Projet XXX

Dossier d’Architecture

volet développement

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

DIFFUSION	
Destinataire	**Organisation**

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

Sommaire

1. Introduction....4
1.1. Documentation de Référence....4
1.2. Glossaire....4
2. Exigences non fonctionnelles....5
2.1. Accessibilité....5
2.2. Ergonomie....5
2.3. Internationalisation....7
2.4. Mode déconnecté....7
3. Pile logicielle....8
3.1. Filière technique retenue....8
3.2. Composants logiciels....8
4. Spécificités d'usine logicielle....9
5. Spécificités des tests....10
6. Patterns notables....11
7. Éco-conception....12

1. INTRODUCTION

Ce document fourni le point de vue logiciel du projet. Se référer aux volets joints pour les points de vue applicatifs, infrastructure et sécurité. Un référentiel joint liste les points à statuer et les hypothèses prises.

1.1. 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 impossible à maintenir.

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

1.2. GLOSSAIRE

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

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

2. EXIGENCES NON FONCTIONNELLES

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

2.1. 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) à https://references.modernisation.gouv.fr/rgaa-accessibilite/#menu qui vise un niveau WCAG 2.0 AA :

Les niveaux d'accessibilité WCAG2.0

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

Source : Wikipedia

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

- *atteindre un niveau d'accessibilité très élevé est en général coûteux, contraignant technologiquement et limite les possibilités ergonomiques. 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 ».*

2.2. ERGONOMIE

2.2.1 Site Web adaptatif

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

- *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).*

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

2.2.2 Charte ergonomique

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

2.2.3 Spécificités sur les widgets

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

2.2.4 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[1] (mode déconnecté, rapide, adaptatif, accessible depuis l'OS, ...)

Exemple : L'application XXX sera totalement PWA. Des tests devront démonter que le site continuer à fonctionner sans réseau et que les pages se chargent en moins de 5 secs en 3G.

2.2.5 Navigateurs supportés

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

Attention : supporter d'anciens navigateur (IE en particulier) peut engendrer des surcoûts rédhibitoires (sauf à utiliser une librairie qui masque cette complexité et en espérant qu'elle fonctionne correctement).

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

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

1 Voir https://developers.google.com/web/progressive-web-apps/checklist

Exemple 2 : L'application YYY 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 ZZZ 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.

2.3. INTERNATIONALISATION

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, affichage des séparateurs décimaux, etc.

Exemple 1 : L'IHM XXX 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. 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 synchronisées une fois sur site. Il peut également s'agir d'applications PWA (voir plus haut) utilisant les service workers et les local storage HTML5.

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.

3. PILE LOGICIELLE

3.1. 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 : comme validé en comité d'architecture du 12/01/2016,, nous utilisons ReactsJS à titre expérimental au sein de l'organisation XXX.

3.2. 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 :
Composant ihm-monappli :

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

4. SPÉCIFICITÉS D'USINE LOGICIELLE

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

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

5. SPÉCIFICITÉS DES TESTS

Une méthodologie ou une technologie particulière est-elle en jeu dans ce projet ?

Exemple 1 : ce projet sera couvert en plus des TU et tests d'intégration car des tests d'acceptance BDD (Behavioral Driven Development) en technologie JBehave + Serenity.

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

6. PATTERNS NOTABLES

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

7. 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 PowerAPI[2] (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 (select) aux pages HTML retournées en passant par les données coté serveur ;*
- *grouper les traitements de masse dans des batchs 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 *http://www.powerapi.org/*