Norme de développement

OCDM Base Stack

Version 0.1

# Historique du document

## Révisions du document

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version | Date | Version applicative concernée | Description | Auteur |
| 0.1 | 06/04/2023 |  | - Version initiale | TGE |
|  |  |  |  |  |

## Validation du document

|  |  |  |
| --- | --- | --- |
| Validation du commanditaire | <Organisation>  <Nom / Prénom>  <Fonction> | Date et visa : |
| Validation du prestataire | <Organisation>  <Nom / Prénom>  <Fonction> | Date et visa : |
| Approbateurs : | <Nom / Prénom>, <Organisation>  <Nom / Prénom>, <Organisation> |  |

Table des matières

[Historique du document 2](#_Toc131692921)

[Révisions du document 2](#_Toc131692922)

[Validation du document 2](#_Toc131692923)

[1. Contexte 4](#_Toc131692924)

[1.1. Objet du document 4](#_Toc131692925)

[1.2. Enjeux projet 4](#_Toc131692926)

[1.3. Utilisateurs 4](#_Toc131692927)

[2. Normes de développements Java 5](#_Toc131692928)

[3. Normes de développement JavaScript 6](#_Toc131692929)

# Contexte

## Objet du document

Le présent document décrit les normes de développement en vigueur pour les projets basés sur Base Stack et plus généralement la cible pour l’ensemble des projets.

## Enjeux projet

Ce projet a pour objectif de fournir une base de démarrage uniforme et prête à l’emploi pour l’implémentation des projets.

L’objectif du déploiement de cette base commune pour nos futurs projets est multiple

* Gains de temps au démarrage des projets
* Industrialisation des projets facilitée (build, packaging, CI…)
* Amélioration du support de la DT et des autres projets
* Facilité pour intervenir sur différents projets
* Amélioration de la capacité à mettre à jour le socle

Ce socle projet n’implémente pas de logique métier, il propose un socle technique fonctionnel répondant aux problématiques transverses commune des projets

## Utilisateurs

Les utilisateurs de ce logiciel sont les équipes projet qui implémentent cette base pour la décliner comme un projet à destination des utilisateurs finaux.

# Normes de développements Java

## Structure des projets

* Maven
  + Structure modules : La racine d'un repo est un projet Maven
    - Il a un POM
    - Il référence les sous modules (et ainsi de suite a tous les niveaux)
      * Même s’il ne les build pas (pour l'import dans les outils de dev)
      * Pas de module "orphelin"
  + Si une conf particulière est requise dans le settings.xml pour builder, un commentaire qui renvoie vers la doc (le readme) est souhaitable
  + Header
  + Formatage
  + Extraction No de version
  + Numérotation / SNAPSHOT
  + Déclarer explicitement les dépendances utilisées directement, ne pas compter sur les dépendances transitives

## Convention de présentation

* README systématique à la racine
  + Instructions de build / packaging / exécution
  + Renvoi vers la doc (wiki / GED…)
  + README à la racine des sous-modules si utile
* Headers : Copyright / licence

### Style du code

* Nommage
  + Nom explicite, en fonction du rôle, pas du type
  + Toujours dans la même langue
  + Casse
    - CamelCase
    - Les classes commencent par une MAJ
    - Les variable/méthode par une minuscule (même les static)
    - Les constantes en majuscules avec des "\_" pour séparer les mots
* Nommage des opérations REST
  + Explicite, simple, hiérarchie cohérente
* Formatage
  + Indentation style K&R
  + Indentation avec des espaces, pas des tabulations
  + Pas d'espaces en fin de ligne
  + Utiliser les règles de formatage fournies via XML (a créer)
  + Accolades systématiques, même pour un bloc d'une seule ligne
* Fichiers encodés en UTF8 (sauf fichiers .properties en ISO-8859-1)
* Commentaires utiles (dans le code et dans la Javadoc)
  + Pattern utilisé
  + Exceptions attendues
  + Conditions de sortie anticipée
  + Algorithme complexe…

## Bonne pratiques et optimisations

* Écrire des test unitaires (2e vague atelier)
  + Utiles
  + Indépendant de l'environnement
  + Qui initialisent les données de test
* Mettre des logs (2e vague atelier)
  + Utiliser un logeur (slf4j)
  + Utiliser les bons niveaux de log
* Éviter les imports avec wildcard (chaque classe importée explicitement : perf, lisibilité, maintenabilité)
* Pas d'utilisation systématique des interfaces
  + Uniquement si plusieurs implémentations sont prévues
* Pas d'utilisation systématique des DTO
* Les contrôleurs REST peuvent utiliser les repositories dans passer par les services
* Utiliser le niveau de syntaxe le plus élevé par rapport à la version de Java utilisée
  + 7 : multi catch, gestion des ressources automatique
  + 8 : lambda, annotation de type
  + 9 : méthodes privées dans les interfaces
  + 10 : inférence de type pour les variables locales
  + 11 : inférence de type dans les arguments des expressions lambda, nouvelle API String
  + 14 : expressions switch
  + 15 : blocs de texte
  + 16 : classes record, pattern matching
  + …
* Utiliser des listes à la place des tableaux quand c'est possible
* Utiliser des Stream à la place des boucles avec test quand c'est possible
* Gestion de la sécurité…

## Gestion des sources

* Utiliser les pull/merge request, faire de la review croisée

## Formateur

// TODO formateur format XML eclipse

## Linter

Installer le plugin SonarLint sur son IDE et suivre les recommandations.

// TODO SonarLint : revue règles par défaut et customisation si necessaire

# Normes de développement JavaScript

Les norme de développement JavaScript recommandées sont celles décrites dans le guide Javascript Airbbnb.

Le guide est disponible ici : <https://github.com/airbnb/javascript>

|  |
| --- |
| *⚠️ La traduction française du guide est obsolète, elle recommande notamment, à tort, l’utilisation de ‘var’ au lieu de ‘let’ et ‘const’.* |

La lecture du guide est recommandée, il contient de nombreux exemples.

Un linter reprenant les règles du guide est intégré dans le projet ‘Base-Stack’, ce linter corrige automatiquement le formatage et affiche les erreurs de code dans la console lors du build et lorsque le linter est lancé manuellement avec « npm run lint ».