ALDES

Audit Qualité
Direction Technique

Date: 21/12/2020
Version: 1.4
Author: Adrien LUCAS & Gaëtan MONTAGNAC

Cycle de vie du document

Validation du document

Nom	Role	Organisme
Adrien LUCAS	Auditeur	Smile
Gaëtan MONTAGNAC	Auditeur	Smile

Historique du document

Nom	Version	Commentaire	Date
Adrien LUCAS	1.0	Création	07/12/2020
Gaëtan MONTAGNAC	1.1	Ajouts	07/12/2020
Gaëtan MONTAGNAC	1.3	Ajouts	18/12/2020
Adrien LUCAS	1.4	Ajouts	21/12/2020

Documents liées

Version	Nom du Document	Reference

Mission

La mission a été composée d'une réunion de présentation fonctionnelle et technique de l'application, d'une étude détaillée de l'application et de son code source, de la rédaction, et restitution de ce présent document.

Objectifs

Afin de répondre à des enjeux métiers forts, ALDES a lancé en 2020 un projet d'usine à sites reposant sur la technologie eZ Platform.

Dans ce cadre Smile réalise un audit de qualité qui va permettre d'évaluer si les règles de bases permettant la pérennité d'un projet sont respectée.

Intervenants Smile

- Adrien LUCAS - Auditeur, Expert Symfony, Direction Technique - Smile
- Gaëtan MONTAGNAC - Auditeur, Expert eZPlatform - Smile

Qualité de code

Code mort

Présence de code auto-généré, à supprimer :

- `\AppBundle\Repository\Doctrine\ProductRepository`
- `\AppBundle\Repository\Doctrine\CaracRepository`
- `\AppBundle\Repository\Doctrine\PimCaracValueRepository`
- `\AppBundle\Repository\Doctrine\ParameterRepository`

Présence de code commenté :

- `\AppBundle\QueryType\ContentChildrenQueryType` lignes 59-63
- `\AppBundle\Listener\PageBuilderBlockListener` lignes 78-84
- `app/Resources/views/themes/front/full/product.html.twig` lignes 7-14

Filtres et fonction twig déclarés mais jamais utilisés (dans `\AppBundle\Services\Twig\AldesExtension`) :

- le filtre `breadcrumb`
- le filtre `relative_path`
- la fonction `relative_path` (à noter que cette fonction est déjà implémenté dans le coeur Symfony)

> Le code est dit "mort" lorsqu'il est présent dans la base de code mais n'est plus exécuté ou n'est plus utile. La présence d'une grande quantité de code mort rend difficile toute opération de maintenance ou de refactoring à grand echelle.

Comparaisons non-strictes

Beaucoup de comparaisons au sein de la base de code ne sont pas strictes (i.e `==`). Par exemple :

```
// src/AppBundle/Services/PimHelper.php:120
$code = $response->getStatusCode();
$result = $response->getBody();


if ($code == '200') { // $code est un entier
        // ...
}
```

à remplacer par

```
// src/AppBundle/Services/PimHelper.php:120
$code = $response->getStatusCode();
$result = $response->getBody();


if ($code === 200) {
        // ...
}
```

> Il existe au moins 25 comparaisons non-strictes au sein de la base de code. En PHP, il est préférable de faire usage de comparaisons strictes pour éviter les effets de bords non maîtrisés liés aux mécanismes de transtypage.

Visibilité Orientée objets

Il manque la visibilité sur les constantes de classes. Il s'agit d'indiquer si ces valeurs sont utiles dans le cadre restreint de la classe où elles sont déclarée (`private` ou `protected`) ou si au contraire, elles sont utilisées par d'autre entités

du système (`public`). Par exemple, la constante `\AppBundle\Entity\Doctrine\Product::CARAC_TYPE_VALUE_MAPPER` doit être déclarée privée.

> Depuis la version 7.1 de PHP, les constantes de classes peuvent avoir une visibilité.

Certaines méthodes sont déclarées avec la mauvaise visibilité. Par exemple, `\AppBundle\Command\PimImportArboPublicationCommand::syncArboPimToTags` est déclarée publique alors qu'elle devrait être privée.

> Il faut absolument indiquer l'information de visibilité sur l'ensemble des constantes, des méthodes et des propriétés de classes pour faciliter la maintenance de l'application.

Absence du type de retour

Certaines méthodes ne déclarent pas leurs type de retour :

- `\AppBundle\Entity\Doctrine\Parameter::getId`
- `\AppBundle\Entity\Doctrine\Parameter::getName`
- `\AppBundle\Entity\Doctrine\Parameter::getValue`
- `\AppBundle\Entity\Doctrine\Parameter::getParams`
- `\AppBundle\Command\PimImportCaracteristiquesCommand::execute`
- `\AppBundle\Listener\EmbedListener::getSubscribedEvents`
- `\AppBundle\Listener\TestimonialListener::getSubscribedEvents`
- `\AppBundle\Listener\IllustratedBenefitsListener::getSubscribedEvents`
- `\AppBundle\Listener\ComposantsSolutionsListener::getSubscribedEvents`
- `\AppBundle\Listener\IllustratedBenefitsOldListener::getSubscribedEvents`
- `\AppBundle\Listener\CenteredImageListener::getSubscribedEvents`
- `\AppBundle\Listener\KeyFactsListener::getSubscribedEvents`
- `\AppBundle\Listener\InfographicListener::getSubscribedEvents`
- `\AppBundle\Listener\MultiContentListener::getSubscribedEvents`
- `\AppBundle\Listener\PushNewsListener::getSubscribedEvents`
- `\AppBundle\Listener\BlockBannerListener::getSubscribedEvents`
- `\AppBundle\Listener\FaqListener::getSubscribedEvents`
- `\AppBundle\Listener\PageBuilderBlockListener::getSubscribedEvents`
- `\AppBundle\Listener\MultiplePushListener::getSubscribedEvents`
- `\AppBundle\Services\UrlAliasRouter::matchRequest`
- `\AppBundle\Services\UrlAliasRouter::generate`
- `\AppBundle\Services\UrlAliasRouter::getUrlAlias`
- `\AppBundle\Services\PimHelper::getContextID`
- `\AppBundle\Services\PimHelper::getLangID`
- `\AppBundle\Services\PimHelper::getRootContext`
- `\AppBundle\Services\TagHelper::getChildTags`
- `\AppBundle\Services\AdminHelper::setAdminUser`
- `\AppBundle\Services\BreadcrumbHelper::generate`
- `\AppBundle\Services\Twig\AldesExtension::getFunctions`
- `\AppBundle\Services\Twig\AldesExtension::getFilters`
- `\AppBundle\Services\PageBuilder\Attribute\TagsListFormType::getParent`
- `\AppBundle\QueryType\ContentChildrenQueryType::getName`
- `\AppBundle\Controller\BaseController::disabledViewAction`
- `\AppBundle\Controller\SearchController::getForm`

- `\AppBundle\Controller\SearchController::getSearchResults`

> Depuis PHP 7.0 il est possible de typer les retours de fonctions et de méthodes. Ce mécanisme doit être mis en œuvre de manière exhaustive sur la base de code pour améliorer la robustesse et la stabilité de l'application.

Absence du type de paramètre

Certaines méthodes ne déclares pas le type de certains de leurs paramètres :

- `\AppBundle\Services\Twig\AldesExtension::getDoctrineProduct(idproduct)`
- `\AppBundle\Services\ImportsHelper::setFieldPim(valueObject)`
- `\AppBundle\Services\ImportsHelper::setFieldPim(contentStruct)`
- `\AppBundle\Services\MediaHelper::recupCumulus(cumulus)`
- `\AppBundle\Services\MediaHelper::convert(path)`
- `\AppBundle\Command\PimImportArboPublicationCommand::removeUselessEzProduct(selecteur)`
- `\AppBundle\Command\PimImportArboPublicationCommand::removeUselessEzProduct(parentlevelExtId)`
- `\AppBundle\Command\PimImportArboPublicationCommand::removeUselessEzProduct(currLevelArbo)`
- `\AppBundle\Services\TagHelper::createTag(parentExtID)`
- `\AppBundle\Services\TagHelper::createTag(parentLabel)`
- `\AppBundle\Services\TagHelper::createTag(tagExtId)`
- `\AppBundle\Services\TagHelper::createTag(label)`

> Depuis PHP 7.0 il est possible de typer les paramètres scalaires de fonctions et de méthodes. Ce mécanisme doit être mis en œuvre de manière exhaustive sur la base de code pour améliorer la robustesse et la stabilité de l'application.

> Pour profiter au maximum des bénéfices du typage PHP, il faut activer la directive `strict_types` sur l'ensemble des scripts de l'application.

Duplication de code

Comme le code "mort", le code dupliqué est un frein à la maintenance de l'application. Sa présence peut aussi mener à des régressions lors de la réalisation. Dans la base de code de l'application, il existe notamment une portion de code de 10 lignes qui est dupliquée 3 fois :

- `\AppBundle\Listener\IllustratedBenefitsOldListener` lignes 43-53
- `\AppBundle\Listener\CenteredImageListener` lignes 43-53
- `\AppBundle\Listener\EmbedListener` lignes 38-48

> Pour éviter que des duplications de code soient introduites par erreur, il peut être utile d'ajouter l'outil `phpcpd` dans la procédure d'intégration continue.

Dépendances applicative

Les dépendances de l'application sont déclarées dans le fichier `composer.json`. Plusieurs anomalies sont relevés ici.

Dépendances abandonnées :

- `white-october/pagerfanta-bundle`, il faut utiliser son remplaçant `babdev/pagerfanta-bundle`.
- `sensio/generator-bundle`, à remplacer par `symfony/maker-bundle`

Dépendance inutilisée :

- `gregwar/captcha-bundle`
- `overblog/graphql-bundle`
- `overblog/graphiql-bundle`

Des dépendances sont déclarée en `require` alors qu'il s'agit de dépendances utiles seulement pour la phase de développement et qui devraient donc être déclarée en `require-dev` :

- `phpmd/phpmd`
- `sensiolabs/security-checker`
- `squizlabs/php_codesniffer`

> La déclaration de dépendances sur lesquels l'application ne dépend pas vraiment ou pas directement peut entraîner des impossibilité de mise à jour et peut aussi des difficultés d'analyse d'anomalies.

Surcharge "brutale" du code d'une dépendance

La procédure d'installation de l'application applique un patch sur le dossier `vendor` qui correspond aux dépendances de l'application. Il n'existe pour l'instant qu'un seul patch dans le dossier `patches` et l'installation se fait à la première ligne du "script" composer `symfony-scripts` :

```
        // ...
        "scripts": {
            "symfony-scripts": [
                "cp -R patches/* vendor/",
        // ...
```

> L'utilisation de tels patchs est une très mauvaise pratique qui peut potentiellement empêcher la mise à jour automatique des dites dépendances et donc à terme entraîner des instabilités, voire des failles de sécurité.

Absence de tests automatisés

Les tests automatisés, qu'ils soient unitaires ou fonctionnels, constituent un outil de qualité très important aussi bien lors de la phase de build que de la phase de run. Ils permettent d'assurer la pérennité exhaustive des fonctionnalités et de détecter les régressions avant la mise en production; ils permettent en outre de lancer des chantiers de refactoring ambitieux de manière maîtrisée budgétairement et fonctionnellement.

> L'absence de tests sur cette application risque d'engendrer des coûts élevés de maintenance, voire l'apparition de régressions au cours de la réalisation.

Mauvais usage des outils d'analyse.

L'outil `phpmd` semble être utilisé dans la chaîne d'intégration continue actuelle, c'est une bonne pratique. Malheureusement, afin de "cacher" certain alertes remontées par l'outil, le code comporte des portions inutiles tel que :

```
// src/AppBundle/Services/PageBuilder/Attribute/FormTypeMapper/TagsListFormTypeMapper.php:49
// ...
        // Useless, only to pass phpmd \o/
        $attributes = $blockDefinition->getAttributes();
        $attributes = reset($attributes);
// ...
```

La plupart des outils d'analyse statique propose de passer sous silence certaines alertes de manière explicite et ponctuelle, souvent via des annotations. Il faut absolument utiliser ces mécanismes au lieu d'ajouter du code inutile dans le seul but de "cacher" les alertes.

Mauvais design

Une surcharge du routeur `\eZ\Publish\Core\MVC\Symfony\Routing\UrlAliasRouter` est réalisée afin de gérer les contenus multi-positionnés. Cette surcharge dans `\AppBundle\Services\UrlAliasRouter` fait usage de l'héritage plutôt que d'opter pour la composition et semble avoir été la cause de plusieurs bugs corrigés depuis.

L'usage du pattern decorator pour atteindre le même objectif avec moins de duplication de code et moins de risques d'instabilités. Exemple :

```
services:
        \AppBundle\Services\UrlAliasRouter:
            decorates: ezpublish.urlalias_router
```

A noter aussi qu'une seconde méthode est surchargée, supposément pour corriger un bug de la fonctionnalité de recherche, cependant cette implémentation est identique à l'implémentation d'origine. Garder cette surcharge, à priori inutile, est un risque pour la stabilité lors d'éventuelle mise à jour.

Respect des bonnes pratiques eZ Platform

Performances

Une courte analyse de performance à été réalisée. Il en ressort que la génération des pages est ralentie par la génération du menu principal. En effet, lors de notre analyse sur une instance locale de l'application, nous avons constaté que le chargement prend environ 1.5 secondes en désactivant le header, et passe à plus de 4.5 secondes si on le réactive. Cette lenteur semble due à deux raisons principales :

- L'utilisation des fonctions `fetch_content` et `fetch_children` du module de `sqli/eztoolbox` dans le template `ezplatform/app/Resources/views/themes/front/header/header_menu_element/category.html.twig`, qui récupèrent des données distantes et régènére les éléments du menu (en provenance du PIM) sans mise en cache.

> Pour éviter de re-accéder plusieurs fois à une ressource distante si celle-ci n'a pas changée, on pourra mettre en place deux types de cache. Soit un cache applicatif, côté site internet, pour stocker les réponses du PIM pendant un temps déterminé (quelques minutes par exemple). Soit un cache HTTP côté PIM, pour avoir des temps de réponses plus rapides.

- L'utilisation du couple de fonctions `render(controller (...))` dans le template `ezplatform/app/Resources/views/themes/front/header/header_menu_element/header_product_elements.html.twig` qui est assez coûteuse en ressources et temps de chargement, d'antan plus dans le cas présent où cette structure est utilisé à l'intérieur de boucles `for`.

> Le mécanisme de sous-requête de Symfony est très puissant mais aussi assez consommateur, on préfèrera construire les éléments d'une page au sein de la requête principale (`master`) pour optimiser le rendu.

Configuration des siteaccess

Les fichiers de configurations d'eZ Platform sont conformes aux bonnes pratiques concernant la configuration de plusieurs sites sur la même instance eZ Platform.

- Configuration des différents `siteaccess` lié aux différents sites (en cours et à venir)

```
siteaccess:
        [...]
            group_front:
                - fr
                - en
                - es
```

- Mapping des différents `siteaccess` vers leur `RootNodeId` correspondants

```
    aldes.en.location_id: 56
    aldes.fr.location_id: 102
    aldes.es.location_id: 105
```

- Configuration du `siteaccess` par défaut ES

```
siteaccess:
        [...]
        default_siteaccess: es
```

De plus, il n'y a pas de présence dans le code applicatif d'appel à ur`siteaccess` particulier, qui nécessiterait un rework pour la mise en place d'autres sites.

> Configuration unique pour le `siteaccess` admin donnant accès au Back-Office Ezplatform, qui aurait pu être scindé en différents `siteaccess` (un par site) en fonction des besoins métiers.

Injection du conteneur de services

Injecter le conteneur de service dans un objet est une violation franche du principe même d'inversion de dépendance (*D* de *SOLID*).

- `\AppBundle\Services\AdminHelper`
- `\AppBundle\Command\PimImportArboPublicationCommand`
- `\AppBundle\Command\PimImportCaracteristiquesCommand`

> Le principe d'inversion de dépendance rend le code de l'application plus robuste et maintenable dans le temps.

Affichage sans templating

Certains éléments de l'affichage sont directement inscrit dans le code métier, sans passer par le moteur de templating : `\AppBundle\Listener\PageBuilderBlockListener::buildTweet`

Duplication de template / Template inutilisé

Plusieurs templates sont dupliqués pour des types de contenus différents :

- News Page
- Solution
- Editorial Page

Chacun de ses types de contenus possède son propre template mais les trois templates sont identiques.

De plus, un template pour un type de contenu `Article` est présent dans le code mais n'est pas utilisé.

> Ces fichiers inutiles ne présente pas de danger pour le fonctionnement de l'application, mais il est conseillé d'optimiser au maximum le code afin de respecter les bonnes pratiques et dans un soucis de performances

Appels contrôleur inutile

Pour des besoins d'affichage d'informations spécifiques, l'appel à un contrôleur peut être réalisé en amont de l'affichage du template d'un type de contenu. Cet appel influe sur le temps d'affichage de la page, et certains types de contenu possèdent un appel de ce type sans avoir le besoin applicatif de le faire.

> Dans l'optique d'évolutions futures, il est conseillé de maximiser les performances et le temps d'affichage des pages, notamment en se passant d'appels inutiles.

Développement natif

Création d'une fonction permettant de modifier l'utilisateur courant afin d'exécuter des scripts de niveau admin, alors que le natif eZ Platform fournit 2 méthodes pour le faire de base.

- `\AppBundle\Services\AdminHelper::setAdmin()`

Développement inutile car fourni par l'API Ezplatform.

Montée de version vers Ibexa 3.2

Avantages montée de version

Les avantages d'une montée de version vers Ibexa 3.2 (nouveau nom de eZ Platform) sont :

- Un socle à jour et pérenne dans le temps
    - La version 2.5 d'eZ Platform se base sur un socle symfony en version 3.4, dont la fin de maintenance est prévue pour Novembre 2021.
    - La version 3.2 d'Ibexa en revanche se base sur le socle symfony en version 5.2, qui est toujours en cours d'évolution mais elle va suivre les mises à jours Symfony pour arriver sur une version qui sera maintenue plus longtemps dans le temps.
- Nouveaux modules :
    - ElasticSearch
    - Site Factory
    - Fonctionnalités supplémentaires de contribution pour les utilisateurs Back-Office

Site Factory

Une nouvelle fonctionnalité qui permet de laisser aux utilisateurs Back-Office la possibilité de créer des "sites" directement depuis l'interface.

> Cela nécessite tout de même la création de thèmes et de squelettes de site qui seront proposés par la suite au contributeur BO lors de la création d'un nouveau site depuis le BO. Voir si cette fonctionnalité est utile et compatible avec les besoins métiers.

Passage de SolR à Elastic

Ibexa 3.2 introduit une compatibilité native avec le moteur de recherche Elasticsearch. Cet outil, bien que basé sur le même socle technique que Solr, est plus moderne que son concurrent et propose plus de fonctionnalités. Le support étant natif dans la solution, une migration vers Elasticsearch est conseillée.

Choix de solution

La version Ibexa 3.2 propose 3 solutions différentes qui sont :

- Ibexa Content : L'offre de base qui fournit les fonctionnalités essentielles pour un CMS.
- Ibexa Experience : Fournit des fonctionnalités supplémentaires pour l'expérience utilisateur en Back Office, avec un Page Builder, plus intuitif pour la création de page, de formulaires. Elle fournit également un module de planification de publication des contenus ainsi que le Site Factory.
- Ibexa Commerce : Offre pour la mise en place de site e-commerce.

> La solution Ibexa Experience offrant plus de possibilités Back-Office aux contributeurs, elle semble la plus adaptée.

Evolution du socle applicatif

Cette montée de version nécessite une mise à jour du socle Symfony de la version 3.4 à 5.2, ce qui nécessite une reprise et modification du code actuel pour faire évoluer le code applicatif ainsi que l'arborescence applicative.

Cette mise à jour va entraîner plusieurs incompatibilités au niveau de la base de code actuelle :

- Suppression du bundle `AppBundle`, remplacer par le namespace `App` à la racine du dossier `src`
- Renommage du dossier `web` en `public`
- Déplacement des assets, des templates et de traductions; respectivement de `app/Resources/assets` vers

`assets`, de `app/Resources/views` vers `templates` et de `app/Resources/translations` vers `translations`.

- Changement des notations d'héritage dans les templates
- Petits ajustements de la configuration eZ Platform
- Petites retro-incompatibilités d'interfaces PHP à corriger
- Mise à jours de certaines dépendances tierces

Plusieurs étapes sont donc nécessaires pour cette montée de version :

- Mise à jour des sources applicatives
- Mise à jour du code applicatif (un module de mise à jour est fourni)
- Mise à jour des templates
- Mise à jour de la configuration eZ Platform
- Mise à jour de la configuration serveur
- Mise à jour de la base de données via les scripts fourni

Conclusion

Compte tenu de l'architecture et des pratiques présentes dans la base de code, l'application semble capable de se comporter comme une usine à site à proprement parler.

Cependant, le présent audit met en lumière un certain nombre de manquements à la qualité et aux bonnes pratiques Symfony et eZ Platform ainsi que plusieurs problèmes de performance sur la génération des pages.

On relèvera aussi que le nombre de petites anomalies dans la base de code est élevé relativement au temps total de réalisation et aux nombres de fonctionnalités codées.

En outre, le choix de la version 3 d'Ibexa semble plus pertinent dans le contexte de cette application.

Au regard de ces conclusions, Smile s'engage à vous remettre une proposition technique et financière pour reprendre en maintenance l'application.

Annexe 1 : Macro Chiffrage

TOTAL : 22,5J

Reprise de l'existant :

- Reprise du code en cours : 7J
  - Nettoyage/Suppression du code :
  - Script d'import des produits
  - Prise en main de l'interaction avec le PIM
- Faire évoluer la procédure d'installation du projet : 1J

Soit 8J.

Migration eZ Platform

- MAJ version : 14,5 J
  - Mise à jour des sources applicatives : 2J
  - Mise à jour du code applicatif (un module de mise à jour est fourni)
    - Controller : 1,5J
    - Services : 2,5J
    - Command : 1J
    - Listener : 2J
    - Templates : 2J
    - Configuration eZ Platform : 1J
  - Mise à jour de la configuration serveur : 1,5J
  - Mise à jour de la base de données via les scripts fournis : 1J

Annexe 2 : notes d'analyse fonctionnelle du code

Controllers

BaseController :

initView() :

- Sert à initialiser le type view dans les méthodes suivantes.

validQuery() :

- Sert à valider une query dans la méthode "initView" et "viewCategoryAction".

viewPageAction() :

- Appelé lors de l'affichage d'un contenu de type ['homepage', 'product, 'article', 'editorial_page', 'news_page', 'solution']
- Sert à initialiser différents paramètres pour la view
- Pas nécessairement utile dans certains template où les paramètres ne sont pas utilisés.

viewPagedPageAction() :

- Appelé lors de l'affichage d'un contenu de type ['news_list', 'youtube_video_list']

viewCategoryAction() :

- Appelé lors de l'affichage d'un contenu de type 'category'

disabledViewAction() :

- Appelé pour tous les types de contenu dont le template d'affichage n'est pas spécifié dans le views.yml

SearchController :

searchAction() :

- Appelé pour l'affichage de la route /content/search

displayFormAction() :

- Appelé dans le template "header" pour afficher le bloc de recherche.

getForm() :

- Création du formulaire de contact (Formulaire de type Symfony)

getSearchResults() :

- Appelé pour effectuer la recherche SOLR lors de la soumission du formulaire de recherche.

Listeners

- La plupart des listeners semble être lié à la mise en place de nouveau blocks pour le pageBuilder (à approfondir)

DeleteListener :

- Sert à supprimer les entité de type Product dans doctrine lors du signal "onDelete"
- Entité lié au type de contenu Produit eZ Platform pour le stockage d'informations complexes
- Placé au mauvais endroit (2 répertoires listener, l'un dans Service, l'autre au dessus à la racine de AppBundle)

Services

BreadcrumbHelper

- Ne contient qu'une méthode pour la génération du fil d'ariane

- Appeler par l'extension twig et la fonction déclarée depuis le template lié.

AdminHelper

- Ne contient qu'une méthode (en plus du constructeur) pour modifier l'utilisateur courant
- Appeler dans une commande (PimImportArboPublicationCommand) et dans un helper (createTag) pour des fins de droits d'écriture notamment.
- https://doc.ibexa.co/en/latest/api/public_php_api/#back-office-authentication

Tag Helper

- CreateTag ne semble pas utilisé

ProductPimHelper

- Id utilisé en dur "ElementType===3" alors qu'une déclaration des types a été faite dans une constante de classe dans `PimHelper`, pourtant inutilisée.

ImportHelper

- La plupart des fonctions sont inutilisées (en cours de dev ?)
- Du code commenté dans les fonctions
- Dump($e) dans une exception catché dans setFieldPimCumulusImage()

MediaHelper

- Récupération et conversion des images depuis un proxy.

PimHelper

- Déclaration d'une constante des types non utilisée
- Déclaration des paths d'une API directement dans les fonctions, aurait pu être mis dans une constante pour une éventuelle réutilisation future.

Commands

PimImportArboPublicationCommand

- Des ids utilisés en dur "ElementType==`101` || ElementType==`102` ||ElementType==`103`" alors qu'un déclaration des types a été faite dans une constante inutilisée du `PimHelper`.

PimImportCaracteristiquesCommand

- Import des caractéristiques depuis le PIM pour import dans la base.