le développement. Il est donc important que les développeurs profilent leur code, dès leur poste de travail (à prévoir dans le Definition Of Done du projet). Il ne sera pas possible de détecter tous les problèmes (scalabilité, concurrence, robustesse, tuning des caches, ...) mais la plupart des problèmes de temps de réponse. Il est également souvent possible de simuler de la concurrence et de la charge. Nous présentons ici quelques pistes très basiques et à la portée de tout développeur.

Coté Frontend :

- Limiter la complexité des CSS (sélecteurs ou fonctions en particulier)
- Utiliser un profiler (comme celui de Chrome)
- Privilégier les appels asynchrones
- ...

Coté Backend :

- S'assurer que la pagination serveur va bien jusqu'à la base de donnée (utiliser `FETCH FIRST x ROWS ONLY` mais pas `LIMIT` and `OFFSET`).
- Ne pas mettre en place de contraintes inutiles en base de données.
- Limiter le nombre de jointures et les relations many-to-many.
- Dans des cas de grosses volumétries, étudier les solutions de partitionnement de tables.
- Ne pas oublier d'ajouter tous les index nécessaires, utiliser l'analyse du plan d'exécution pour vérifier qu'il n'y a pas de full scans.
- Attention aux fonctions SQL qui 'cassent' les index (comme `UPPER()`). Privilégier les traitements coté code backend si possible.
- Activer les logs de requêtes (exemple Hibernate : `org.hibernate.SQL=DEBUG,-Dhibernate.generate_statistics`) et vérifier les équêtes SQL et leur nombre (pour détecter en particulier le problème du `SELECT N+1`, très courant).
- Disposer même sur poste de travail d'un jeu de donnée minimal (une centaine d'enregistrement).
- Vérifier avec un profiler (comme JVisualVM en Java) la consommation mémoire pour détecter les fuites ou les surconsommations.
- Vérifier qu'il n'y a pas de fuite de threads ou de deadlocks en comptant le nombre de threads

actifs sur une période suffisamment longue (une nuit complète par exemple).

- Stresser les API *a minima* (avec des injecteurs comme JMeter ou K6) et via une rampe progressive.
- Traquer les IO (des millions de fois plus lents que des accès mémoire).
- ...

Frontend et backend :

- Toute ressource (taille de chaîne, nombre d'appel sur une durée, ...) doit systématiquement être bornée par une limite (pas d'open bar).
- Vérifier que la taille des requêtes HTTP reste en dessous de quelques dizaines de Kio (hors GET sur fichiers). Utiliser la pagination cliente et serveur.
- Traquer le bavardage réseau : grouper les requêtes quand possible (il faut trouver un compromis avec la règle précédente). S'aider de la règle 'I' de SOLID (Interface Segregation).
- Prévoir des endpoints multivalués (exemple: `GET /personnes?list=id1,id2...`) pour récupérer plusieurs éléments à la fois (doit se concrétiser par un seul `SELECT WHERE .. IN` dans la requête finale, pas une boucle dans le code !)

Ne pas tomber à l'inverse dans l'optimisation prématurée "source de tous les problèmes" selon Donald Knuth. Écrire le code le plus simple possible et suivre un bon design, ne l'optimiser qu'ensuite. N'optimiser que si cela vaut le coût (loi de Pareto). Commencer par les optimisations les plus significatives et ne pas perdre son temps à grappiller des microsecondes voire nanosecondes.

2.5.3 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, l'image Docker est relasée dans Nexus.

2.5.4 Normes de développement et qualimétrie

Rendre explicite les règles et le niveau de qualité requis pour le code

Exemple 1 : Les règles de qualité à utiliser pour le code seront (les règles standards SonarQube pour Java).

Exemple 2 : Le niveau de qualité exigé correspond au Quality Gate SonarQube recommandé :

- 80% de couverture de code minimum
- 3 % max de lignes dupliquées
- Niveau A en Maintainability, Reliability et Security

Exemple 3 : Quelle langue utilisée pour le code ? français pour les termes fonctionnels (il est impératif d'utiliser les termes métiers comme préconisé par le DDD) et l'anglais pour les termes techniques génériques.

2.5.5 Patterns notables

Préciser si ce projet a mis en œuvre des patterns (GoF, JEE ou autre) structurants. 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>.

2.5.6 Spécificités des tests

Une méthodologie ou une technologie particulière est-elle en jeu dans ce projet ? Quelle est la stratégie de tests ?

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)

Exemple 3 : Types de tests

TABLE 2.5: Types de tests

| Type de test | Temps à investir | Manuel ou automatisé ? | Type de module ciblé | Taux de Couverture visée | Détail |
|-----------------------------------|------------------|------------------------|----------------------|---|--|
| TU | Très élevé | Automatisé | Backend et Frontend | env. 80% | Format BDD : spécifications de comportements des classes et méthodes |
| Spécifications exécutables | Très élevé | Automatisé | Api | env. 100% pour les classes du domaine | Mode bouchonné. |
| Tests de contrats | Faible | Automatisé | Liens UI/API | env. 100% du code appelant coté UI et des contrôleurs Spring coté API | Teste la non régression des échanges lors de l'appel des opérations des API REST (principe CDC=Consumer-Driven Contract) via les outils Pact et pact-react-consumer. |

| Type de test | Temps à investir | Manuel ou automatisé ? | Type de module ciblé | Taux de Couverture visée | Détail |
|---------------------------------|--|------------------------|--|--|---|
| Tests d'architecture | Très faible | Automatisé | API et batchs | N/A, 100% du code est validé par l'outil | En particulier, ces tests simples à écrire vérifieront le respect des règles de l'architecture hexagonale. Utilisation du framework de test ArchUnit. |
| TI (tests d'intégration) | Faible | Automatisé | Composants appelant des systèmes externes (bases de données, API...) | 50 à 60% | Chaque TI ne doit tester qu'un seul système externe à la fois |
| E2E (tests bout en bout) | Faible | Automatisé | UI | 30%, cas nominaux (happy path) | Ecrits en CodeceptJS, Selenium ou technologie similaire. Ils seront limités à un rôle de smoke tests (détection de problèmes grossiers). Ces tests ne seront pas bouchonnés mais seront effectués sur chaîne de liaison instanciée de bout en bout. Pour éviter le travail inutile, ces tests seront faits au niveau de features entières, pas forcément à chaque sprint. Ces tests feront office également de tests système puisqu'ils solliciteront un maximum de modules débouchonnés. |
| Tests de performance | Faible (hors campagnes de performance dédiées) | Automatisé | API critiques | 20% | Possiblement automatisés en CI en DEV mais également lancé manuellement par les développeurs |

| Type de test | Temps à investir | Manuel ou automatisé ? | Type de module ciblé | Taux de Couverture visée | Détail |
|-------------------------------|------------------|------------------------|----------------------------|--|---|
| Tests d'accessibilité | Moyenne | Automatisé + manuel | UI | 50% | Tests Axe-Core lancés en CI à compléter d'un audit manuel |
| Tests de sécurité | Moyenne | Manuel | Tous | Faible, uniquement sur les fonctions sensibles | Audit à prévoir |
| Tests système | Faible | Manuels | UI et batchs | 10% | Tests menés par l'équipe de développement couvrant des scénarios fonctionnels complets. Le but est ici de tester le fonctionnement de l'ensemble des modules (ce qui n'est pas automatisable) et de détecter un maximum de bugs avant les tests d'UAT. |
| Tests UAT (acceptance) | Moyenne | Manuels | UI, batchs lancé à la main | de 30% à 80% selon le nombre de scénarios prévus | Tests menés en recette par des utilisateurs finaux sur environnement non bouchonné avec des cahiers de tests. Tests d'acceptance de bout n bout (on suit un cahier de tests avec les cas nominaux), Tests exploratoires (on tente toutes les combinaisons possibles avec un guidage minimal dans le cahier de test) |

Pour un projet d'envergure, la stratégie de test fait en général l'objet d'un document propre. Une stratégie standard peut également être définie au niveau du SI.

2.6 Éco-conception

Lister ici les mesures logicielles permettant de répondre aux exigences d'écoconception listée dans le volet infrastructure. Les réponses à ses problématiques sont souvent les mêmes que celles aux exigences de performance (temps de réponse en particulier). Dans ce cas, y faire simplement référence. Néanmoins, les analyses et solutions d'écoconception peuvent être spécifiques à ce thème. Quelques pistes d'amélioration énergétique du projet :

- Utiliser des profilers ou des outils de développement intégrés dans les navigateurs (comme Google Dev Tools) pour analyser la consommation de ressources (nombre, durée et taille des requêtes).

- Pour les apps, utiliser des outils de supervision de la consommation de batterie comme Battery Historian.
- Utiliser la suite d'analyse spécialisée Greenspector.
- Mesurer la consommation électrique des systèmes avec les sondes PowerAPI2 (développé par l'INRIA et l'université Lille 1).
- Mesurer la taille des images et les réduire (sans perte) avec des outils comme pngcrush, OptiPNG, pngrewrite ou ImageMagick.
- Optimiser la consommation mémoire et CPU des applications, tuner le GC pour une application Java.
- Faire du lazy loading pour le chargement des ressources occasionnelles.
- Limiter les résultats retournés de la base de donnée (pagination).
- Grouper les traitements de masse dans des batches qui seront plus efficaces (lots).

Exemple 1 : le processus gulp de construction de l'application appliquera une réduction de taille des images via le plugin imagemin-pngcrush.

Exemple 2 : des tests de robustesse courant sur plusieurs jours seront effectués sur l'application mobile après chaque optimisation pour évaluer la consommation énergétique de l'application.

Exemple 3 : Les campagnes de performance intégreront une analyse fine de la consommation de bande passante et en cycles CPU même si les exigences en temps de réponses sont couvertes, ceci pour identifier des optimisations permettant de répondre aux exigences d'éco-conception si elles ne sont pas atteintes.

2.6.1 Gestion de la robustesse

2.6.1.1 Gestion des transactions

Lister ici les décisions prises concernant la gestion des transactions. Ceci est surtout utile pour un système distribué. Quelques exemples de problématiques :

- Autorise-t-on les mises jours sur de multiples composants lors d'une même requête ?
- Si oui, assurons nous le caractère ACID du tout (via le mode XA par exemple) ?
- Quel moteur transactionnel utilisons nous ?
- Quel niveau d'isolation transactionnelle (read committed, uncommitted, repeatable read, serializable) ?
- Si aucun moniteur transactionnel n'est utilisé (appel de plusieurs services REST en mise à jour par exemple), prévoit-t-on des transactions compensatoires en cas d'échec de l'une des mises à jours ?

Exemple : nos ressources n'étant pas transactionnelles (services REST), et voulant éviter de faire des transactions compensatoires, il est interdit d'appeler deux services en mise à jour de façon synchrone. Au besoin, nous utiliserons une file pour effectuer des mises à jour au fil de l'eau.

2.6.1.2 Gestion des sessions

Comment gère-t-on les sessions HTTP permettant de fournir un contexte d'exécution à un utilisateur (exemple: son panier d'achat) ?

Notez que ceci est une surtout un problème pour les applications Web classiques dont la présentation est générée sur le serveur, pas pour les applications SPA (Single Page Application) qui gèrent toute la

présentation et leur état en local dans le navigateur.

Les choix faits ici affecteront les choix d'infrastructure. Par exemple, si une session est requise et que l'infrastructure est en cluster, il faudra soit mettre en place de l'affinité de session sur les serveurs pour forcer chaque utilisateur à toujours arriver sur le même serveur disposant de ses données, soit de mettre en place un cache distribué permettant aux serveurs de partager les sessions de tous les utilisateurs (plus complexe).

Exemples de points à traiter :

- Quelles données doivent être conservées en session ? (attention à la volumétrie, surtout si cache distribué)
- Le code doit-il être thread-safe (si le même utilisateur ouvre un autre onglet dans son navigateur par exemple) ?

Exemple : notre application JSF stockera en session HTTP uniquement son panier d'achat, pas les références produits

2.6.1.3 Gestion des erreurs

Comment gère-t-on erreurs ? Exemples de points à traiter :

- Différencions-nous erreurs fonctionnelles (erreurs fonctionnelles prévues) et techniques ? Prévoir un diagramme de classe.
- Comment logue t-on les erreurs ? quel niveau de log ?
- Où sont attrapées les exceptions ? au plus tôt ou en début d'appel de façon centralisée ?
- Utilise-t-on les exceptions standards du langage (`IOException...`) ou notre propre jeu d'exceptions ?
- La liste des erreurs est-elle consolidée ? documentée ?
- Affecte-t-on des codes erreur ?
- Affiche-on les stack-traces complètes ? si oui, coté serveur et coté client ?
- Gère-t-on les rejeux ? si oui, espace-t-on les rejeux ? de façon aléatoire (jitter) ? exponentielle (exponential backoff) ?
- Comment gère-t-on les timeouts ?
- Comment gérons-nous les rejets fonctionnels ? (c.-à-d. que faire des demandes partielles ou erronées?)

Exemple : les erreurs techniques (imprévues) comme le timeout à un appel de service REST sont catchées au plus haut niveau de l'application (via un `ErrorHandler`). Toutes ses informations sont loguées avec la stack-trace complète mais l'appelant ne doit récupérer que le code erreur générique XYZ sans la stack-trace (pour raison de sécurité).

2.6.2 Gestion de la configuration

Comment configure-t-on l'application ? Exemples de points à traiter :

- Quels sont les variables incluses dans le package final de façon statique ?
- Quels sont les paramètres modifiables au runtime ?
- Mon application est-elle paramétrable via feature flags pour des raisons de canary testing par exemple ? si oui, comment je le gère dans le code ?

- Sous quelle forme les paramètres sont-ils injectés dans l'application (variable d'environnement ? fichier `.properties`, base de donnée, ...) ?
- L'application accepte-elle une modification du paramétrage à chaud ?
- Décrire le système de configuration

Exemple (application déployées dans Kubernetes) :

La configuration sera injectée au lancement (non modifiable à chaud) via des variables d'environnements fournies dans le décripteur de déploiement Kubernetes.

2.6.3 Politique de gestion des branches

Quels sont des workflows de branche à prévoir ? git-flow ? TBD (Trunked-based Development) ? autre ?

Exemple :

- La politique générale adoptée est la TBD (Trunk-Based Development)
- La branche principale est `develop`. Il s'agit d'une branche protégée vers laquelle il n'est pas possible pousser de commits.
- Tout commit devra faire l'objet d'une Merge Request avant intégration dans `develop`. Les critères de qualité (évalués de façon automatique lors de l'intégration continue) devront être atteints pour que le commit soit intégré.
- Chaque fonctionnalité, refactoring significatif ou bugfix sera donc réalisé sur une branche topic dédiée.
- Une branche de maintenance sera tirée sur chaque tag de version `x.y`. Seuls les bugfixs seront mergés dans les branches de maintenance depuis `develop` via des `cherry-pick`.

2.6.4 Versioning

Que versionne-t-on et quel système de version utilise-t-on ?

Exemple:

- D'une façon générale, toute ressource non dérivée (source, outil, script de ci-cd, template, DDL de base de données, ...) doit être versionnée.
- Les modules seront versionnés suivant la numérotation `x.y.z (<majeur>.<évolution>.<fix>)`
- Les librairies seront versionnées suivant la même numérotation que les modules mais la valeur `x` sera incrémentée lors de toute montée de version cassant la compatibilité ascendante (principe du Semantic Versioning).
- La version logique globale du projet sera : `<lot>.<no sprint>.<déploiement>`

2.6.5 Gestion de la concurrence

Comment gère-t-on les accès concurrents ? Exemples de points à traiter :

- Quel scope pour les objets (si utilisation d'un moteur IoC) ?
- Les objets doivent-il être thread-safe ?
- Quels méthodes doivent-elles être synchronisées ?
- Risques de race condition ? de starvation ? de dead locks ?

Exemple (Spring MVC) : Tous les controllers seront en scope singleton et ne doivent donc en aucun cas stocker d'état dans leurs attributs pour éviter des race conditions.

2.6.6 Encodage

Quelles sont les règles concernant l'encodage des chaînes de caractère ? Ceci est un problème récurrent dans les SI (qui n'a jamais observé d'accents corrompus sous forme de carrés ?). Ce problème est pourtant relativement simple à résoudre et n'exige que de la rigueur. Voir les exemples ci-dessous pour des exemples de dispositifs effectifs.

Exemple 1 : Le seul encodage autorisé dans tous les modules et composants techniques est l'UTF-8. L'utilisation de l'ISO-8859-1, CP-1252 ou de tout autre encodage est formellement proscrit. Ceci comprend le paramétrage des serveurs d'application (Node, Tomcat...), des sources, des fichiers de configuration, des bases de données et des fichiers.

Dans certains cas, nous n'avons pas la main sur la lecture des .properties (depuis un framework par exemple), il n'est alors pas possible de forcer un encodage en UTF-8.

Exemple 2 : Si un système externe impose d'envoyer ou de recevoir des chaînes de caractères dans un encodage autre que le UTF-8 (exemple : un service REST qui renvoi des données en ISO-8859-1) et qu'il n'est pas possible de modifier le contrat, il est impératif de traduire au sein d'une couche anti-corruption les chaînes de caractères et ceci au plus tôt, dès l'appel. De plus, il ne faut jamais persister dans nos systèmes une donnée dans un encodage non UTF-8.

2.6.7 Fuseaux horaires

Comment gère-t-on le stockage des dates ? Ceci, comme la gestion de l'encodage est un problème récurrent (décalage d'un jour, bugs lors des changements d'heure d'été/hiver, etc.) et pourtant simple à résoudre : suivre la norme ISO 8601 ("Time zones in ISO 8601 are represented as local time (with the location unspecified), as UTC, or as an offset from UTC." [Wikipedia]).

Exemple 1 : Les heures ne seront jamais stockées sans fuseau horaire. En base, on utilisera des timestamps avec timezone (`timestamp_tz`) et en Java ou JS, des objets intégrant le fuseau horaire de façon explicite (ex: `Instant` et pas `LocalDateTime` en java) ou des epochs. La précision sera au moins de la milliseconde.

Exemple 2 : Les dates et date-heures seront stockées en base de données comme epoch millis au format entier long. Dans le cas des dates, on stockera l'epoch millis à 12:00 UTC (et pas 00:00, trop proche du jour précédent, risque de bug).

2.6.8 Gestion des logs

Les aspects d'infrastructure de logs sont détaillés dans le volet infrastructure.

Donner ici les règles générales concernant les traces applicatives (logs), les niveaux et quantité de logs. Penser à l'exploitation des logs, surtout coté serveur. Se demander s'il sera possible d'en tirer profit en cas d'erreur en production au milieu de Mio voire Gio d'autres logs et de n threads loguant en parallèle.

2.6.8.1 Règles générales

Exemple 1 :

- Ne pas laisser de logs de développement dans le code (exemple : `console.out("entrée dans méthode x")` ou `e.printStackTrace()`)
- Penser à utiliser des chaînes de caractère discriminantes (exemple : code erreur) pour faciliter le filtrage dans l'outil de recherche de logs.
- Toujours fournir des identifiants d'entités permettant de retrouver l'objet concerné
- Utiliser des identifiant de corrélation entre tiers (exemple : id de traitement générée coté client en JS, passée au serveur)
- Eviter les calculs coûteux (exemple: beaucoup de concaténations) et utiliser des blocs conditionnels (exemple en Java :

```
if (isDebugEnabled()){
    logger.debug(a+b+c)
}
```

2.6.8.2 Niveaux et quantité de logs

Expliquer quand et quoi loguer de sorte à produire des logs exploitables en production.

Exemple :

TABLE 2.6: Niveaux logs

| Niveau de gravité | Contexte d'utilisation | Volume indicatif | Environnement |
|-------------------|--|--|--|
| DEBUG | En environnement de développement, il permet d'afficher les valeurs de variables, E/S de méthodes etc.. | Max quelques Mio / minute | DEV, Recette. Interdit en PROD sauf demande expresse du projet |
| INFO | Début/fin d'un batch ou d'un appel, chargement d'une nouvelle propriété. Peut être utilisé sous forme condensée pour les appels de service (logging d'un appel et de son contexte). C'est le niveau de proximité utilisé pour la métrologie. | Max 10 logs / sec, quelques Kio / minute | Tous |
| WARN | Tous les messages d'avertissement sur les informations fonctionnelles inattendues | Pas de limites mais ne pas en abuser et y positionner un maximum de détail de contexte | Tous |
| ERROR | Toutes les erreurs qui n'empêchent pas à l'application de fonctionner. | Pas de limites. Positionner un maximum de détail de contexte | Tous |

| Niveau de gravité | Contexte d'utilisation | Volume indicatif | Environnement |
|-------------------|--|------------------|---------------|
| FATAL | Toutes les erreurs bloquantes pour l'application (problème d'accès BDD, HTTP 404 ou 500). Positionner un maximum de détail de contexte. Penser à bien logger ces erreurs sur un <code>append</code> console au cas où l'écriture sur FS serait impossible (disque plein). Penser que lors d'une erreur fatale, l'écriture même du log est sujette à caution (par exemple en cas de dépassement mémoire). | Pas de limites. | Tous |

2.6.9 Outils d'administration

L'application doit-elle fournir des services d'administration ? Il est fortement conseillé (c'est le facteur 12 des Twelve factors d'Heroku) d'intégrer le code d'administration directement avec le code métier.

Exemples de points à traiter :

- Dois-je fournir un moyen de purger des données, logs, caches, ... ? (on appelle quelque fois ce type de service un 'traitement interne')
- Dois-je fournir des indicateurs applicatifs de supervision ? (nombre de dossiers consultés, ...) ?
- Dois-je fournir des outils de migration ?

Exemple : Le service `/interne/maj_2` effectuera une montée de version du modèle de donnée vers la V2

2.6.10 Tri et Pagination

Il est nécessaire de conserver une bonne fluidité de récupération des données en lot. La pagination permet de limiter le bavardage entre les clients (IHM et batchs) et les API. Décrire ici les dispositifs de pagination mis en oeuvre coté client et coté serveur.

Exemple 1 (Coté serveur)

- Les requêtes en sortie de l'api sont systématiquement triées selon un ordre ascendant (le défaut) ou descendant. De plus, il sera possible de choisir le champ sur lequel se fait le tri via un autre query param.
- Afin de limiter le nombre de requêtes à destination de l'api, celle-ci retourne un nombre limité d'éléments (ce nombre sera paramétrable suivant la taille des éléments individuels). Il s'agit du query param `range` contenant le numéro de la page à récupérer + le nombre d'éléments de la page. Chaque API proposera une valeur par défaut (de l'ordre d'une centaine).

Exemple 2 (Coté client)

- Le tri doit s'appliquer sur l'ensemble des éléments en base, pas seulement sur les éléments de la dernière requête retournée par le serveur.
- Les éléments retournés seront affichés dans les tableaux par blocs (taille paramétrable d'une

taille indicative de l'ordre de 20 éléments).

2.6.11 Provisioning et mises à jour des DDL

Décrire comment les DDL (structures de tables en base de données) et les données initiales (comme des nomenclatures) seront gérées puis mis à jour.

Exemple : Nous utiliserons Liquibase embarqué dans les war pour créer et mettre à jour les DDL de la base. Il n'y aura donc pas de scripts SQL à lancer, les requêtes nécessaires seront effectuées directement par l'application lors de son démarrage.

Chapitre 3

Volet dimensionnement

Dernière modification : 2022-05-03

3.1 Introduction

Ceci est le point de vue dimensionnement du projet. Il permet de déterminer la taille de l'infrastructure nécessaire au projet.

Les autres volets du dossier sont accessibles d'ici.

Le glossaire du projet est disponible ici. Nous ne redéfinirons pas ici les termes fonctionnels ou techniques utilisés.

3.1.1 Documentation de Référence

TABLE 3.1: Références documentaires dimensionnement

| N° | Version | Titre/URL du document | Détail |
|----|---------|---|--------|
| 1 | 1.2 | <i>Rapport_benchmark_xyz.pdf</i> | |
| 2 | 2019 | <i>Rapport INSEE sur les entreprises françaises</i> | |

3.2 Non statué

3.2.1 Points soumis à étude complémentaire

TABLE 3.2: Points soumis à étude complémentaire

| ID | Détail | Statut | Porteur du sujet | Échéance |
|----|-----------------------------|-----------------|------------------|-------------------|
| 1 | <i>Capacité disques SSD</i> | <i>EN_COURS</i> | <i>XYZ</i> | <i>01/01/2022</i> |

3.2.2 Hypothèses

Donner ici les hypothèses structurantes prises pour le dimensionnement

Exemple:

TABLE 3.3: Hypothèses

| ID | Détail |
|----|--|
| 1 | <i>Nous estimons que 10M de personnes téléchargeront l'application StopCovid</i> |

3.3 Contraintes

Les contraintes sont les limites applicables aux exigences sur le projet.

Il est intéressant de les expliciter pour obtenir des exigences réalistes. Par exemple, il ne serait pas réaliste d'exiger des temps de réponse d'un web service incompatibles avec le débit du réseau sous-jacent.

3.3.1 Contraintes stockage

Lister ici les éventuelles contraintes liées aux disques

Exemple : L'espace disque maximal allouable à une VM est de 2 To.

3.3.2 Contraintes CPU

Lister ici les éventuelles contraintes liées à la puissance de calcul

Exemple 1 : Une VM sera dotée au maximum de 10 VCPU

Exemple 2 : L'ensemble des pods de l'application ne devra pas demander plus de 1 CPU par node.

3.3.3 Contraintes mémoire

Lister ici les éventuelles contraintes liées à la mémoire

Exemple : un pod ou un job Kubernetes ne devra pas utiliser plus de 6 Go de RAM

3.3.4 Contraintes réseau

Lister ici les éventuelles contraintes de volume réseau utilisé

Exemple 1 : La latence minimale des requêtes sur le WAN entre Londres et Tokyo est de 250 ms

Exemple 2 : Le réseau Ethernet du Datacenter dispose d'une bande passante de 40 Gbps.

3.4 Exigences

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 nouveau

projet. Prévoir alors une marge importante. Les informations données ici pourront servir d'entrants au SLO (Objectif de Service fixé par les exploitants).

Les sections suivantes portant sur les calculs de volumétrie statiques et dynamiques donnent des exemples de calculs et d'éléments à prendre en compte. Sur le plan de la forme, il peut être préférable de remplacer le texte par des feuilles de calculs dans un tableur. Pour un projet complexe ou aux hypothèses mouvantes, c'est indispensable.

3.4.1 Volumétrie statique

Il s'agit des métriques permettant de déterminer le volume de stockage **cumulé** du projet. Penser à bien préciser les hypothèses prises pour les métriques estimées. Il sera ainsi possible de les revoir si de nouveaux éléments métier apparaissent.

3.4.1.1 Métriques

Il s'agit des données métier mesurées ou estimées qui serviront d'entrants au calcul des besoins techniques de stockage.

| Métrique | Description | Mesurée ou Estimée ? | Valeur | Augmentation annuelle prévision- nelle (%) | Source | Détail/hypothèses |
|-----------|---|-------------------------|-------------|---|--------------------|--|
| <i>S1</i> | <i>Nombre d'entre- prises éligibles</i> | <i>Estimé</i> | <i>4M</i> | <i>+1%</i> | <i>INSEE [2]</i> | <i>On considère que MIEL ne concerne pas les auto- entrepreneurs</i> |
| <i>S2</i> | <i>Taille moyenne d'un PDF</i> | <i>Mesurée</i> | <i>40Ko</i> | <i>0%</i> | <i>Exploitants</i> | |

3.4.1.2 Estimation besoins de stockage

Lister ici les besoins en stockage de chaque composant une fois l'application arrivée à pleine charge (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.
- L'espace nécessaire dans un éventuel stockage objet (S3, Swift, Ceph...)
- ...

Ne pas prendre en compte :

- Le volume lié à la sauvegarde : elle est gérée par les exploitants.

- Le volume des binaires (OS, intergiciels...) qui est à considérer par les exploitants 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 exploitants de commander ou réserver suffisamment de disque.

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 (très piètre) estimation préliminaire est de doubler l'espace disque (à affiner ensuite).

N'estimer que les données dont la taille est non négligeable (plusieurs Gio minimum).

1. Exemple de 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 | 2Kio | 100K | 200 Mio | 5 % |
| Table Commande | Les commandes clients | 10Ko | 3M | 26.6 Gio | 10 % |
| Logs | Les logs applicatifs (niveau INFO) | 200 o | 300M | 56 Gio | 0 % (archivage) |

3.4.2 Volumétrie dynamique

Il s'agit des métriques par durée (année, mois, heure...) et permettant de déterminer la charge appliquée sur l'architecture, ce qui aidera à dimensionner les systèmes en terme de CPU, bande passante et performances des disques.

3.4.2.1 Métriques

Ce sont les données métier mesurées ou estimées qui serviront d'entrants au calcul de la charge.

| Métrique | Description | Mesurée ou Estimée ? | Valeur | Augmentation annuelle prévision- nelle (%) | Saisonnalité | Source | Détail/hypothèses |
|-----------|---|----------------------------|-----------|---|--------------|--|--|
| <i>D1</i> | <i>Proportion d'utilisa- teurs se convec- tant au service / J</i> | <i>Estimée</i> | <i>1%</i> | <i>+5%</i> | — | Constant sur l'an- née | <i>Les utili- sateurs sont des profes- sionnels utilisant l'applica- tion depuis la France métropoli- taine aux heures de bureau standards</i> |
| | | | | | — | Constant sur la se- maine | |
| | | | | | — | 3 pics à 20% de la jour- née à 8:00- 9:00, 11:00- 12:00 et 14:00- 15:00 | |

3.4.2.2 Estimation de la charge

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 du même ordre 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 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 pics 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 c'est possible plutôt que sur 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 voire 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 tronçonnera 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 respecter le budget tout en assurant un bon niveau de service.

TABLE 3.7: Exemple : estimation volumétrie dynamique de l'opération `REST GET Detail` de l'application MIEL

| | |
|---|---|
| Taux maximal d'utilisateurs connectés en même temps en pic annuel | $S1 \times F1 \times 0.2 = 8K / H$ |
| Durée moyenne d'une session utilisateur | 15 mins |
| Nombre d'appel moyen du service par session | 10 |
| Charge (Transaction / seconde) | $8K / 4 \times 10 / 3600 = 5.5 \text{ Tps}$ |

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 tous 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 de l'application 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. Inutile de 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.

Exemple : la base de donnée Oracle BD01 est utilisée en lecture par les appels `REST GET DetailArticle` fait depuis l'application end-user et 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.

TABLE 3.8: Exemple estimations nombre de requêtes SQL en pic vers l'instance BD01 de 01:00 à 02:00 en décembre

| | |
|--|--|
| Taux maximal d'utilisateurs connectés en même temps | 0.5% |
| Nombre maximal d'utilisateurs connectés concurrents | 5K |
| Durée moyenne d'une session utilisateur | 15 mins |
| Nombre d'appel moyen du service GET DetailArticle par session | 10 |
| Charge usagers GET DetailArticle (Transaction / seconde) | $(10/15) \times 5K / 60 = 55$ Tps |
| Nombre de requête en lecture et écriture par appel de service | 2 et 0 |
| Nombre d'appel journalier du service POST DetailArticle depuis le batch B03 | 4K |
| Nombre de requêtes INSERT et SELECT par appel de service | 3 et 2 |
| Nombre journalier d'articles modifiés par le batch B03 | 10K |
| Nombre de requêtes SELECT et UPDATE | 1 et 3 |
| Nombre de SELECT / sec | $55 \times 2 + 2 \times 4K/3600 + 1 \times 10K/3600 = 115$ Tps |
| Nombre de INSERT / sec | $0 + 3 \times 4K/3600 = 3.4$ Tps |
| Nombre de UPDATE / sec | $0 + 3 \times 10K/3600 = 8.3$ Tps |

3.4.3 Exigences de temps de réponse

3.4.3.1 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 (90ème 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) car ce type de critère n'a plus grand sens aujourd'hui, particulièrement pour une application SPA).

TABLE 3.9: Exemple de types de sollicitation :

| Type de sollicitation | Bon niveau | Niveau moyen | Niveau insuffisant |
|------------------------------|------------|--------------|--------------------|
| <i>Chargement d'une page</i> | $< 0,5$ s | < 1 s | > 2 s |
| <i>Opération métier</i> | < 2 s | < 4 s | > 6 s |

| Type de sollicitation | Bon niveau | Niveau moyen | Niveau insuffisant |
|------------------------------------|----------------|----------------|--------------------|
| <i>Édition, Export, Génération</i> | $< 3\text{ s}$ | $< 6\text{ s}$ | $> 15\text{ s}$ |

Exemple d'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 valeurs, l'application devra être optimisée et repasser en recette puis être soumise à nouveau aux tests de charge.

3.4.3.2 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 aux usagers, ces premiers doivent impérativement se terminer avant la fin de la plage batch définie plus haut.

Exemple 2 : le batch mensuel B1 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 stricte sur 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.

3.5 Dimensionnement cible

Nous donnons un dimensionnement final devant supporter la volumétrie statique et dynamique et respecter les exigences.

3.5.1 Estimation des besoins en ressources par composant technique

Donner ici RAM, disque et CPU par instance de composant technique déployé (à affiner après campagne de performance ou MEP).

Exemple :

TABLE 3.10: Estimation des besoins en ressources par composant technique

| Unité déployable | Besoin en (V)CPU par instance | Besoin mémoire par instance (Mio) | Périodes d'activité | Commentaires |
|-----------------------|-------------------------------|-----------------------------------|--|---|
| <i>tomcat-batchs1</i> | <négligeable> | 1024 | Toutes les heures, 24/7/365 | Le serveur d'application reste démarré même en dehors de l'exécution des jobs |
| <i>spa</i> | <négligeable> | 50 | 24/6, activité principale 8-17h Europe/Paris lun-ven | Appli Web SPA, s'exécute dans le navigateur |
| <i>bdd-postgresql</i> | 2 | 2024 | 24/7, activité principale 8-17h Europe/Paris lun-ven | Instance Postgresql |

3.5.2 Dimensionnement des machines

Voir le modèle de déploiement.

Cette section fournit le dimensionnement final des machines nécessaires

- Pour les VM, attention à vérifier qu'un VCPU = 1 cœur physique (et non un thread si hyperthreading activé)
- Le disque interne concerne le disque nécessaire à l'OS et aux binaires. Pour une machine physique, il s'agit de stockage local (disques locaux SDD, NMVe ou HDD). Pour une VM, il peut s'agir d'un disque local sur la machine physique exécutant la VM ou d'un SAN.
- Le disque distant concerne du stockage sur une baie de disque (SAN)
- Le stockage externe hors SAN concerne du stockage fichier sur un filesystem distribué (NFS, CIFS, WebDav, ...) ou un stockage objet (Swift, S3, ...)

TABLE 3.11: Dimensionnement des machines

| Zone | Type de machine | Nb de machines | Nb (V)CPU | Mémoire (Gio) | Disque interne (Gio) | Disque distant SAN (Gio) |
|---------|----------------------------------|----------------|-----------|---------------|----------------------|--------------------------|
| Zone 01 | VM serveur applicatif | 3 | 2 | 4 | 100 | 0 |
| Zone 02 | Machine physique Base de données | 1 | 2 | 6 | 50 | 1024 |

TABLE 3.12: Dimensionnement du stockage externe hors SAN

| Nature | Taille (Gio) | Type(s) de machine utilisant ce partage |
|---|--------------|---|
| <i>NFS</i> <i>(montage</i> <i>NAS)</i> | <i>248</i> | <i>Machine physique Base de données</i> |
| <i>OpenStack</i> <i>Object</i> <i>Storage</i> <i>("Swift")</i> | <i>20</i> | <i>VM serveur applicatif</i> |

Chapitre 4

Volet infrastructure

Dernière modification : 2022-05-03

4.1 Introduction

Ceci est le point de vue infrastructure de l'application. Il décrit le déploiement des modules applicatifs dans leur environnement d'exécution cible et l'ensemble des dispositifs assurant leur bon fonctionnement.

Les autres volets du dossier sont accessibles d'ici.

Le glossaire du projet est disponible ici. Nous ne redéfinirons pas ici les termes fonctionnels ou techniques utilisés.

Ce point de vue est aussi souvent appelé « point de vue technique » et concerne l'infrastructure : serveurs, réseaux, systèmes d'exploitation, bases de données, intergiciels (middleware), ...

Bref, elle porte sur tout ce qui est externe à l'application et nécessaire à son exécution

4.1.1 Documentation de Référence

Mentionner ici les documents d'architecture de référence (mutualisés). Ce document ne doit en aucun cas reprendre leur contenu sous peine de devenir rapidement obsolète et non maintenable.

TABLE 4.1: Références documentaires

| N° | Version | Titre/URL du document | Détail |
|----|---------|--------------------------------|--|
| 1 | | <i>Regles__sauvegardes.pdf</i> | <i>Règles concernant les sauvegardes</i> |

4.2 Non statué

4.2.1 Points soumis à étude complémentaire

TABLE 4.2: Points soumis à étude complémentaire

| ID | Détail | Statut | Porteur du sujet | Échéance |
|------------|---|-----------------|-------------------------------|-------------------|
| <i>EI1</i> | <i>Le choix technique de la solution d'API Management reste soumise à étude complémentaires</i> | <i>EN_COURS</i> | <i>Équipe Archi Technique</i> | <i>AVANT 2040</i> |

4.2.2 Hypothèses

TABLE 4.3: Hypothèses

| ID | Détail |
|------------|---|
| <i>HI1</i> | <i>Nous prenons l'hypothèse que d'ici à la MEP du projet, PostgreSQL 11 sera validé en interne.</i> |

4.3 Contraintes

Les contraintes sont les limites applicables aux exigences sur le projet.

Il est intéressant de les expliciter pour obtenir des exigences réalistes. Par exemple, il ne serait pas valide d'exiger une disponibilité incompatible avec le niveau de sécurité Tier du datacenter qui l'hébergera.

4.3.1 Contraintes sur la disponibilité

Les éléments ici fournis pourront servir de base au SLO (Service Level Objective). Idéalement, ce dossier devrait simplement pointer sur un tel SLO sans plus de précision.

Ce chapitre a une vocation pédagogique car il rappelle la disponibilité plafond envisageable : la disponibilité finale de l'application ne pourra être qu'inférieure.

4.3.1.1 MTDD

Donner les éléments permettant d'estimer le temps moyen de détection d'incident.

Exemple 1 : l'hypervision se fait 24/7/365

Exemple 2 : le service support production est disponible durant les heures de bureau mais une astreinte est mise en place avec alerting par e-mail et SMS en 24/7 du lundi au vendredi.

4.3.1.2 MTTR

Donner les éléments permettant d'estimer le temps moyen de réparation (Mean Time To Repair en anglais). A noter qu'il est important de distinguer le MTDD du MTTR. En effet, ce n'est pas parce qu'une panne est détectée que les compétences ou ressources nécessaires à sa correction sont disponibles.

Préciser les plages de présence des exploitants en journée et les possibilités d'astreintes.

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

Exemple 1 : Cinq serveurs physiques de spare sont disponibles à tout moment.

Exemple 2 : Le contrat de support Hitashi prévoit une intervention sur les baies SAN en moins de 24h.

Exemple 3 : Au moins un expert de chaque domaine principal (système et virtualisation, stockage, réseau) est présent durant les heures de bureau.

Exemple 4 : Comme toute application hébergée au datacenter X, l'application disposera de la présence d'exploitants de 7h à 20h jours ouvrés. Aucune astreinte n'est possible.

Exemple 5 : le remplacement de support matériel IBM sur les lames BladeCenter est assuré en 4h de 8h à 17h, jours ouvrés uniquement.

4.3.1.3 Outils et normes de supervision

Donner ici les outils et normes de supervisions imposés au niveau du SI et les éventuelles contraintes liées.

Exemple 1 : L'application sera supervisée avec Zabbix

Exemple 2 : Les batchs doivent pouvoir se lancer sur un endpoint REST

Exemple 3 : un batch en erreur ne doit pas pouvoir se relancer sans un acquittement humain

4.3.1.4 Interruptions programmées

Donner ici la liste et la durée des interruptions programmées standards dans le SI.

Exemple 1 : On estime l'interruption de chaque serveur à 5 mins par mois. Le taux de disponibilité effectif des serveurs en prenant en compte les interruptions programmées système est donc de 99.99 %.

Exemple 2 : suite aux mises à jour de sécurité de certains packages RPM (kernel, libc...), les serveurs RHEL sont redémarrés automatiquement la nuit du mercredi suivant la mise à jour. Ceci entraînera une indisponibilité de 5 mins en moyenne 4 fois par an.

4.3.1.5 Niveau de service du datacenter

Donner ici le niveau de sécurité du datacenter selon l'échelle Uptime Institute (Tier de 1 à 4).

TABLE 4.4: Niveaux Tier des datacenters (source : Wikipedia)

| Niveau Tier | Caractéristiques | Taux de disponibilité | Indisponibilité statistique annuelle | Maintenance à chaud possible ? |
|------------------------|------------------|-----------------------|--------------------------------------|--------------------------------|
| Tolérance aux pannes ? | Tier 1 | Non redondant | 99,671 % | 28,8 h |

| Niveau Tier | Caractéristiques | Taux de disponibilité | Indisponibilité statistique annuelle | Maintenance à chaud possible ? |
|----------------------|------------------|-----------------------|--------------------------------------|--------------------------------|
| Non | Non | Tier 2 | Redondance partielle | 99,749 % |
| 22 h | Non | Non | Tier 3 | Maintenabilité |
| 99,982 % | 1,6 h | Oui | Non | Tier 4 |
| Tolérance aux pannes | 99,995 % | 0,4 h | Oui | Oui |

Exemple : le datacenter de Paris est de niveau Tier III et celui de Toulouse Tier II.

4.3.1.6 Synthèse de la disponibilité plancher

En prenant en compte les éléments précédents, estimer la disponibilité planché (maximale) d'une application (hors catastrophe). Toute exigence devra être inférieure à celle-ci. Dans le cas d'un cloud, se baser sur le SLA du fournisseur. Dans le cas d'une application hébergée en interne, prendre en compte la disponibilité du datacenter et des indisponibilité programmées.

Exemple : $\langle \text{disponibilité datacenter} \rangle * \langle \text{plage de fonctionnement effective} \rangle * \langle \text{disponibilité système} \rangle * \langle \text{disponibilité hardware} \rangle = 99.8 \times 99.99 \times 99.6 \times 99.99 \approx \mathbf{99.4\%}$.

4.3.1.7 Gestion des catastrophes

Les catastrophes peuvent être classées en trois catégories:

- Naturelle (tremblements de terre, inondations, ouragans, canicules...).
- Incident sur l'infrastructure du datacenter (accidentel comme les accidents industriels, incendies, pannes électriques majeures, pannes majeures du réseau / stockage / serveurs, les erreurs critiques d'administrateurs ou intentionnelles: militaire, terroriste, sabotage ...).
- Cyber (DDOS, virus, Ransomware ...)

PRA (Plan de Reprise d'Activité) comme PCA (Plan de Continuité d'Activité) répondent à un risque de catastrophe sur le SI (catastrophe naturelle, accident industriel, incendie...). Un PRA permet de reprendre l'activité suite à une catastrophe après une certaine durée de restauration. Il exige au minimum un doublement du datacenter.

Un PCA permet de poursuivre les activités critiques de l'organisation (en général dans un mode dégradé) sans interruption notable. Ce principe est réservé aux organisations assez matures car il exige des dispositifs techniques coûteux et complexes (filesystems distribués et concurrents par exemple).

Un architecte n'utilise pas les mêmes technologies suivant qu'on vise un PRA ou un PCA. Par exemple, si on vise un PCA, il faut prévoir des clusters actifs-actifs multi-zonaux (situés dans des datacenters distants géographiquement) alors que pour un PRA, l'important est la qualité et la vitesse de sauvegarde/restauration des données dans le datacenter de secours.

Note: Dans la plupart des grands comptes, PRA comme PCA impliquent une réplication par lien optique des baies SAN pour limiter le RPO au minimum et s'assurer que l'ensemble des données du

datacenter soient bien répliquées. Les systèmes de sauvegarde/restauration classiques sont rarement suffisants à couvrir ce besoin. La différence est que dans le cas d'un PRA, il faut prévoir une bascule et une préparation conséquente du datacenter de secours alors que dans le cas d'un PCA, des deux (ou plus) datacenters fonctionnent en parallèle en mode actif/actif de façon nominale.

Note: La gestion des catastrophes est un sujet complexe. C'est l'un des points forts des Clouds publics (OVH, GCP, Azure, AWS...) que de gérer une partie de cette complexité pour vous. Des solutions Cloud spécifiques existent (Disaster Recovery as a Service (DRaaS)).

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 PRA, donner le **Recovery Time Objective** (durée maximale admissible de rétablissement en heures) et le **Recovery Point Objective** (durée maximale admissible de données perdues en heures depuis la dernière sauvegarde).
- Pour un PCA les dispositifs de réplication de données (synchrone ? fil de l'eau ? Combien de transactions peuvent-être perdues ?).
- Présenter la politique de failback (réversibilité) : doit-on rebasculer vers le premier datacenter ? Comment ?
- Comment sont organisés les tests de bascule à blanc ? Avec quelle fréquence ?

Exemple de PRA : Pour rappel (voir [doc xyz]), les VM sont répliquées dans le datacenter de secours via la technologie vSphere Metro Storage Cluster utilisant SRDF en mode asynchrone pour la réplication inter-baies. En cas de catastrophe, la VM répliquée sur le site de secours est à jour et prête à démarrer. Le RPO est de ~0 secs et le RTO de 30 mins.

Autre exemple de PRA (PME avec son propre datacenter à Paris) : Stockage de deux serveurs de spare dans les locaux de Lille. Sauvegarde à chaud toutes les quatre heures des données principales de l'entreprise et envoi (avec chiffrement client) sur BackBlaze.com. Le RPO est de 4h, le RTO de 2H.

Exemple de PCA avec élasticité: Les applications s'exécutent sous forme de POD Kubernetes sur au moins trois clusters situées dans des zones géographiquement distantes. Les données MongoDB sont shardées et synchronisées entre zones via un système de ReplicatSet. Le système est auto-régulé par Kubernetes et tout plantage d'un DC sera compensé en quelques secondes par la création de nouveaux POD dans les deux clusters restants. Ainsi, non seulement les utilisateurs n'auront pas de perte de disponibilité mais ils ne verront pas non plus leurs performances dégradées. Le RTO est de 0, le RPO est ~0 (zéro strict est impossible, voir le théorème de CAP).

4.3.2 Hébergement

- Où sera hébergée cette application ? datacenter "on premises" ? Cloud interne ? Cloud IaaS ? PaaS ? autre ?
- Qui administrera cette application ? en interne ? Sous-traité ? Pas d'administration (PaaS) ... ?

Exemple 1: Cette application sera hébergée en interne dans le datacenter de Nantes (seul à assurer la disponibilité de service exigée) et il sera administré par l'équipe X de Lyon.

Exemple 2 : Étant donné le niveau de sécurité très élevé de l'application, la solution devra être exploitée uniquement en interne par des agents assermentés. Pour la même raison, les solutions de cloud sont

exclues.

Exemple 3 : Étant donné le nombre d'appels très important de cette application vers le référentiel PERSONNE, elle sera colocalisée avec le composant PERSONNE dans le VLAN XYZ.

4.3.3 Contraintes réseau

Lister les contraintes liées au réseau, en particulier le débit maximum théorique et les découpages en zones de sécurité.

Exemple 1 : le LAN dispose d'un débit maximal de 10 Gbps

Exemple 2 : les composants applicatifs des applications intranet doivent se trouver dans une zone de confiance inaccessible d'Internet.

4.3.4 Contraintes de déploiement

Lister les contraintes liées au déploiement des applications et composants techniques.

Exemple 1 : Une VM ne doit héberger qu'une unique instance PostgreSQL

Exemple 2 : Les applications Java doivent être déployées sous forme de jar exécutable et non de war.

Exemple 3 : Toute application doit être packagées sous forme d'image OCI et déployable sur Kubernetes via un ensemble de manifests structurés au format Kustomize.

4.3.5 Contraintes de logs

Lister les contraintes liées aux logs

Exemple 1 : une application ne doit pas produire plus de 1Tio de logs / mois.

Exemple 2 : la durée de rétention maximale des logs est de 3 mois

4.3.6 Contraintes de sauvegardes et restaurations

Lister les contraintes liées aux sauvegardes.

Exemple 1 : L'espace disque maximal pouvant être provisionné par un projet pour les backups est de 100 Tio sur HDD.

Exemple 2 : la durée de retentions maximale des sauvegardes est de deux ans

Exemple 3 : Compter 1 min / Gio pour une restauration NetBackup.

4.3.7 Coûts

Lister les limites budgétaires.

Exemple 1 : les frais de services Cloud AWS ne devront pas dépasser 5K€/ an pour ce projet.

4.4 Exigences

Contrairement aux contraintes qui fixaient le cadre auquel toute application devait se conformer, les exigences non fonctionnelles sont données par les porteurs du projet (MOA en général).

Prévoir des interviews pour les recueillir.

Si certaines exigences ne sont pas réalistes, le mentionner dans le document des points non statués.

Les exigences liées à la disponibilité devraient être précisées via une étude de risque (type EBIOS Risk Manager)

4.4.1 Plages de fonctionnement

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

Penser aux utilisateurs situés dans d'autres fuseaux horaires.

Les informations données ici serviront d'entrants au SLA de l'application.

TABLE 4.5: Exemple plages de fonctionnement

| No plage | Heures | Détail |
|----------|---|--|
| 1 | <i>De 8H00-19H30 heure de Paris , 5J/7 jours ouvrés</i> | <i>Ouverture Intranet aux employés de métropole</i> |
| 2 | <i>De 21h00 à 5h00 heure de Paris</i> | <i>Plage batch</i> |
| 3 | <i>24 / 7 / 365</i> | <i>Ouverture Internet aux usagers</i> |
| 4 | <i>De 5h30-8h30 heure de Paris, 5J/7 jours ouvrés</i> | <i>Ouverture Intranet aux employés de Nouvelle Calédonie</i> |

4.4.2 Exigences de 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 de l'application.

Attention à bien cadrer ces exigences car un porteur de projet a souvent tendance à demander une disponibilité très élevée sans toujours se rendre compte des implications. 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.

On estime en général que la haute disponibilité (HA) commence à deux neufs (99%), c'est à dire environ 90h d'indisponibilité par an.

Donner la disponibilité demandé par plage.

La disponibilité exigée ici devra être en cohérence avec les Contraintes sur la disponibilité du SI.

TABLE 4.6: Durée d'indisponibilité maximale admissible par plage

| No Plage | Indisponibilité maximale |
|----------|----------------------------|
| 1 | 24h, maximum 7 fois par an |
| 2 | 4h, 8 fois dans l'année |
| 3 | 4h, 8 fois dans l'année |

4.4.3 Modes dégradés

Préciser les modes dégradés applicatifs envisagés.

Exemple 1 : Le site *monsie.com* devra pouvoir continuer à accepter les commandes en l'absence du service de logistique.

Exemple 2 : Si le serveur SMTP ne fonctionne plus, les mails seront stockés en base de donnée puis soumis à nouveau suite à une opération manuelle des exploitants.

4.4.4 Exigences de robustesse

La robustesse du système indique sa capacité à ne pas produire d'erreurs lors d'événements exceptionnels comme une surcharge ou la panne de l'un de ses 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 exigeant sur ce point car une grande robustesse peut impliquer la mise en place de systèmes à tolérance de panne complexes, coûteux et pouvant aller à l'encontre des capacités de montée en charge, voire même de la disponibilité.

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 la section ???.

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

4.4.5 Exigences de RPO

La sauvegarde (ou backup) consiste à recopier les données d'une système sur un support dédié en vue d'une restauration en cas de perte. Ces données sont nécessaires au système pour fonctionner.

Donner ici le Recovery Point Objective (RPO) de l'application. Il peut être utile de restaurer suite à :

- Une perte de données matérielle (peu probable avec des systèmes de redondance).
- Une fausse manipulation d'un power-user ou d'un administrateur (assez courant).
- Un bug applicatif.
- Une destruction de donnée volontaire (attaque de type ransomware)...

Exemple : on ne doit pas pouvoir perdre plus d'une journée de données applicatives

4.4.6 Exigences d'archivage

L'archivage est la recopie de données importantes sur un support dédié offline en vue non pas d'une restauration comme la sauvegarde mais d'une *consultation* occasionnelle. Les archives sont souvent exigées pour des raisons légales et conservées trente ans ou plus.

Préciser si des données de l'application doivent être conservées à long terme. Préciser les raisons de cet archivage (légales le plus souvent).

Préciser si des dispositifs spécifiques de protection de l'intégrité (pour empêcher toute modification principalement) doivent être mis en place.

Exemple 1: comme exigé par l'article L.123-22 du code de commerce, les données comptables devront être conservées au moins dix ans.

Exemple 2 : 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. Une empreinte SHA256 sera calculée au moment de l'archivage et stockée séparément pour vérification de l'intégrité des documents en cas de besoin.

4.4.7 Exigences de purges

Il est 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).

Les purges peuvent également être imposées par la loi. Le RGPD apporte depuis 2018 de nouvelles contraintes sur le droit à l'oubli pouvant affecter la durée de rétention des informations personnelles.

Il est souvent judicieux d'attendre la MEP voire plusieurs mois d'exploitation pour déterminer précisément les durées de rétention (âge ou volume maximal par exemple) mais il convient de prévoir le principe même de l'existence de purges dès la définition de l'architecture de l'application. En effet, l'existence de purges a souvent des conséquences importantes 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 dossiers de plus de six mois seront purgés (après archivage)

4.4.8 Exigences de déploiements et de mise à jour

4.4.8.1 Coté serveur

Préciser ici comment l'application devra être déployée coté serveur.

Par exemple :

- L’installation est-elle manuelle ? scriptées avec des outils d’IT Automation comme Ansible ou SaltStack ? via des images Docker ?
- Comment sont déployés les composants ? Sous forme de paquets ? Utilise-t-on un dépôt de paquets (type yum ou apt) ? Utilise-t-on des conteneurs ?
- Comment sont appliquées les mises jour ?

4.4.8.2 Coté client

Préciser ici comment l’application devra être déployée coté client :

- 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 ?

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 paquet ?
- Comment sont appliquées les mises jour ?

4.4.8.3 Stratégie de déploiement spécifiques

- Prévoit-on un déploiement de type blue/green ?
- Prévoit-on un déploiement de type canary testing ? si oui, sur quel critère ?
- Utilise-t-on des feature flags ? si oui, sur quelles fonctionnalités ?

Exemple: L’application sera déployée sur un mode blue/green, c’est à dire complètement installée sur des machines initialement inaccessibles puis une bascule DNS permettra de pointer vers les machines disposant de la dernière version.

4.4.9 Exigences de gestion de la concurrence

Préciser ici les composants internes ou externes pouvant interférer avec l’application.

Exemple 1 : Tous les composants de cette application doivent pouvoir fonctionner en concurrence. En particulier, la concurrence batch/IHM 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.

4.4.10 Exigences d'écoconception

L'écoconception consiste à limiter l'impact environnemental des logiciels et matériels utilisés par l'application. Les exigences dans ce domaine s'expriment généralement en WH ou équivalent CO2.

Prendre également en compte les impressions et courriers.

Selon l'ADEME (estimation 2014), les émissions équivalent CO2 d'un KWH en France continentale pour le tertiaire est de 50g/KWH.

Exemple 1 : La consommation électrique moyenne causée par l'affichage d'une page Web ne devra pas dépasser 10mWH, soit pour 10K utilisateurs qui affichent en moyenne 100 pages 200 J par an : $50 \text{ g/KWH} \times 10\text{mWH} \times 100 \times 10\text{K} \times 200 = 100 \text{ Kg équivalent CO2 / an.}$

Exemple 2 : Le Power usage effectiveness (PUE) du site devra être de 1.5 ou moins.

Exemple 3 : La consommation d'encre et de papier devra être réduite de 10% par rapport à 2020.

4.5 Architecture cible

4.5.1 Principes

Quels sont les grands principes d'infrastructure de notre application ?

Exemples :

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

4.5.2 Disponibilité

La disponibilité est le pourcentage de temps minimal sur une année pendant lequel un système doit être utilisable dans des conditions acceptables. Il est exprimé en % (exemple: 99.9%).

Donner ici les dispositifs permettant d'atteindre les Exigences de disponibilité.

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 MTDD (temps moyen de détection).
- **Dispositifs organisationnels** :
 - la présence humaine (astreintes, heures de support étendues...) qui permet d'améliorer le MTTR (temps moyen de résolution) et sans laquelle la supervision est inefficace ;
 - 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 affecte le MTTR.

- Dispositifs de **haute disponibilité (HA)** (clusters, RAID...) qu'il ne faut pas surestimer si les dispositifs précédents sont insuffisants.
- Dispositifs de **restauration de données** : la procédure de restauration est-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 dans cette situation, pouvoir restaurer rapidement la dernière sauvegarde améliore grandement le MTTR.

Principes de disponibilité et de redondance:

- La **disponibilité d'un ensemble de composants en série** : $D = D1 * D2 * \dots * Dn$. Exemple : la disponibilité d'une application 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** : $D = 1 - (1-D1) * (1-D2) * \dots * (1-Dn)$. 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 d'application JEE si tous ces serveurs attaquent une base de donnée 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 à l'application ou mutualisé au niveau SI.
- Les **niveaux de redondance** d'un système (modèle NMR = N-Modular Redundancy) les plus courants sont les suivants (avec N, le nombre de dispositifs assurant un fonctionnement correct en charge) :
 - **N** : aucune redondance (exemple : si l'alimentation unique d'un serveur tombe, le serveur s'arrête)
 - **N+1** : un composant de rechange est disponible, on peut supporter la panne d'un matériel (exemple : on a une alimentation de spare disponible).
 - **N+M** : Un seul spare n'est pas suffisant pour tenir la charge, on prévoit au moins M spares.
 - **2N** : le système est entièrement redondé et peut supporter la perte de la moitié des composants (exemple : on dispose de deux alimentations actives en même temps dont chacune suffit à alimenter le serveur). Ce système est tolérant aux pannes (Fault-tolerant).
 - **2N+1** : En plus d'un système entièrement redondé, un système de secours est disponible (pour les opérations de maintenance par exemple).

Clustering:

- Un cluster est un **ensemble de nœuds (machines) hébergeant le même module applicatif**.
- En fonction du niveau de disponibilité recherché, chaque nœud peut être :
 - **actif** : le nœud traite les requêtes (exemple : un serveur Apache parmi dix et derrière un répartiteur de charge). Temps de failover : nul ;
 - **passif en mode «hot standby»** : le nœud est installé et démarré mais ne traite pas les requêtes (exemple: une base MySQL slave qui devient master en cas de panne de ce dernier via l'outil mysqlfailover). MTTR de l'ordre de la seconde (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 (exemple : un serveur avec une instance Tomcat éteinte hébergeant notre application). En cas de panne, notre application est démarrée automatiquement. MTTR : de l'ordre de la minute (temps de la détection de la panne et d'activation de l'application) ;
- **passif en mode «cold standby»** : le nœud est un simple spare. Pour l'utiliser, il faut installer l'application et la démarrer. MTTR : 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.
- On peut classer les architectures de clusters actif/actif en deux catégories :
 - 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. Note : 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** dans lesquels tous les nœuds partageant les mêmes données en mémoire. Dans cette architecture, toute donnée de contexte doit être répliquée dans tous les nœuds (ex : cache distribué de sessions HTTP répliqué avec JGroups).

Failover:

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 défectueux mais vers un nœud opérationnel. Ce **processus est automatique**.

Sans failover, c'est au client de détecter la panne et de se reconfigurer pour rejouer sa requête vers un nœud actif. Dans les faits, ceci est rarement praticable et les **clusters disposent presque toujours de capacités de failover**.

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

- Quelle **stratégie de failback** ? Par exemple: "Fail fast" (un nœud est noté comme tombé dès le premier échec); "On fail, try next one" ; "On fail, try all"...
- Quelle **solution de détection des pannes** ?
 - les répartiteurs de charge utilisent des **sondes** (health check) très variées (requêtes fictives, analyse du CPU, des logs, etc...) vers les nœuds ;
 - les détections de panne des clusters actifs/passifs fonctionnent la plupart du temps par écoute des palpitations (**heartbeat**) du serveur actif par le serveur passif, par exemple via des requêtes multicast UDP dans le protocole VRRP utilisé par keepalived.
- Quel **délai de détection** de la panne ? il convient de paramétrer correctement les solutions de détection de panne (le plus court possible sans dégradation de performance).
- Quelle **pertinence de la détection** ? le serveur en panne est-il **vraiment** en panne ? un mauvais paramétrage ou une microcoupure réseau ne doit pas provoquer une indisponibilité totale d'un cluster alors que les nœuds sont sains.
- Quelle stratégie de failback ?
 - 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 AWS Lambda ou CaaS comme Kubernetes) : on distribue les applications du nœud

en panne vers d'autres nœuds actifs (le cluster ayant été dimensionné en prévision de cette éventuelle surcharge).

- **Transparent via à vis de l'appelant** ou pas ? En général, les requêtes pointant vers un serveur dont la panne n'a pas encore été détectée tombent en erreur (en timeout la plupart du temps). Certains dispositifs ou architectures de FT (tolérance de panne) permettent de le rendre transparent pour le client.

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 pas actif/actif sinon on ne fait que déplacer le problème et il faudrait un "répartiteur de répartiteurs"). Le répartiteur passif doit surveiller à fréquence élevée le répartiteur actif et le remplacer dès qu'il tombe.
- 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 (exemple : option `redispatch` de HAProxy) permettent la transparence vis à vis de l'appelant en configurant 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.
- Lisser la charge entre les nœuds et ne pas forcément 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). Attention néanmoins à bien les tester et en maîtriser les implications pour éviter les catastrophes.
- Dans le monde Open Source, voir par exemple LVS + `keepalived` ou HAProxy + `keepalived`.

La **tolérance de panne** :

La tolérance de panne (FT = Fault Tolerance) ne doit pas être confondue avec la Haute Disponibilité. Il s'agit d'une version plus stricte de HA où la **disponibilité est de 100% et aucune donnée ne peut être perdue** (Wikipédia: "La tolérance aux pannes est la propriété qui permet à un système de continuer à fonctionner correctement en cas de défaillance d'un ou de certains de ses composants"). Seuls les systèmes critiques (santé, militaires, transport, industrie...) ont en général besoin d'un tel niveau de disponibilité.

Historiquement, cela signifiait une redondance matérielle complète. Dans un monde de micro-services, cela peut également être réalisé au niveau logiciel avec des clusters actifs-actifs. De plus, un véritable système de tolérance aux pannes devrait éviter une dégradation significative des performances vue par les utilisateurs finaux.

Par exemple, un lecteur RAID 1 offre une tolérance aux pannes transparente : en cas de panne, le processus écrit ou lit sans erreur après le basculement automatique sur le disque sain. Un cluster Kubernetes peut également atteindre la tolérance aux pannes en démarrant de nouveaux POD. Ou encore, un cache distribué en mémoire en cluster peut éviter de perdre une session HTTP.

Pour permettre la tolérance de panne d'un cluster, il faut obligatoirement **disposer d'un cluster actif/actif avec fort couplage** dans lequel les données de contexte sont répliquées à tout moment. Une autre solution (bien meilleure) est d'éviter tout simplement les données de contexte (en gardant les données de session dans le navigateur via un client JavaScript 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 totalement transparente, il faut en plus prévoir un répartiteur de charge assurant les rejeux lui-même.

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

- **Complexifient l'architecture** et la rendent donc moins robuste et plus coûteuse à construire, tester, exploiter.
- **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 ou un REDIS avec persistance sur le master), toute mise à jour transactionnelle 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 ?
- **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.

TABLE 4.7: Quelques solutions de disponibilité

| Solution | Coût | Complexité de mise en œuvre indicative | Amélioration de la disponibilité indicative |
|---|------|--|---|
| Disques en RAID 1 | XXX | X | XXX |
| Disques en RAID 5 | X | X | XX |
| Redondance des alimentations et autres composants | XX | X | XX |
| Bonding des cartes Ethernet | XX | X | X |
| Cluster actif/passif | XX | XX | XX |
| Cluster actif/actif (donc avec LB) | XXX | XXX | XXX |
| Serveurs/matériels de spare | XX | X | XX |
| Bonne supervision système | X | X | XX |
| Bonne supervision applicative | XX | XX | XX |
| Systèmes de test de vie depuis un site distant | X | X | XX |

| Solution | Coût | Complexité de mise en œuvre indicative | Amélioration de la disponibilité indicative |
|--|------|--|---|
| Astreintes dédiées à l'application, 24/7/365 | XXX | XX | XXX |
| Copie du backup du dernier dump de base métier sur baie SAN (pour restauration expresse) | 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 applications.
- 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 DC de niveau tier 3) :

- Tous les serveurs en RAID 1 + alimentations redondées + interfaces en bonding.
- Répartiteur HAProxy + keepalived actif/passif dédié à l'application.
- Cluster actif /actif de 4 serveurs (soit une redondance 2N) Apache + mod_php.
- Instance Oracle en RAC sur deux machines (avec interconnexion FC dédiée).

4.5.3 Déploiement en production

Fournir ici le modèle de déploiement des composants en environnement cible sur les différents intergiciels et nœuds physiques (serveurs). Ne représenter les équipements réseau (pare-feu, appliances, routeurs...) que s'ils aident à la compréhension.

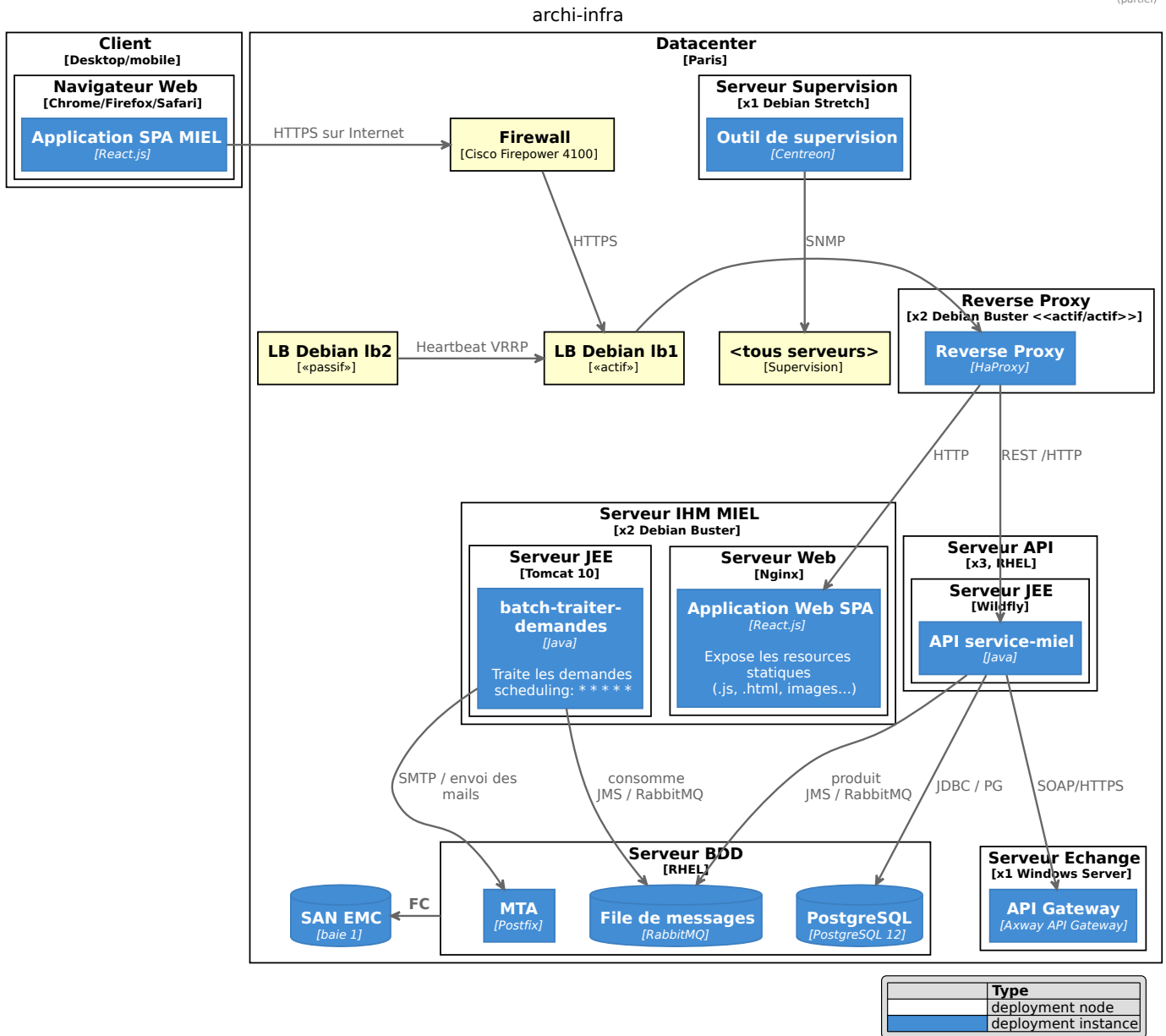
Tout naturellement, on le documentera de préférence avec un diagramme de déploiement UML2 ou un diagramme de déploiement C4.

Pour les clusters, donner le facteur d'instanciation de chaque nœud.

Donner au besoin en commentaire les contraintes d'affinité (deux composants doivent s'exécuter sur le même nœud ou le même intergiciel) ou d'anti-affinité (deux composants ne doivent pas s'exécuter sur le même nœud ou dans le même intergiciel).

Identifier clairement le matériel dédié à l'application (et éventuellement à acheter).

Exemple :



4.5.4 Versions des composants d'infrastructure

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

TABLE 4.8: Exemple de composants d'infrastructure

| 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 |

| Composant | Rôle | Version | Environnement technique |
|-------------------|----------------------------------|---------------|-------------------------|
| <i>PHP +</i> | <i>Pages dynamiques de l'IHM</i> | <i>5.6.29</i> | <i>nginx</i> |
| <i>php5-fpm</i> | <i>XYZ</i> | | |
| <i>PostgreSQL</i> | <i>SGBDR</i> | <i>9.3.15</i> | <i>CentOS 7</i> |

4.5.5 Matrice des flux techniques

Lister 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 organisations, 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 pas nécessaire de faire référence aux flux applicatifs car les lecteurs ne recherchent pas les mêmes informations. Ici, les exploitants ou les intégrateurs recherchent l'exhaustivité des flux à fin d'installation et de configuration des pare-feu par exemple.

Les types de réseaux incluent les informations utiles sur le réseau utilisé afin d'apprécier les performances (TR, latence) et la sécurité: LAN, VLAN, Internet, LS, WAN. . .)

TABLE 4.9: Exemple partiel de matrice de flux techniques

| ID | Source | Destination | Type de réseau | Protocole | Port d'écoute |
|----------|----------------------------|--|----------------|-------------------------------|---------------|
| <i>1</i> | <i>lb2</i> | <i>IP multicast</i> <i>224.0.0.18</i> | <i>LAN</i> | <i>VRRP</i> <i>sur UDP</i> | <i>3222</i> |
| <i>2</i> | <i>lb1</i> | <i>host1, host2</i> | <i>LAN</i> | <i>HTTP</i> | <i>80</i> |
| <i>3</i> | <i>host3, host4, host5</i> | <i>bdd1</i> | <i>LAN</i> | <i>PG</i> | <i>5432</i> |
| <i>4</i> | <i>sup1</i> | <i>host[1-6]</i> | <i>LAN</i> | <i>SNMP</i> | <i>199</i> |

4.5.6 Environnements

Donner ici une vision générale des environnements utilisés par l'application. Les environnements les plus communs sont : développement, recette, pré-production/benchmarks, production, formation.

Dans les gros SI, il est souvent utile de segmenter les environnements en 'plateformes' (ou 'couloirs') constituées d'un ensemble de composants techniques isolés les uns des autres (même s'il peuvent partager des ressources communes comme des VM suivant la politique de l'organisation). Par exemple, un environnement de recette peut être constitué des plateformes UAT1 et UAT2 permettant à deux testeurs de travailler en isolation.

TABLE 4.10: Environnements

| Environnement Rôle | Contenu | Plateforme |
|--|---|------------|
| Développement Déploiement continu (CD) pour les développeurs | Branche develop déployée à chaque commit | Un seul |

| Environnement Rôle | | Contenu | Plateforme |
|--------------------|--|---------------------------------------|--------------|
| Recette | Recette fonctionnelle par les testeurs | Tag déployé à la fin de chaque Sprint | UAT1 et UAT2 |

4.6 Écoconception

Lister ici les mesures d'infrastructure permettant de répondre aux Exigences d'écoconception.

Les réponses à ses problématiques sont souvent les mêmes que celles aux exigences de performance (temps de réponse en particulier) et à celles des coûts (achat de matériel). Dans ce cas, y faire simplement référence.

Néanmoins, les analyses et solutions d'écoconception peuvent être spécifiques à ce thème. Quelques pistes d'amélioration de la performance énergétique :

- Mesurer la consommation électrique des systèmes avec les sondes PowerAPI (développé par l'INRIA et l'université Lille 1).
- Utiliser des caches (cache d'opcode, caches mémoire, caches HTTP...).
- Pour des grands projets ou dans le cadre de l'utilisation d'un cloud CaaS, l'utilisation de cluster de containers (solution type Swarm, Mesos ou Kubernetes) permet d'optimiser l'utilisation des VM ou machines physiques en les démarrant / arrêtant à la volée de façon élastique.
- Héberger ses serveurs dans un datacenter performant. Les fournisseurs de cloud proposent en général des datacenters plus performants que on-premises. L'unité de comparaison est ici le PUE (Power Usage Effectiveness), ratio entre l'énergie consommée par le datacenter et l'énergie effectivement utilisée par les serveurs (donc hors refroidissement et dispositifs externes). OVH propose par exemple des datacenter avec un PUE de 1.2 en 2017 contre 2.5 en moyenne.
- Néanmoins :
 - vérifier l'origine de l'énergie (voir par exemple les analyses de Greenpeace en 2017 sur l'utilisation d'énergie issue du charbon et du nucléaire par Amazon pour son cloud AWS) ;
 - garder en tête que l'énergie consommée par l'application coté client et réseau est très supérieure à celle utilisée coté serveur (par exemple, on peut estimer qu'un serveur consommant à peine plus qu'une station de travail suffit à plusieurs milliers voire dizaines de milliers d'utilisateurs). La réduction énergétique passe aussi par un allongement de la durée de vie des terminaux et l'utilisation de matériel plus sobre.

Exemple 1 : la mise en place d'un cache Varnish devant notre CMS réduira de 50% le nombre de construction de pages dynamiques PHP et permettra l'économie de deux serveurs.

Exemple 2 : L'application sera hébergée sur un cloud avec un PUE de 1.2 et une origine à 80 % renouvelable de l'énergie électrique.

4.6.1 Régulation de la charge

4.6.1.1 Coupe-circuits

Dans certains cas, des pics extrêmes et imprévisibles sont possibles (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 et permet de gérer le problème et non de le subir car on assure un bon fonctionnement acceptable aux utilisateurs déjà connectés.

4.6.1.2 Qualité de Service

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

- Via du throttling (écrêtage du 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 1 : 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 espacer progressivement dans le temps).

TABLE 4.11: Exemple : répartition des jetons sera la suivante par défaut

| Opération sur <code>DetailArticle</code> | Proportion des jetons |
|--|-----------------------|
| GET | 80% |
| POST | 5% |
| PUT | 15% |

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

4.6.2 Gestion des timeouts

Décrire ici les différents timeouts mis en œuvre sur les chaînes de liaison. Garder en tête que dans une chaîne de liaison allant du client à la persistance, les timeouts devraient diminuer au fur et à mesure qu'on avance dans la chaîne de liaison (exemple: 10 secs sur le Reverse proxy , 8 secs sur le endpoint REST, 5 secs sur la base de donnée).

En effet, dans le cas contraire, un composant technique peut continuer à traiter une requête alors que son composant appelant a déjà abandonné, ce qui pose à la fois des problèmes de gaspillage de ressource mais surtout des effets difficile à prévoir.

Éviter également d'utiliser la même valeur dans tous les composants techniques pour éviter les effets inattendus lié à la concomitance des timeouts.

Exemple :

| Composant | Timeout (ms) |
|------------------------|--------------|
| Client Rest JavaScript | 5000 |
| API Gateway | 4000 |
| API Rest Node.js | 3500 |
| Base de donnée PG | 3000 |

4.6.3 Exploitation

Lister ici les grands principes d'exploitation de la solution. Les détails (filesystems sauvegardés, plan de production, planification des traitements...) seront consigné dans un DEX (Dossier d'EXploitation) séparé.

Si cette application reste dans le standard de l'organisation, se référer simplement à un dossier commun.

4.6.3.1 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 externes 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 d'éventuelles problématiques en cas de démarrage partiel (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 d'ordre de démarrage :

1. pg1 sur serveur bdd1
2. mq1 sur bdd1
3. services1 sur serveurs host3, host4 et host5
4. services2 sur serveurs host3, host4 et host5
5. batchs sur serveurs host1, host2
6. ihm sur serveurs host1, host2

Exemple d'ordre d'arrêt :

Inverse exact du démarrage

4.6.3.2 Opérations programmées

Lister de façon macroscopique (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 (purgas, 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.

4.6.3.3 Mise en maintenance

Expliquer (si besoin) les dispositifs et procédures permettant de mettre l'application 'offline' de façon explicite pour les utilisateurs.

Exemple : Nous utiliserons le F5 BigIp LTM pour afficher une page d'indisponibilité.

4.6.3.4 Sauvegardes et restaurations

Donner la politique générale de sauvegarde. Elle doit répondre aux Exigences de RPO. De même les dispositifs de restauration doivent être compatibles avec les Exigences de disponibilité :

- 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 des répertoires ?
 - des bases de données sous forme de dump ? sous forme binaire ?
 - les logs ? les traces ?
- Les sauvegardes sont-elles chiffrées ? si oui, chiffrement de la partition toute entière, fichier par fichier, les deux ? Faut-il chiffrer également le nom des répertoires et fichiers sauvegardés ? Préciser l'algorithme de chiffrement utilisé et comment seront gérées les clés.
- Les sauvegardes sont-elles compressées ? si oui, avec quel algorithme ? (lzw, deflate, lzma?, ...), avec quel niveau de compression ? attention à trouver le compromis entre durée de compression / décompression et gain de stockage.
- Quel outillage est mis en œuvre ? Simple cron + tar/rsync ? Outil Open Source comme « backup-manager » ? IBM TSM ? Outil orienté Cloud comme « Duplicity » ou « Restic » ?, etc.
- 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 ? Sont-ils facilement accessibles ? Contiennent-ils des informations sensibles ?
- Où sont stockées les sauvegardes ? (idéalement le plus loin possible du système sauvegardé tout en permettant une restauration dans un temps compatible avec le RTO).
- Qui accède physiquement aux sauvegardes et à ses logs ? à la clé de chiffrement ? (penser aux exigences de confidentialité).

Il est conseillé :

- d'utiliser un support distinct des données sources (ne pas sauvegarder sur un disque HD1 des données de ce même disque).
- de disposer d'au moins deux supports de stockage distincts si les données sont vitales à l'organisation.
- de faire en sorte que les sauvegardes ne soient pas modifiables par la machine qui a été sauvegardée (par exemple, une sauvegarde sur NAS peut être supprimée par erreur en même temps que les données sauvegardées)

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.

Enfin, il est important de garder à l'esprit que ce que nous voulons *vraiment*, ce sont des restaurations, pas des sauvegardes. Il est crucial de s'assurer que la restauration sera fonctionnelle :

- Les sauvegardes sont-elles correctes et complètes ?
- Quels tests de restauration sont prévus ? à quelle fréquence (une fois par an est un minimum) ?
- Combien de temps va prendre une restauration (benchmarks) ? Est-ce compatible avec le RTO ?
- Avons nous des dépendances externes pouvant nous ralentir (coffre accessible en journée uniquement par exemple) ?
- Avons nous suffisamment de ressources pour la restauration (stockage intermédiaires, CPU et mémoire pour la décompression/déchiffrement , etc...) ?

4.6.3.5 Archivage

Décrire ici les dispositifs permettant de répondre aux ??? avec les modalités de stockage suivantes :

- 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 : les relevés bancaires de plus de 10 ans seront archivés sur bande LTO et disque dur. Les deux supports seront stockés en coffre dans deux banques différentes.

4.6.3.6 Purgés

Donner ici les dispositifs techniques répondant aux ???.

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 requête SQL `DELETE` après validation par l'exploitant de la complétude du dump.

4.6.3.7 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 ? (indispensable pour les architectures SOA ou micro-services). Voir par exemple la stack ELK.
- 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) ?

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.

4.6.3.8 Supervision

La supervision est un pilier central de la disponibilité en faisant diminuer drastiquement le MTDD (temps moyen de détection de la panne).

Idéalement, elle ne sera pas uniquement réactive mais également proactive (detection des signaux faibles).

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 disposant de seuils (ex : 'niveau critique si l'utilisation de CPU sur le serveur s1 reste au delà de 95% pendant plus de 5 minutes').

4.6.3.8.1 Supervision technique Lister les métriques :

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

Exemple : on mesure le % de wait io et la charge serveur.

4.6.3.8.2 Supervision applicative Lister les métriques applicatives (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. . .

4.6.3.8.3 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 de séries chronologiques (comme RRD).
- Consolide les indicateurs depuis les métriques.
- Affiche les tendances dans le temps de ces indicateurs.
- Permet de fixer des seuils d’alerte basés sur les indicateurs et de notifier les exploitants en cas de dépassement.

Exemple : la pilotage de la supervision se basera sur la plate-forme Grafana.

4.6.3.8.4 Alerting Préciser ici les conditions d’alertes et le canal utilisé

Exemple : SMS si aucune demande depuis les 4 dernières heures ou si le nombre d’erreurs techniques d’un composant dépasse 10/h.

4.6.3.8.5 Suivi des opérations programmées Indiquer l’ordonnanceur ou le planificateur utilisé pour piloter les jobs et consolider le plan de production (exemple : VTOM, JobScheduler, Dollar Universe, Control-M. . .). Détailler les éventuelles spécificités de l’application :

- Degré de parallélisme des jobs
- Plages de temps obligatoires
- Rejeux en cas d’erreur
- . . .

Les jobs doivent-ils produire un rapport d’exécution ? sous quelle forme et avec quel contenu ?

Exemple 1 : les jobs seront ordonnancés par l’instance JobScheduler de l’organisation.

- Les jobs ne devront jamais tourner les jours fériés.
- 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 job J1 en parallèle.

Exemple 2 : Les jobs devront produire un rapport d’exécution à chaque lancement (avec des données de base comme le nombre d’éléments traités, la durée du traitement et tout indicateur pertinent).

4.6.3.8.6 Supervision boîte noire Il est également fortement souhaitable et peu coûteux de prévoir un système de tests de supervision boîte-noire (via des scénarios déroulés automatiquement). L’idée est ici de tester un système dans son ensemble et avec une vue end-user la plus externe possible (à l’inverse d’une supervision whitebox pour laquelle on supervise des composants bien précis avec un comportement attendu).

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 de mise à jour. Si tel est le cas, il faudra être en mesure d’identifier dans tous les composants 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 supervision boîte noire seront mis en œuvre via des requêtes HTTP lancées via l'outil uptrends.com. En cas de panne, un mail est envoyé aux exploitants.

4.6.3.8.7 Métrologie 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 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 à la supervision boîte noire (voir plus haut).

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

4.6.4 Migration

Ce chapitre permet de décrire une éventuelle migration depuis un ancien système.

Décrire de façon macroscopique la procédure envisagée ainsi que les retours arrières prévus.

Décrire éventuellement un fonctionnement 'à blanc' en parallèle de l'ancien système avant activation.

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.

4.6.5 Décommissionnement

Ce chapitre sera instruit quand l'application arrive en fin de vie et devra être supprimée ou remplacée. Il décrit entre autres :

- Les données à archiver ou au contraire à détruire avec un haut niveau de confiance.
- Les composants physiques à évacuer ou à détruire.
- Les procédures de désinstallation côté 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 de matériel).

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

Chapitre 5

Volet architecture sécurité

Dernière modification : 2022-05-03

5.1 Introduction

Ceci est le point de vue sécurité. Il décrit l'ensemble des dispositifs mis en œuvre pour empêcher l'utilisation non-autorisée, le mauvais usage, la modification illégitime ou le détournement des modules applicatifs.

Les autres volets du dossier sont accessibles d'ici.

Le glossaire du projet est disponible ici. Nous ne redéfinirons pas ici les termes fonctionnels ou techniques utilisés.

La disponibilité est traitée dans le volet infrastructure.

5.1.1 Documentation de Référence

Mentionner ici les documents d'architecture de référence (mutualisés). Ce document ne doit en aucun cas reprendre leur contenu sous peine de devenir rapidement obsolète et non maintenable.

TABLE 5.1: Références documentaires sécurité

| N° | Version | Titre/URL du document | Détail |
|----|---------|---------------------------------|--|
| 1 | 1.0 | <i>Dispositifs_securite.pdf</i> | <i>Catalogue de dispositifs de sécurité habilités</i> |
| 2 | latest | <i>Normes sécurité société</i> | <i>http://masociete/monurl</i> |

5.2 Non statué

5.2.1 Points soumis à étude complémentaire

TABLE 5.2: Points soumis à étude complémentaire

| ID | Détail | Statut | Porteur du sujet | Échéance |
|------------|--|-----------------|------------------------|-------------------|
| <i>ES1</i> | <i>Il conviendra de valider que les dispositifs anti-CSRF mis en place résolvent également les failles liées au couplage TLS + compression (type CRIME ou BREACH).</i> | <i>EN_COURS</i> | <i>Équipe sécurité</i> | <i>AVANT 2040</i> |

5.2.2 Hypothèses

TABLE 5.3: Hypothèses

| ID | Détail |
|------------|---|
| <i>HS1</i> | <i>La solution SAML actuellement en place dans l'organisation ne permet pas de répondre aux besoins d'authentification exprimés pour cette application Internet. Une solution OpenID Connect sera proposée.</i> |

5.3 Contraintes

Lister ici les contraintes relatives à la sécurité, ceci inclut par exemple mais pas seulement :

- L'isolation des composants au sein de zones réseaux étanches (DMZ, pare-feux, reverse-proxy...)
- Les normes applicables (comme les politiques de mot de passe)
- Les contraintes légales (RGPD par exemple)

Exemple 1 : la politique de mot de passe devra se conformer à la norme xyz

Exemple 2 : il est formellement interdit à un composant de la zone internet d'accéder à la zone intranet

Exemple 3 : en application du RGPD, les données utilisateur devront être chiffrées

5.4 Exigences

Présenter ici les exigences, **pas les dispositifs y répondant**. Ceux-ci seront détaillées au chapitre 3.

Pour les projets particulièrement sensibles, prévoir un dossier d'analyse de risque. Pour cela, utiliser par exemple la méthode EBIOS Risk Manager (Expression des Besoins et Identification des Objectifs de Sécurité).

5.4.1 Exigences d'intégrité

L'intégrité concerne la durabilité, la justesse et le niveau de confiance dans les données de l'application.

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, par exemple dans le cadre d'une attaque de type "man in the middle" ou par une personne s'étant octroyé des droits indus.

Attention à ne pas multiplier les classes de données. Il est possible de ne définir qu'une seule classe de donnée pour l'ensemble de l'application (cas courant).

TABLE 5.4: Niveau d'intégrité exigée par classe de données

| <i>Classe de données</i> | <i>Niveau « Non intégr » (La donnée peut ne pas être totalement intégr)</i> | <i>Niveau « Délectable » (La donnée peut ne pas être intégr si l'altération est identifiée dans un délai raisonnable)</i> | <i>Niveau « Maîtrisé » (La donnée peut ne pas être intégr, si l'altération est identifiée et l'intégrité du bien essentiel retrouvée)</i> | <i>Niveau « Intégr » (La donnée doit toujours être rigoureuse- ment intégr)</i> |
|--|---|---|---|---|
| <i>Données de la base métier</i> | | | | <i>X</i> |
| <i>Données archivées</i> | | <i>X</i> | | |
| <i>Données calculées stats entreprises</i> | | | <i>X</i> | |
| <i>Silo NoSQL des données Big Data avant consolidation</i> | <i>X</i> | | | |
| <i>Sources de l'application</i> | | | | <i>X</i> |
| <i>Avis d'imposition en PDF</i> | | | | <i>X</i> |

5.4.2 Exigences de confidentialité

La confidentialité est le fait de s'assurer que l'information n'est accessible qu'à ceux dont l'accès est autorisé (norme 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 de l'application (cas courant).

TABLE 5.5: Niveau de confidentialité exigée par classe de données

| <i>Classe de données</i> | <i>Niveau « Public » (Tout le monde peut accéder à la donnée)</i> | <i>Niveau Limité » (La donnée n'est accessible qu'aux personnes habilitées)</i> | <i>Niveau « Réservé » (La donnée n'est accessible qu'au personnel interne habilité)</i> | <i>Niveau « Privé » (La donnée n'est visible que par l'intéressé(e))</i> |
|---|---|---|---|--|
| <i>Contenu éditorial</i> | <i>X</i> | | | |
| <i>Profil du compte du site Web</i> | | <i>X</i> | | |

TABLE 5.5: Niveau de confidentialité exigée par classe de données

| | | |
|---|----------|----------|
| <i>Historique du compte</i> | <i>X</i> | |
| <i>Logs techniques des activités de l'internaute</i> | <i>X</i> | |
| <i>Données RH de type "aides sociales aux employés"</i> | | <i>X</i> |

5.4.3 Exigences d'identification

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

Exemple 1 : Un utilisateur ne peut avoir qu'un identifiant et un identifiant ne peut être partagé par plusieurs utilisateurs. L'adresse e-mail personnelle est donc un bon identifiant.

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

5.4.4 Exigences d'authentification

L'authentification permet de vérifier la cohérence entre l'identité d'un utilisateur et une personne physique se connectant.

A noter que les dispositifs techniques (comme les batchs) peuvent également faire l'objet d'identification et d'authentification (batch qui utilise un access-token pour appeler un service par exemple).

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** (classiquement un mot de passe).
- Quelque chose qu'on **est** (biométrie).
- Quelque chose qu'on **possède** (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 (authentification initiale).

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

Il est bien sûr possible d'ajouter au besoin dans le tableau ci-dessous des facteurs d'authentification spécifiques à votre organisation.

Les facteurs d'authentification requis en fonction des situations sont (on peut exiger plusieurs occurrences

du même facteur, utiliser autant de croix) :

TABLE 5.6: Exigence d'authentification par cas d'utilisation

| <i>Cas d'authen- tification</i> | <i>Mot de passe respectant la politique de mot de passe P</i> | <i>Clé publique ssh connue</i> | <i>OTP par Token</i> | <i>Biométrie</i> | <i>Connaissance de données métier</i> | <i>E-mail d'activa- tion</i> | <i>Délégation d'authen- tification</i> |
|--|---|--|--------------------------|------------------|---|--------------------------------------|--|
| <i>Utilisateur déjà inscrit</i> | X | | | | | | |
| <i>Création d'un compte</i> | | | | | XX | X | |
| <i>Modification du mot de passe</i> | X | | | | | X | |
| <i>Accès aux logs</i> | | X | | | | | |
| <i>Ajout d'un béné- ficiaire de virement</i> | X | | X | | | | |
| <i>Application mobile</i> | Y | | | | | | X |

5.4.5 Exigences de fédération d'identité

La fédération d'identité est l'utilisation d'une même identité gérée par un identity provider (IdP) depuis plusieurs entités différentes.

Par exemple, France Connect très utilisé par les administrations et basé sur OpenId Connect permet de réutiliser le compte d'une administration pour se loguer sur le compte d'une autre (DGFIP et CNAM par exemple).

Voir aussi les « Connect with [Google|Twitter|...] » en technologie OpenId Connect. Contrairement au SSO, la fédération d'identité n'assure pas un login automatique à une application comme le SSO mais permet simplement de réutiliser les mêmes credentials (login/mot de passe).

Exemple : L'identification et l'authentification seront externalisés au fournisseur d'identité Auth0 pour simplifier la gestion de la sécurité et réduire les coûts de développement et d'exploitation.

5.4.6 Exigences de SSO et SLO

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

Nous entendons ici SSO dans son sens le plus 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 peut être complexe, surtout si l'infrastructure (ID provider...) n'existe pas encore.

Elle nécessite souvent une adaptation des applications.

Le SSO est souvent demandé par les métiers mais cette exigence doit être justifiée.

Une application périphérique ou un outil rarement utilisé n'a en général pas besoin de SSO (une simple authentification centralisée au sein d'un annuaire peut suffire).

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 de l'application 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 (jusqu'à expiration de sa session). De même, une déconnexion depuis l'une des applications doit assurer la déconnexion de toutes les applications de l'intranet.

5.4.7 Exigences de 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 le texte n°2000-230 du 13 mars 2000 du code civil.

TABLE 5.7: Besoins de non-répudiation

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

5.4.8 Exigences d'anonymat et de respect de la vie privée

Lister les contraintes d'anonymat et de vie privée légale (exigée par le RGPD). Voir <https://www.cnil.fr/fr/rgpd-par-ou-commencer>.

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

Exemple 2 : Par soucis de confidentialité 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, politique, syndicales, religieuse 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'utilisateur de la présence de cookies.

Exemple 6 : En application du RGPD, un consentement explicite des utilisateurs dans la conservation de leurs données personnelles de santé sera proposé.

5.4.9 Exigences sur les habilitations

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

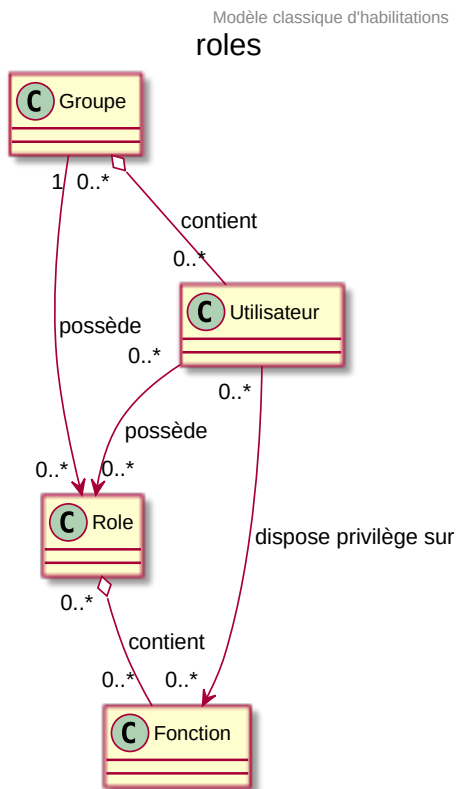
Exemples de fonctions : 'faire un virement inter-bancaire', 'voir l'historique de son compte', 'supprimer un utilisateur'

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 habilitations par factorisation, on peut :

- Regrouper les utilisateurs dans des groupes (comme `G_chef_service`).
- Associer une liste de fonctions à un rôle (comme `R_Administrateur`, `R_banquier_niv1`, `R_chef_service`) qu'on peut affecter à une personne ou à un groupe.

Exemple de modèle classique de gestion des habilitations :



Penser à spécifier les éventuels pseudo-utilisateurs et leurs rôles comme :

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

Préciser si l'application doit utiliser de la délégation d'autorisation (type OAuth2) et si oui, l'application est-elle 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

Exemple 2 : l'application s'appuiera sur une gestion des autorisations matricielle de type [rôles] → [groupes ou utilisateurs] comme décrit plus bas. Le détail des autorisations sera donnée dans les SFD.

TABLE 5.8: Exemple de matrice de rôles

| <i>Groupe ou utilisateur</i> | <i>Rôle suppression</i> | <i>Rôle administration</i> | <i>_Rôle_consultation données de base</i> |
|------------------------------|-------------------------|----------------------------|---|
| <i>Groupe g_usagers</i> | | | <i>X</i> |
| <i>Groupe @anonyme</i> | | | |
| <i>Groupe g_admin</i> | <i>X</i> | <i>X</i> | <i>X</i> |
| <i>Utilisateur xyz</i> | <i>X</i> | | <i>X</i> |

5.4.10 Exigences de traçabilité et d'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 elles-mêmes sensibles et nécessitent souvent un bon niveau de confidentialité.

Différentier :

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

Pour les données les plus sensibles, il est possible de prévoir une traçabilité à deux niveaux (tracer la consultation des traces) pour éviter une traçabilité hiérarchique abusive.

La traçabilité des données des référentiels (base des personnes typiquement) nécessite une historisation complète, ce qui est de toute façon une bonne pratique d'urbanisation (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 X, 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.

TABLE 5.9: Données à conserver pour preuves

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

5.5 Mesures de sécurité

5.5.1 Intégrité

Dispositifs répondant aux exigences d'intégrité :

TABLE 5.10: Mesures pour assurer le niveau d'intégrité demandée

| Classe de données | Niveau exigé | Mesures |
|----------------------------------|----------------|---|
| <i>Données de la base métier</i> | <i>Intègre</i> | <ul style="list-style-type: none"> — 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 |
| <i>Données archivées</i> | <i>Détecté</i> | <i>Génération de checksums SHA-256 des backups</i> |

| Classe de données | Niveau exigé | Mesures |
|--|--------------------|---|
| <i>Données calculées D1</i> | <i>Maîtrisé</i> | <i>Stockage d'un checksum SHA1, relance du calcul automatiquement par batch dans les 24H.</i> |
| <i>Silo NoSQL des données Big Data avant consolidation</i> | <i>Non intègre</i> | <i>Pas de mesure particulière, pas de backup</i> |
| <i>Sources</i> | <i>Intègre</i> | <i>Utilisation du SCM Git</i> |
| <i>Avis d'imposition PDF</i> | <i>Intègre</i> | <i>Signature numérique du montant net + date + nom au format PKCS#7 (RSA, SHA256) avec horodatage. La signature résultante sera intégrée a posteriori au format hexadécimal en pied de page du PDF.</i> |

5.5.2 Confidentialité

Dispositifs répondant aux Exigences de confidentialité :

TABLE 5.11: Mesures pour assurer le niveau de confidentialité demandé

| Classe de données | Niveau exigé | Mesures |
|---|----------------|---|
| <i>Contenu éditorial</i> | <i>Public</i> | <i>Échanges en HTTPS, pas d'authentification</i> |
| <i>Profil du compte du site Web</i> | <i>Limité</i> | <i>L'accès à ce contenu nécessite une authentification réussie par login/mot de passe</i> |
| <i>Historique du compte</i> | <i>Réservé</i> | <i>L'accès à ce contenu est réservé aux exploitants habilités, uniquement via des requêtes PL/SQL de la base de données</i> |
| <i>Logs des activités de l'internaute</i> | <i>Réservé</i> | <i>L'accès aux fichiers de log est réservé aux exploitants habilités (accès SSH à la machine M et mot de passe Unix)</i> |

| Classe de données | Niveau exigé | Mesures |
|---|--------------|--|
| <i>Données RH aides sociales aux employés</i> | <i>Privé</i> | <i>Ces données sont chiffrées en AES 256 sous forme d'un BLOB en base, remontées au client Web via le service REST Y puis déchiffrées au sein du navigateur dans l'application Angular (bibliothèque forge.js) via un mot de passe complémentaire de l'utilisateur (non stocké côté 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 X.</i> |

Penser aussi à la confidentialité des 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...)

5.5.3 Identification

Dispositifs répondant aux exigences d'identification :

*Exemple 1 : L'Id des usagers de l'application sera l'attribut uid des DN **cn=XXX,ou=service1,dc=entreprise** 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 ajoutée dans l'application et requêtée avant toute création de nouveau compte.

5.5.4 Authentification

Dispositifs répondant aux exigences d'authentification :

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.

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

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.

5.5.4.1 Comptes de service

Les comptes de service sont utilisés pour l'authentification à un composant technique depuis un batch ou une API.

TABLE 5.12: Comptes de service

| Compte | Ressource requérant authentification | mode de stockage des credentials |
|--|--------------------------------------|--|
| Comptes JDBC (un compte par base de données) | Instances PG et SqlServer. | Stockage en clair dans la configuration des datasources. Valorisé à partir des pilars Salt des API. |

5.5.5 Fédération d'identité

Dispositifs répondant aux exigences de fédération d'identité :

Les solutions les plus courantes sont actuellement : OpenId Connect (OIDC), SAML ou Oauth 2.0 (pseudo-authentification seulement pour cette dernière).

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 (basé sur OIDC) 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 : <faire un schéma>

5.5.6 SSO, SLO

Dispositifs répondant aux «Exigences de SSO et SLO» :

Détailler la technologie choisie et son intégration. Quelques solutions courantes : CAS, OpenAM, LemonLDAP::NG. Pour les applications Web, préciser les contraintes navigateur (activation des cookies en particulier).

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

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

5.5.7 Non-répudiation

Dispositifs répondant aux Exigences de non répudiation :

Exemple : La déclaration d'impôt sera signée par le certificat client de l'utilisateur (certificat X509, RSA, SHA-256) qui lui a été fourni par le composant X.

5.5.8 Anonymat et vie privée

Dispositifs répondant aux exigences d'anonymat et de respect de la vie privée :

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 publique seront exportées 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 oeuvre sur toutes les pages de l'application Angular via le module `angular-cookie-law`.

5.5.9 Habilitations

Dispositifs répondant aux Exigences sur les habilitations :

Exemple 1 : la gestion des autorisations sera gérée applicativement et stockée dans la base applicative PostgreSQL. Ces tables seront décrites dans le dossier de spécification.

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

5.5.10 Tracabilité, auditabilité

Dispositifs répondant aux exigences de tracabilité et d'auditabilité :

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.

Exemple 2 : l'outil d'IDS hybride (réseau + host) OSSEC sera installé sur l'ensemble des machines utilisées par l'application.

Exemple 3 : Les tables X, Y, .. seront historisées suivant le principe suivant : ... <diagramme de classe>

Exemple 4 : tous les documents servant de preuve seront archivés dans la GED.

Exemple 5 : Les logs contenant le tag [PREUVE] et issu de l'ensemble des composants seront centralisés via le système de centralisation de log Elastic Search puis insérés avec traitement Logstash de façon journalière vers l'index Elastic `preuves`.

5.6 Auto-contrôles

5.6.1 Auto-contrôle des vulnérabilités

La gestion des vulnérabilités dépasse largement le cadre de ce document mais il est bon de s'auto-contrôler 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.

Bien entendu, il existe de nombreux autres points de contrôle dépendants du contexte de l'application

TABLE 5.13: Checklist d'auto-contrôle de prise en compte des vulnérabilités courantes

| <i>Vulnérabilité</i> | <i>Pris en compte ?</i> | <i>Mesures techniques entreprises</i> |
|--|-------------------------|---|
| <i>Accès à des ports privés</i> | <i>X</i> | <i>Configuration du pare-feu iptables sur la machine exposée à Internet. Seul les ports 80 et 443 sont ouverts. Le pare-feu sera configuré en mode stateful (avec extension conntrack)</i> |
| <i>Attaque de mot de passe par force brute</i> | <i>X</i> | <i>Utilisation de fail2ban, mise en prison de 1h au bout de 3 tentatives de connexion ssh.</i> |
| <i>Visibilité des URLs directes</i> | <i>X</i> | <i>Centralisation de tous les accès depuis Internet via un reverse proxy Apache + mod_proxy. Réécriture d'URLs pour masquer les URL internes.</i> |
| <i>Contournement du contrôle d'accès</i> | <i>X</i> | <i>Utilisation du SSO CAS, voir chapitre 3</i> |
| <i>Injection SQL</i> | <i>X</i> | <i>Utilisation de PreparedStatement uniquement, audit des requêtes SQL.</i> |
| <i>Injection NoSQL</i> | <i>X</i> | <i>Désactivation du support JS par MongoDB</i> |
| <i>Injection OS</i> | <i>X</i> | <i>Vérification qu'il n'y a aucun appel de commandes systèmes dans le code (type Runtime.exec())</i> |
| <i>Violation de gestion d'authentification et de session</i> | <i>X</i> | <i>Traité avec le dispositif anti-CSRF, voir plus bas. On logue l'IP à fin d'audit.</i> |
| <i>XSS</i> | <i>X</i> | <ul style="list-style-type: none"> — <i>Utilisation de librairie d'échappement. Pour les modules Java, nous utiliserons StringEscapeUtils.escapeHtml4() de commons-lang</i> — <i>Utilisation des headers HTTP : X-Frame-Options SAMEORIGIN, Content-Security-Policy</i> — <i>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 contournant l'anti-XSS</i> |

TABLE 5.13: Checklist d'auto-contrôle de prise en compte des vulnérabilités courantes

| | | |
|---|----------|---|
| <i>ReDOS</i> | <i>X</i> | <i>Vérification que les expressions régulières utilisées par les dispositifs anti-XSS ne sont pas éligibles à ce type d'attaque, voir https://www.owasp.org/index.php/Regular_expression_Denial_of_Service_-_ReDoS</i> |
| <i>Référence directe à un objet</i> | <i>X</i> | <i>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.</i> |
| <i>Planification des mises à jour de sécurité</i> | <i>X</i> | <ul style="list-style-type: none"> — <i>Les mises à jour Centos seront planifiées tous les premiers mercredi du mois</i> — <i>Les mises à jour Wildfly sont appliquées au plus deux semaines après leur sortie</i> |
| <i>Exposition de données sensibles</i> | <i>X</i> | <ul style="list-style-type: none"> — <i>Tous les algorithmes de sécurité sont à jour : au minimum SHA-256, AES 256</i> — <i>Le SSL V2 et V3 est désactivé coté Apache suite à la faille DROWN (<code>SSLProtocol all -SSLv2 -SSLv3</code>)</i> — <i>L'application ne fonctionne qu'en HTTPS</i> — <i>Le serveur Web fixera le header HSTS avec <code>includeSubDomains</code> sur toutes les ressources</i> |
| <i>CSRF</i> | <i>X</i> | <i>Utilisation du dispositif anti-CSRF d'AngularJS (https://docs.angularjs.org/api/ng/service/\$http)</i> |
| <i>Manque de contrôle d'accès au niveau fonctionnel</i> | <i>X</i> | <ul style="list-style-type: none"> — <i>Mise en place de la politique d'autorisation décrite au chapitre 2</i> — <i>Campagne de tests fonctionnels</i> |
| <i>Log injection</i> | <i>X</i> | <ul style="list-style-type: none"> — <i>Échappement des logs avant de les transmettre à log4j</i> — <i>Vérification des outils de consultation de logs</i> |
| <i>Attaques HTTPS + compression CRIME/BREACH</i> | <i>X</i> | <ul style="list-style-type: none"> — <i>Désactivation de la compression HTTPS au niveau de l'Apache : <code>SSLCompression off</code></i> — <i>Dispositif anti-CSRF</i> |
| <i>Upload de fichiers malicieux</i> | <i>X</i> | <i>Validation des pièces jointes par l'anti-virus ClamAV</i> |

5.6.2 Auto-contrôle RGPD

Cette section aide à vérifier la prise en compte des exigences du RGPD.

A noter que le RGPD ne concerne que les personnes physiques, pas les personnes morales.

Cette liste n'est qu'un exemple partiel, faire valider votre projet par votre service sécurité et juridique.

TABLE 5.14: Checklist d'auto-contrôle de respect du RGPD

| <i>Exigence RGPD</i> | <i>Prise en compte ?</i> | <i>Mesures techniques entreprises</i> |
|---|--------------------------|--|
| <i>Registre du traitement de données personnelles</i> | <i>X</i> | <i>Liste des traitements et données personnelles dans le document XYZ</i> |
| <i>Pas de données personnelles inutiles</i> | <i>X</i> | <i>Vérifié, la rétention de numéro de CB a été supprimée car inutile.</i> |
| <i>Droits des personnes (information, accès, rectification, opposition, effacement, portabilité et limitation du traitement.)</i> | <i>X</i> | <i>Oui, traitement manuel sur demande depuis le formulaire situé à http://xyz, traitement en 1 mois max</i> |
| <i>Sécurisation des données</i> | <i>X</i> | <i>Oui, voir les mesures listées dans ce document notamment sur la confidentialité, audibilité et intégrité.</i> |