Ajouter_Contraintes_Models

I – DESCRIPTION DE LA DEMANDE

A/ Description

Le contrôle des informations en entrée du logiciel redpill est actuellement fait par l'intermédiaire de form validation utilisant un fichier Validation.yml aussi bien en saisie que dans les api ce qui alourdit le processus d'api. Le fichier Validation.yml contient des contraintes à tester par models. La demande consiste à ajouter des contraintes de bases qui ne sont actuellement pas ou peu controllées, ainsi que les contraintes définies dans le fichier validation.yml directement dans les model définis dans redpill.

B/ Finalités

1/ Fiabiliser les informations stockées en base de données

2/ Sécuriser l'applicatif
→ Eviter les plantages lors de l'enregistrement en base de données
→ Pouvoir tracer plus facilement des erreurs
→ Pouvoir vérifier plus facilement la non régression dans le cadre des évolutions.

a/ Sécuriser les saisies dans les vues sans avoir à redéfinir toutes les contraintes.

b/ Sécuriser les Api sans avoir à passer par des forms en récupérant les models avec annotations sous la forme d'un fichier JSON (projet pour Alain)

3/ Faciliter la lisibilité des contraintes (suppression du fichier validation.yml)

4/ Documenter le schéma de base de données

II - EXISTANT

A/ Les models non concernés par le fichier Validation.yml

1/ 89 models à traiter
(d'après /redpill/src/Eliberty/RedpillBundle/Resources/config/doctrine/*.orm.xm)
(avec quelques écarts avec /redpill/src/Eliberty/RedpillBundle/Model/*.php)

2/ Types de champs rencontrés :
(Liste d'après extraction sur les fichiers de mapping doctrine .orm.xml :
(redpill/src/Eliberty/RedpillBundle/Resources/config/doctrine/*.orm.xml
(voir démarche en annexe)

- relation
- integer
- amount → Erreur : remplacer le type par integer
- decimal
- string
- enum*
- text
- date
- datetime
- boolean
- array
- json_array
- json_array_text

3/ Utilisation de setters dans la définition des classes models
Sauf exception, les champs sont mis à jour par l'intermédiaire de setters.
- Les setters sont généralement typés pour les champs de type relations sur l'interface et pour les champs datetime (date et datetime).
- Dans quelques cas, les setters sont également typés pour les champs de types integer ou array.

4/ Cas générant des erreurs doctrine ou base de données :

- Erreurs de type :
→ relation : - génère une erreur pour toute valeur différente d'un objet de la classe correspondante (ou null).
→ integer : - génère une erreur pour toute valeur différente d'un numerique entier ou d'une string numérique entier. (erreur si partie décimale avec .)
→ decimal : - génère une erreur pour toute valeur différente d'un numérique ou d'une string numériques (avec ou sans partie décimale avec .).
→ string : - génère une erreur pour toute valeur différente d'un type simple (string, integer, boolean, …). Un objet (datetime ou autre classe) génère une erreur.
→ enum* = type string
→ text : - génère une erreur pour toute valeur différente d'un type simple (string, integer, boolean, …). Un objet (datetime ou autre classe) génère une erreur.
→ date, datetime : génère une erreur pour toute valeur différente d'un objet de type DateTime (ou null).

→ boolean : - génère une erreur pour les string autre que ‘true’, ‘false’,’t’,’f’, non numeriques.
- accepte les objets (datetime, classe autres, …) → initialise à true
- accepte les integer, float → initialise à true pour toute valeur <> 0
- accepte les expressions booleennes ex : $name == ‘keyboard’

- Erreurs de capacité de champ :
Type de champs concernés :
→ string / enum : longueur maximum définie dans le fichier .orm.xls (= 255 par défaut)
→ integer : longueur maximum (= 10 par défaut)
→ decimal : longueur maximum (= 10 par défaut),
longueur maximum partie décimale (= 0 par défaut)
valeur maximum pour numeric(18,12) = 999999.999999995 sinon erreur
valeur maximum pour numeric(10) = 9999999999.4 sinon erreur

- Erreurs de valeur non nulle :
Tous les types de champs sont concernés.
Par défaut, les valeurs null ne sont pas acceptée (le champ doit être initialisé) sauf si ‘nullable = true’ est mentionné sur le fichier .orm.xls.

- Erreurs de valeur non acceptée :
Type de champ concernés : enum*
Toute valeur non existante dans les valeurs listées dans la classe correspondante génére une erreur.

- Erreurs ci-dessus, rencontrées dans un sous-objet, défini dans le model.

- Erreurs de relation inexistante :
Type de champ concernés : relation
Une erreur est générée pour un sous-objet n’existant pas dans la table concernée.
2 cas:

→ un sous-objet initialisé avec new (donc inexistant dans la table) pour lequel la table n’a pas été définie avec un cascasde persist (ne pourra pas être créé automatiquement dans la table).

Exemple obtenu en initialisant simplement avec un new Category :

A new entity was found through the relationship 'App\Entity\Product#category' that was not configured to cascade persist operations for entity: App\Entity\Category@0000000017cb68350000000054a6081a. To solve this issue: Either explicitly call EntityManager#persist() on this unknown entity or configure cascade persist this association in the mapping for example @ManyToOne(..,cascade={"persist"}). If you cannot find out which entity causes the problem implement 'App\Entity\Category#__toString()' to get a clue.

→ un sous-objet lu en table qui a été supprimé entre-temps.

An exception occurred while executing 'INSERT INTO product (id, name, price, description, available, date1, date2, datetime1, datetime2, latitude, size, zone, category_id) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)' with params [146, true, 123, "xx", 1, null, "2018-07-11", null, "2018-07-11 07:34:53", "999999.999999995", "9999999999.4", null, 3]:

SQLSTATE[23503]: Foreign key violation: 7 ERROR: insert or update on table "product" violates foreign key constraint "fk_d34a04ad12469de2"
DETAIL: Key (category_id)=(3) is not present in table "category".

- Erreur due à une unique-constraint
Pour le groupe de champs mentionnés dans la contrainte d'unicité, on ne peut pas créer un 2e enregistrement avec les mêmes valeurs pour ce groupe de champ dans la table.

B/ Les models concernés par le fichier Validation.yml

Ces models seront étudiés plus en détail dans un 2e temps.

1/ 30 models sont concernés par le fichier validation.yml.
Le but est de rapatrier tous les controles effectués dans le fichier validation.yml sous forme d'assertions dans les models.

2/ Problèmes liés aux groupes de validation :

- bcp trop de groupes, sans intérets réels
  → Revoir la définition des groupes de validation (à étudier)

- Le fait de préciser un ou des groupes dans le service validator exclu les contraintes non rattachées aux groupes indiqués (exclusion également des contraintes non rattachées explicitement à un groupe et qui sont en fait rattachées au groupe 'Default')
  → Ajouter systématiquement le groupe 'Default' quand on définit des groupes de validation dans le validator (`$errors = $validator->validate($product,null,array('Default'));`
(→ communiquer cette contrainte à tous les développeurs)
  (→ ou Ajouter le group sur toutes les contraintes du groupe 'Default' sur le model ?
  (`@Assert\Type("string",groups={"Default","Group1"})`

  (→ voir également possibilité de l'option payload?)

III – PROPOSITION TECHNIQUE

A – Models non concernés par le fichier validation.yml

Les erreurs seront gérées en ajoutant les contraintes directement dans les classes models sous la forme d'annotations utilisées par le service validator de symfony.

1/ Contraintes utilisées pour prévenir chaque type d'erreurs:

- Erreurs de type : Assert\Type("type de champ")

- Erreurs de capacité de champ : Assert\Length(max=?) ou Assert\Range(max= ?)

- Erreurs de valeur non nulle : Assert\NotNull().

- Erreurs de valeur non acceptée : Assert\Choice(callback={"\App\Enum\?","getValues"})
Pour les champs de type enum, la valeur du champ doit exister parmi les valeurs renvoyées par la méthode getValues.

- Erreurs précédement citées pour un sous-objet défini dans un model : Assert\Valid()
Pour les champs de type relation, permet de vérifier les contraintes également sur le sous-objet.

- Erreurs de relation inexistante
- un sous-objet initialisé après un new mais non géré en 'cascade-persist' dans la déclaration doctrine générera une erreur.
- un sous-objet inexistant en table (par exemple, supprimé entre-temps) générera une erreur.
→ Gestion de ces cas non prévu pour l'instant.
(NON, la mise à jour dans la table doit mettre en œuvre le persist depuis le programme ou la mise à (jour est faite / ss-objet inexistant en table n'arrive qu'exceptionnellement (Romain).

- Erreur due à une unique-constraint : UniqueEntity
(Voir contrainte UniqueEntity)

- Autres cas particuliers :
Les cas particuliers (du genre : Email, Iban, …) devraient être traité dans le fichier validation.yml qui fait l'objet d'une 2ème partie.

2/ Contraintes à ajouter sur les models par type de champs:
(Ajouter `use Symfony\Component\Validator\Constraints as Assert;`)

- type integer
Assert\Type('integer') → accepte seulement un integer sans partie décimale (sans .)
n'accepte pas de string numérique !!
les form renvoient ils toujours des champs typés integer ???
(ou solution à chercher pour accepter aussi les string numérique ?)
Assert\Length(max=10)
Assert\NotNull() si pas de 'nullable = true'

- type decimal

Assert\Type(‘float’) → accepte uniquement les types float (pas de string numerique))
Assert\Range(max=999999.999999995) pour numeric(18,12)
Assert\Range(max=9999999999.4) pour numeric(10)
(La partie décimale ne génère pas d’erreur de capacité, elle est arrondie selon la capacité tolérée
(pour la partie décimale. La partie entière peut générer une erreur de capacité si elle est
(supérieure à (longueur totale max – longueur max partie décimale).
Assert\NotNull() si pas de ‘nullable = true’

- type string (= varchar en bdd)

Assert\Type("string") → seul le type string sera accepté (pas d’integer, float, boolean)
→ cet Assert est il nécessaire ? OUI (Romain)
Assert\Length(max= ?) ? = longueur max définie dans la déclaration orm, 255 par défaut
Assert\NotNull() si pas de ‘nullable = true’ → contrôle seulement l’initialisation du champ, une chaine vide est acceptée.

- type text (= text en bdd)

Assert\Type("string") → seul le type string sera accepté (pas d’integer, float, boolean)
→ cet Assert est il nécessaire ? OUI (Romain)
Assert\NotNull() si pas de ‘nullable = true’
(pas de contrainte sur la longueur : champ text dans postgres sans limite de longueur / string dans php limité à plus de 2Go)

- type enum* (=string en bdd)

Assert\Choice(callback={"\App\Enum*","getValues"}) avec * = nom de la classe
→ la valeur du champ doit exister parmi les valeurs renvoyées par la méthode getValues.
(Les Assert\Type() et Assert\Length() ne sont pas nécessaires car l’Assert\Choice couvre ces cas)

- type boolean

Assert\Type(‘boolean’) → n’accepte que true ou false ou null (pas de numerique ou string = ‘true’ ou ‘false’)
Assert\NotNull() si pas de ‘nullable = true’
(Un boolean à null sera considéré comme false)

- type date ou datetime (= date ou timestamp en bdd) :

Assert\Type(‘datetime’) → cet Assert est-il nécessaire étant donné que les setters devront être typés ?
OUI au cas où les setters sont oubliés (Romain)
Assert\NotNull() si pas de ‘nullable = true’

+ ajouter le typage sur setter si pas fait
(\DateTime $field = null) si ‘nullable = true’
(\DateTime $field) sinon

- type relation
Assert\Type(‘‘Eliberty\RedpillBundle\Model*Interface’’) avec * = NomClasse
→ cet Assert est-il nécessaire étant donné que les setters devront être typés ?
OUI au cas où les setters sont oubliés (Romain)
Assert\NotNull() si pas de ‘nullable = true’ → valide seulement que l’objet à été instancié pas qu’il contient un objet existant en table
Assert\Valid() → lance la validation sur le sous-objet.

(Pas de contrôle de l’existence du sous-objet en table, ni de la possibilité de création d’un sous-objet inexistant ? NON, c’est le programme de mise à jour qui doit gérer cela (Romain).)

+ ajouter le typage sur setter si pas fait (sur l’interface associé plutot que sur le model)
(\NomclasseInterface $field = null) si ‘nullable = true’
(\NomclasseInterface $field) sinon

- Contrainte sur l’entité
pas de contrôle de l’unicité d’un champ ou d’un ensemble de champ ?
NON (Romain) car les contraintes sur un model doivent rester "1er niveau ", cad valider seulement l’objet en cours et non pas l’objet en cours par rapport à d’autres objets.
→ à faire valider par Xavier

(Notes :
- la contrainte UniqueEntity sur le model ne permet plus de faire un test unitaire puisqu’il faut accéder à une base de données pour valider cette contrainte.
- l’utilisation d’un mock pour l’initialisation d’un champ relation faisant parti d’une contrainte UniqueEntity fait planter les tests.)

3/ Mettre en place des tests unitaires avec PHPUnit
→Pour chaque model, écrire une classe de test :
/redpill/tests/Unit/Eliberty/RedpillBundle/Model/AssertValidation/NomClasseValidation.php
(Voir exemple en annexe)

→ Fonctions de tests à implémenter :
(les fonctions de test ne généreront pas d’erreurs sur les-sous-objets par la contrainte valid() en utilisant un mock pour initialiser les sous-objets)

- une fonction de test sans erreur à la validation
Cette fonction sera implémentée sur tous les models.
Tous les champs du model seront initialisés à leur capacité maximum en respectant les contraintes sur le type, la capacité et l’enum.
Les sous-objets seront initialisés avec un mock.

La validation ne devra pas renvoyer d’erreurs.

- une fonction de test sur le type erreur valeur non nulle
Cette fonction sera implémentée sur tous les models.
Aucun champ ne sera initialisé sur le model.

Seuls les champs concernés par le type d'erreur NotNull devront générer un message d'erreur NotNull à la validation. Ceux-ci seront stockés dans un tableau d'erreurs attendues.
La validation devra renvoyer tous les messages d'erreurs attendus et aucun message d'erreurs non attendus.

- une fonction de test sur le type erreur Type
Cette fonction sera implémentée sur tous les models.
Les champs seront initialisés avec un type incorrect.
Pb : Les champs (relation et date) dont le setter teste déjà le Type → Ces champs devront être initialisé avec un type correct (mock pour les relation, datetime pour les date).

Seuls les champs concernés par le type d'erreur Type et choice devront générer un message d'erreur Type ou Choice à la validation. Les messages d'erreurs seront stockés dans un tableau d'erreurs attendues.
La validation devra renvoyer tous les messages d'erreurs attendus et aucun message d'erreurs non attendus.

- une fonction de test sur le type erreur Capacité
Cette fonction sera implémentée sur tous les models.
Les champs seront initialisés avec une longueur supérieure d'un caractère à la longueur maximum définie.
Les sous-objets seront initialisés avec un mock.
Les dates seront initialisées avec un objet datetime.

Seuls, les champs concernés par le type d'erreur Length, Range, ou Choice devront générer une erreur Length, Range ou Choice à la validation. Ceux-ci seront stockés dans un tableau d'erreurs attendues.
La validation devra renvoyer tous les messages d'erreurs attendus et aucun message d'erreurs non attendus.

- une fonction de test sur le type erreur valeur non acceptée (choice)
Cette fonction sera implémentée uniquement sur les models contenant des champs de type enum.
Les champs string seront initialisés avec des valeurs '123456789012...' utilisant toute la longueur du champ.
Les champs numérique seront initialisés avec 123456… utilisant toute la longueur du champ.
Les champs date seront initialisés avec un objet date.
Les sous-objets seront initialisés avec un mock.

Seuls, les champs concernés par le type d'erreur Choice devront générer un message d'erreur à la validation.Ceux-ci seront stockés dans un tableau d'erreurs attendues.
La validation devra renvoyer tous les messages d'erreurs attendus et aucun message d'erreurs non attendus.

- une fonction de test sur les contraintes Valid ??? NON pas pour l'instant (Romain)
(mise en place sensible tant que tous les models n'ont pas été modifiés …)

4/ Automatisation des tests unitaires

L'idéal serait de pouvoir automatiser les tests unitaires dans un seul programme, du genre :

1 - Récupérer une liste des modèles à traiter
- soit à partir d'un tableau ou fichier
- soit à partir de la liste des models générée à partir du répertoire model avec éventuellement un tableau des models avec cas particuliers qui ne doivent pas être traités.

2 - Pour chaque model de la liste :
    a- récupérer la liste des champs du model avec type, longueur, nullable, enum dans le fichier de description orm ,
    b- pour chaque fonction de test définie ci-dessus (voir 3/ Mettre en place des tests unitaires avec PHPUnit)
        - générer les messages d'erreurs attendus à partir de la liste des champs du model
        - initialiser le model à partir de la liste des champs du model
        - lancer la fonction de test.

→ Faisabilité technique ???
→ Intéret ?

B/ Les models concernés par le fichier Validation.yml

Ces models seront étudiés plus en détail dans un 2e temps.

IV – Démarche à suivre.

A/ Etude globale du projet → Terminé

B/ Traitement des models non concernés par le fichier Validation.yml

1/ Etude technique → En cours – à valider
- Lister les models → Terminé
- Lister les types de champs rencontrés → Terminé
- Tests sur les types d'erreurs rencontrées → En cours
- Recherche des contraintes à mettre en place → En cours
- Mise en place démarche à suivre → En cours

2/ Développement → Commencé
→ Pour chaque model non concernés par le fichier Validation.yml (Voir liste des models en annexe)
- créer une branche git par model
- Faire une lecture du fichier .orm et du script sql pour repérer d'éventuels cas particuliers
- Ajouter les assertions dans le model d'après le fichier orm
- écrire le programme de test unitaires d'après le fichier orm
    * duplication d'un programme précédant
    * sur chaque fonction de test :
        maj du tableau des erreurs attendues
        maj de l'initialisation du model
- exécuter le programme de tests
- pusher dans git perso
- passer en exploitation au fil de l'eau

→ Etudier les cas particuliers
- models sans correspondance model/fichier.orm
- models non présents dans /model
- models syrius

C/ Traitement des models concernés par le fichier Validation.yml

1/ Etude technique du problème

2/ Développement

ANNEXE : Liste des models

Doctrine /Resources/config/doctrine/	Models / Model/	validation.yml	nb champs simples
AccessToken.orm.xml			0
Action.orm.xml	Action.php		2
Address.orm.xml	Address.php	Address:	19
AdjustmentMapping.orm.xml	AdjustmentMapping.php		1
Adjustment.orm.xml	Adjustment.php		7
AlfiKeycard.orm.xml		AlfiKeycard:	3
Apartment.orm.xml	Apartment.php		18
ApiToken.orm.xml	ApiToken.php		7
Asset.orm.xml	Asset.php	Asset:	11
AuthCode.orm.xml			0
BankCard.orm.xml			7
BankTransaction.orm.xml	BankTransaction.php		17
BankTransfer.orm.xml		BankTransfer:	2
BookingOrderitem.orm.xml			5
BookingOrder.orm.xml			17
BookingUnit.orm.xml			3
CalendarPrice.orm.xml	CalendarPrice.php		13
Campaign.orm.xml	Campaign.php		8
CashDeskData.orm.xml	CashDeskData.php		13
CashdeskSale.orm.xml	CashdeskSale.php		23
CatalogImport.orm.xml	CatalogImport.php		7
Catalog.orm.xml	Catalog.php		15
Client.orm.xml			5
CodeBatch.orm.xml	CodeBatch.php		3
Configfield.orm.xml	Configfield.php		4
Consumercategory.orm.xml	Consumercategory.php		16
Contact.orm.xml	Contact.php	Contact:	33
Contractor.orm.xml	Contractor.php		7
Country.orm.xml	Country.php		2
Coupon.orm.xml	Coupon.php	Coupon:	5
Dci4crmTracking.orm.xml	Dci4crmTracking.php		16
DetectionStrategy.orm.xml	DetectionStrategy.php		5
Device.orm.xml	Device.php		4
Document.orm.xml	Document.php	Document:	4
Exclusion.orm.xml			7
ExternalNumber.orm.xml	ExternalNumber.php		1
ExternalReference.orm.xml	ExternalReference.php		1
Field.orm.xml	Field.php		3
FiledayProcessing.orm.xml	FiledayProcessing.php		3
FollowUp.orm.xml	FollowUp.php		10
Gate.orm.xml	Gate.php		7
GroupDevice.orm.xml	GroupDevice.php		4
Groupfield.orm.xml	Groupfield.php		2
GroupInstance.orm.xml	GroupInstance.php		3
History.orm.xml	History.php		5
KeycardNumber.orm.xml	KeycardNumber.php	KeycardNumber:	3
Keycard.orm.xml	Keycard.php	Keycard:	19
Mandate.orm.xml	Mandate.php		7
MessageTracking.orm.xml	MessageTracking.php		10

MulticashdeskMaster.orm.xml	MulticashdeskMaster.php		3
MulticashdeskSlave.orm.xml	MulticashdeskSlave.php		0
OpenActivationItem.orm.xml		OpenActivationItem:	6
OpenActivation.orm.xml			3
OpenKeycard.orm.xml		OpenKeycard:	1
Orderitem.orm.xml	Orderitem.php		28
Order.orm.xml	Order.php	Order:	31
Partner.orm.xml	Partner.php	Partner:	22
PartnerPrice.orm.xml	PartnerPrice.php		5
PaymentConfig.orm.xml	PaymentConfig.php		13
PaymentMean.orm.xml	PaymentMean.php		5
Payment.orm.xml	Payment.php	Payment:	11
PermissionOperation.orm.xml	PermissionOperation.php		9
Permission.orm.xml	Permission.php		14
PickupSite.orm.xml	PickupSite.php		12
Pricecategory.orm.xml	Pricecategory.php		13
PriceCheck.orm.xml	PriceCheck.php		10
Price.orm.xml	Price.php		7
PrintableDocument.orm.xml	PrintableDocument.php	PrintableDocument:	2
Processing.orm.xml	Processing.php		12
Productcategory.orm.xml	Productcategory.php		16
ProductChildren.orm.xml	ProductChildren.php	ProductChildren:	7
ProductExclusion.orm.xml	ProductExclusion.php		2
Product.orm.xml	Product.php		38
Promotion.orm.xml	Promotion.php	Promotion:	24
ProposalGenerator.orm.xml	ProposalGenerator.php		9
RefreshToken.orm.xml			0
Renewal.orm.xml	Renewal.php		5
Right.orm.xml	Right.php	Right:	4
Rule.orm.xml	Rule.php		1
SaleschannelConfiguration.orm.xml	SaleschannelConfiguration.php	SaleschannelConfigura	5
Schedule.orm.xml	Schedule.php		4
Season.orm.xml	Season.php		14
ShipmentItem.orm.xml	ShipmentItem.php		0
Shipment.orm.xml	Shipment.php		0
ShippingCategory.orm.xml	ShippingCategory.php	ShippingCategory:	0
ShippingMethod.orm.xml	ShippingMethod.php	ShippingMethod:	1
ShippingRule.orm.xml	ShippingRule.php		0
SkidataKeycard.orm.xml		SkidataKeycard:	0
Skiday.orm.xml	Skiday.php		23
Sponsorship.orm.xml	Sponsorship.php	Sponsorship:	7
Tagging.orm.xml	Tagging.php		0
Tag.orm.xml	Tag.php		2
Tarifsector.orm.xml	Tarifsector.php		11
Taxonomy.orm.xml	Taxonomy.php		1
Taxon.orm.xml	Taxon.php		0
TeamaxessKeycard.orm.xml		TeamaxessKeycard:	0
TemplateCategory.orm.xml	TemplateCategory.php		4
Template.orm.xml	Template.php		6
TemplateSection.orm.xml	TemplateSection.php		4

TicketEventDataAttrib.orm.xml	TicketEventDataAttrib.php		2
TicketEventData.orm.xml	TicketEventData.php		30
TicketOrderitem.orm.xml		TicketOrderitem:	4
TrackingCampaign.orm.xml	TrackingCampaign.php	TrackingCampaign:	11
Tracking.orm.xml	Tracking.php		16
TrackingProcessingDate.orm.xml	TrackingProcessingDate.php		3
TrackingProcessing.orm.xml	TrackingProcessing.php		11
Translation.orm.xml			6
Transmission.orm.xml	Transmission.php		29
User.orm.xml	User.php	User:	7
UserPreference.orm.xml	UserPreference.php		4
Validitycategory.orm.xml	Validitycategory.php		11
Validityperiod.orm.xml	Validityperiod.php		5
Variant.orm.xml	Variant.php	Variant:	7
VoucherCode.orm.xml	VoucherCode.php	VoucherCode:	8
Webhook.orm.xml	Webhook.php	Webhook:	5
WebInstance.orm.xml	WebInstance.php		8
Website.orm.xml	Website.php		13
ZoneMember.orm.xml	ZoneMember.php		3
Zone.orm.xml	Zone.php		5
119		30	
	Models sans .orm :		
	CartDeliveryMode.php		
	Cartitem.php		
	FinalizationOrder.php		
	Opencard.php		
	Proposal.php		
	RestResponse.php		
	Simulation.php		
	SkiCase.php		
	Uploadable.php		
	UserCheckoutIdentification.php		

ANNEXE : Exemple de programme de test = AssetValidation.php

```php
<?php
namespace Tests\Unit\Eliberty\RedpillBundle\Model\AssertValidation;
use Eliberty\RedpillBundle\Entity\Asset;
use Eliberty\RedpillBundle\Enum\AssetTriggerType;
use Eliberty\RedpillBundle\Enum\AssetType;
use Eliberty\RedpillBundle\Model\ContactInterface;
use Eliberty\RedpillBundle\Model\ContractorInterface;
use Symfony\Bundle\FrameworkBundle\Test\KernelTestCase;
use Symfony\Component\Validator\Validator\ValidatorInterface;
/**
 * Class AssetValidation
 * @package Tests\Unit\Eliberty\RedpillBundle\Model\AssertValidation
 */
class AssetValidation extends KernelTestCase
{
    protected const FIELD_NAME    = 'fieldname';
    protected const ERROR_MESSAGE = 'errormessage';
    /**
     * @var ValidatorInterface
     */
    protected $validator;
    /**
     * Set up
     */
    protected function setUp()
    {
        $kernel = self::bootKernel();
        $this->validator = $kernel->getContainer()->get('validator');
    }
    /**
     * {@inheritDoc}
     */
    protected function tearDown()
    {
        parent::tearDown();
        //$this->validator->close();
        $this->validator = null; // avoid memory leaks
    }
    /**
     * @test
     */
    public function validationShouldReturnNoError()
    {
        // all fields are initialised with a valid value -> Validate should
detect no errors on the model
        // No errors expected
        // $errorsExpected = [];
        // Initialize model
        // set the sub-object with a mock to disable the constraint Assert\Valid
        $Asset = new Asset();
        $Asset
            ->setContact($this->createMock(ContactInterface::class))
            ->setContractor($this->createMock(ContractorInterface::class))
            ->setCredit(1234567890)
            ->setDebit(1234567890)
            ->setrelatedTo(
                "12345678901234567890123456789012345678901234567890" .
```

```
                "12345678901234567890123456789012345678901234567890" .
                "12345678901234567890123456789012345678901234567890" .
                "12345678901234567890123456789012345678901234567890" .
                "12345678901234567890123456789012345678901234567890123456789012345")
            ->setrelatedToEntity(
                "12345678901234567890123456789012345678901234567890" .
                "12345678901234567890123456789012345678901234567890" .
                "12345678901234567890123456789012345678901234567890" .
                "12345678901234567890123456789012345678901234567890" .
                "12345678901234567890123456789012345678901234567890123456789012345")
            ->setMessage(
                "12345678901234567890123456789012345678901234567890" .
                "12345678901234567890123456789012345678901234567890" .
                "12345678901234567890123456789012345678901234567890" .
                "12345678901234567890123456789012345678901234567890" .
                "12345678901234567890123456789012345678901234567890123456789012345678901234567890")
            ->setType(AssetType::MONEY)
            ->setTrigger(AssetTriggerType::ADJUSTMENT);
        // launch validator on model
        $errors = $this->validator->validate($Asset);
        // collect the errors -> detected by validator not expected
        $errorsDetected_NotExpected = [];
        $i = 0;
        foreach ($errors as $error) {
            $errorsDetected_NotExpected[$i] = [
                self::FIELD_NAME => $error->getPropertyPath(),
                self::ERROR_MESSAGE => $error->getMessage()
            ];
            $i++;
        }
        // Show the result of the test function with an Assertion assertTrue
        // -> list of errors Detected by validator not Expected
        $i=\iter\count($errorsDetected_NotExpected);
        if ( $i > 0 ) {
            print("AssetValidation - ValidationShouldReturnNoErrors() : " . $i .
" errors detected by validator not expected : ");
            print_r($errorsDetected_NotExpected);
        }
        $this->assertTrue(\iter\isEmpty($errorsDetected_NotExpected),
            "validationShouldReturnNoError() : See the errors above");
    }
    /**
     * @test
     */
    public function validationShouldReturnNotNullErrors()
    {
        // all fields = null value -> Validate detect the NotNull errors on the
model
        // define the expected errors
        $errorsExpected = [
            [
                self::FIELD_NAME    => 'type',
                self::ERROR_MESSAGE => 'This value should not be null.'
            ],
            [
                self::FIELD_NAME    => 'trigger',
                self::ERROR_MESSAGE => 'This value should not be null.'
            ],
            [
```

```
                self::FIELD_NAME    => 'credit',
                self::ERROR_MESSAGE => 'This value should not be null.'
            ],
            [
                self::FIELD_NAME    => 'debit',
                self::ERROR_MESSAGE => 'This value should not be null.'
            ],
        ];
        // Initialize model
        $Asset = new Asset();
        // launch validator on model
        $errors = $this->validator->validate($Asset);
        // collect the errors -> detected by validator not expected
        //                    -> expected not detected by validator
        $errorsDetected = [];
        $i = 0;
        foreach ($errors as $error) {
            $errorsDetected[$i] = [
                self::FIELD_NAME => $error->getPropertyPath(),
                self::ERROR_MESSAGE => $error->getMessage()
            ];
            $i++;
        }
        $errorsDetected_NotExpected = array_filter($errorsDetected,
            function ($error) use ($errorsExpected) { return !in_array($error,
$errorsExpected); });
        $errorsExpected_NotDetected = array_filter($errorsExpected,
            function ($error) use ($errorsDetected) { return !in_array($error,
$errorsDetected); });
        // Show the result of the test function with an Assertion assertTrue
        // -> list of errors Expected not detected by validator / Detected by
validator not Expected
        $i=\iter\count($errorsExpected_NotDetected);
        if ( $i > 0 ) {
            print("AssetValidation - validationShouldReturnNotNullErrors() : " .
$i . " NotNull errors expected not detected by validator : ");
            print_r($errorsExpected_NotDetected);
        }
        $i=\iter\count($errorsDetected_NotExpected);
        if ( $i > 0 ) {
            print("AssetValidation - validationShouldReturnNotNullErrors() : " .
$i . " errors detected by validator not expected : ");
            print_r($errorsDetected_NotExpected);
        }
        $this->assertTrue(\iter\isEmpty($errorsDetected_NotExpected) &&
\iter\isEmpty($errorsExpected_NotDetected),
            "validationShouldReturnNotNullErrors() : See the errors above");
    }
    /**
     * @test
     */
    public function validationShouldReturnTypeErrors()
    {
        // fields are initialize with a wrong type (string with integer, integer
with string)
        // except date and relation fields because setter are already typed
        // -> validate detect the type errors on model
        // define the expected errors
        $errorsExpected = [
```

```
            [
                self::FIELD_NAME    => 'credit',
                self::ERROR_MESSAGE => 'This value should be of type integer.'
            ],
            [
                self::FIELD_NAME    => 'debit',
                self::ERROR_MESSAGE => 'This value should be of type integer.'
            ],
            [
                self::FIELD_NAME    => 'relatedTo',
                self::ERROR_MESSAGE => 'This value should be of type string.'
            ],
            [
                self::FIELD_NAME    => 'relatedToEntity',
                self::ERROR_MESSAGE => 'This value should be of type string.'
            ],
            [
                self::FIELD_NAME    => 'message',
                self::ERROR_MESSAGE => 'This value should be of type string.'
            ],
            [
                self::FIELD_NAME    => 'type',
                self::ERROR_MESSAGE => 'The value you selected is not a valid
choice.'
            ],
            [
                self::FIELD_NAME    => 'trigger',
                self::ERROR_MESSAGE => 'The value you selected is not a valid
choice.'
            ],
        ];
        // Initialize model
        // set the sub-object with a mock to disable the constraint Assert\Valid
        $Asset = new Asset();
        $Asset
            ->setContact($this->createMock(ContactInterface::class))
            ->setContractor($this->createMock(ContractorInterface::class))
            ->setCredit('123')
            ->setDebit('123')
            ->setrelatedTo(123)
            ->setrelatedToEntity(123)
            ->setMessage(123)
            //->setType(AssetType::MONEY)
            ->setType(123)
            ->setTrigger(123);
        // launch validator on model
        $errors = $this->validator->validate($Asset);
        // collect the errors -> detected by validator not expected
        //                    -> expected not detected by validator
        $errorsDetected = [];
        $i = 0;
        foreach ($errors as $error) {
            $errorsDetected[$i] = [
                self::FIELD_NAME => $error->getPropertyPath(),
                self::ERROR_MESSAGE => $error->getMessage()
            ];
            $i++;
        }
        $errorsDetected_NotExpected = array_filter($errorsDetected,
```

```
            function ($error) use ($errorsExpected) { return !in_array($error,
$errorsExpected); });
        $errorsExpected_NotDetected = array_filter($errorsExpected,
            function ($error) use ($errorsDetected) { return !in_array($error,
$errorsDetected); });
        // Show the result of the test function with an Assertion assertTrue
        // -> list of errors Expected not detected by validator / Detected by
validator not Expected
        $i=\iter\count($errorsExpected_NotDetected);
        if ( $i > 0 ) {
            print("AssetValidation - validationShouldReturnTypeErrors() : " . $i
. " type errors expected not detected by validator : ");
            print_r($errorsExpected_NotDetected);
        }
        $i=\iter\count($errorsDetected_NotExpected);
        if ( $i > 0 ) {
            print("AssetValidation - validationShouldReturnTypeErrors() : " . $i
. " errors detected by validator not expected : ");
            print_r($errorsDetected_NotExpected);
        }
        $this->assertTrue(\iter\isEmpty($errorsDetected_NotExpected) &&
\iter\isEmpty($errorsExpected_NotDetected),
            "validationShouldReturnTypeErrors() : See the errors above");
    }
    /**
     * @test
     */
    public function validationShouldReturnLengthErrors()
    {
        // fields are initialize with a length = max length + 1  (not for
relation or date fields)
        // -> validate detect the length errors on model
        // define the expected errors
        $errorsExpected = [
            [
                self::FIELD_NAME    => 'credit',
                self::ERROR_MESSAGE => 'This value is too long. It should have
10 character or less.'
            ],
            [
                self::FIELD_NAME    => 'debit',
                self::ERROR_MESSAGE => 'This value is too long. It should have
10 character or less.'
            ],
            [
                self::FIELD_NAME    => 'relatedTo',
                self::ERROR_MESSAGE => 'This value is too long. It should have
255 character or less.'
            ],
            [
                self::FIELD_NAME    => 'relatedToEntity',
                self::ERROR_MESSAGE => 'This value is too long. It should have
255 character or less.'
            ],
            [
                self::FIELD_NAME    => 'type',
                self::ERROR_MESSAGE => 'The value you selected is not a valid
choice.'
            ],
```

```
            [
                self::FIELD_NAME    => 'trigger',
                self::ERROR_MESSAGE => 'The value you selected is not a valid
choice.'
            ],
        ];
        // Initialize model
        // set the sub-object with a mock to disable the constraint Assert\Valid
        $Asset = new Asset();
        $Asset
            ->setContact($this->createMock(ContactInterface::class))
            ->setContractor($this->createMock(ContractorInterface::class))
            ->setCredit(12345678901)
            ->setDebit(12345678901)
            ->setrelatedTo(
                "12345678901234567890123456789012345678901234567890" .
                "12345678901234567890123456789012345678901234567890" .
                "12345678901234567890123456789012345678901234567890" .
                "12345678901234567890123456789012345678901234567890" .
                "12345678901234567890123456789012345678901234567890123456")
            ->setrelatedToEntity(
                "12345678901234567890123456789012345678901234567890" .
                "12345678901234567890123456789012345678901234567890" .
                "12345678901234567890123456789012345678901234567890" .
                "12345678901234567890123456789012345678901234567890" .
                "12345678901234567890123456789012345678901234567890123456")
            ->setMessage(
                "12345678901234567890123456789012345678901234567890" .
                "12345678901234567890123456789012345678901234567890" .
                "12345678901234567890123456789012345678901234567890" .
                "12345678901234567890123456789012345678901234567890" .
                "12345678901234567890123456789012345678901234567890123456789012345678901234567890")
            ->setType(
                "12345678901234567890123456789012345678901234567890" .
                "12345678901234567890123456789012345678901234567901")
            ->setTrigger(
         "12345678901234567890123456789012345678901234567890" .
                "12345678901234567890123456789012345678901234567901");
        // launch validator on model
        $errors = $this->validator->validate($Asset);
        // collect the errors -> detected by validator not expected
        //                    -> expected not detected by validator
        $errorsDetected = [];
        $i = 0;
        foreach ($errors as $error) {
            $errorsDetected[$i] = [
                self::FIELD_NAME => $error->getPropertyPath(),
                self::ERROR_MESSAGE => $error->getMessage()
            ];
            $i++;
        }
        $errorsDetected_NotExpected = array_filter($errorsDetected,
            function ($error) use ($errorsExpected) { return !in_array($error,
$errorsExpected); });
        $errorsExpected_NotDetected = array_filter($errorsExpected,
            function ($error) use ($errorsDetected) { return !in_array($error,
$errorsDetected); });
        // Show the result of the test function with an Assertion assertTrue
```

```
        // -> list of errors Expected not detected by validator / Detected by
validator not Expected
        $i=\iter\count($errorsExpected_NotDetected);
        if ( $i > 0 ) {
            print("AssetValidation - validationShouldReturnLengthErrors() : " .
$i . " Length errors expected not detected by validator : ");
            print_r($errorsExpected_NotDetected);
        }
        $i=\iter\count($errorsDetected_NotExpected);
        if ( $i > 0 ) {
            print("AssetValidation - validationShouldReturnLengthErrors() : " .
$i . " errors detected by validator not expected : ");
            print_r($errorsDetected_NotExpected);
        }
        $this->assertTrue(\iter\isEmpty($errorsDetected_NotExpected) &&
\iter\isEmpty($errorsExpected_NotDetected),
            "validationShouldReturnLengthErrors() : See the errors above");
    }
    /**
     * @test
     */
    public function validationShouldReturnChoiceErrors()
    {
        // fields are initialize with a value non existent in the enum values
        // -> validate detect the choice errors on model
        // define the expected errors
        $errorsExpected = [
            [
                self::FIELD_NAME    => 'type',
                self::ERROR_MESSAGE => 'The value you selected is not a valid
choice.'
            ],
            [
                self::FIELD_NAME    => 'trigger',
                self::ERROR_MESSAGE => 'The value you selected is not a valid
choice.'
            ],
        ];
        // Initialize model
        // set the sub-object with a mock to disable the constraint Assert\Valid
        $Asset = new Asset();
        $Asset
            ->setContact($this->createMock(ContactInterface::class))
            ->setContractor($this->createMock(ContractorInterface::class))
            ->setCredit(1234567890)
            ->setDebit(1234567890)
            ->setrelatedTo(
                "12345678901234567890123456789012345678901234567890" .
                "12345678901234567890123456789012345678901234567890" .
                "12345678901234567890123456789012345678901234567890" .
                "12345678901234567890123456789012345678901234567890" .
                "123456789012345678901234567890123456789012345678901234567890123456789012345")
            ->setrelatedToEntity(
                "12345678901234567890123456789012345678901234567890" .
                "12345678901234567890123456789012345678901234567890" .
                "12345678901234567890123456789012345678901234567890" .
                "12345678901234567890123456789012345678901234567890" .
                "123456789012345678901234567890123456789012345678901234567890123456789012345")
            ->setMessage(
```

```
                "12345678901234567890123456789012345678901234567890" .
                "12345678901234567890123456789012345678901234567890" .
                "12345678901234567890123456789012345678901234567890" .
                "12345678901234567890123456789012345678901234567890" .
                "12345678901234567890123456789012345678901234567890")
            ->setType(
                "12345678901234567890123456789012345678901234567890" .
                "12345678901234567890123456789012345678901234567890")
            ->setTrigger(
                "12345678901234567890123456789012345678901234567890" .
                "12345678901234567890123456789012345678901234567890");
        // launch validator on model
        $errors = $this->validator->validate($Asset);
        // collect the errors -> detected by validator not expected
        //                    -> expected not detected by validator
        $errorsDetected = [];
        $i = 0;
        foreach ($errors as $error) {
            $errorsDetected[$i] = [
                self::FIELD_NAME => $error->getPropertyPath(),
                self::ERROR_MESSAGE => $error->getMessage()
            ];
            $i++;
        }
        $errorsDetected_NotExpected = array_filter($errorsDetected,
            function ($error) use ($errorsExpected) { return !in_array($error,
$errorsExpected); });
        $errorsExpected_NotDetected = array_filter($errorsExpected,
            function ($error) use ($errorsDetected) { return !in_array($error,
$errorsDetected); });
        // Show the result of the test function with an Assertion assertTrue
        // -> list of errors Expected not detected by validator / Detected by
validator not Expected
        $i=\iter\count($errorsExpected_NotDetected);
        if ( $i > 0 ) {
            print("AssetValidation - validationShouldReturnChoiceErrors() : " .
$i . " choice errors expected not detected by validator : ");
            print_r($errorsExpected_NotDetected);
        }
        $i=\iter\count($errorsDetected_NotExpected);
        if ( $i > 0 ) {
            print("AssetValidation - validationShouldReturnChoiceErrors() : " .
$i . " errors detected by validator not expected : ");
            print_r($errorsDetected_NotExpected);
        }
        $this->assertTrue(\iter\isEmpty($errorsDetected_NotExpected) &&
\iter\isEmpty($errorsExpected_NotDetected),
            "validationShouldReturnChoiceErrors() : See the errors above");
    }
}
```

ANNEXE : Démarche pour la recherche des types de champs rencontrés :

```
 790  grep  "type" src/Eliberty/RedpillBundle/Resources/config/doctrine/* >zzzisa/listType
 797  grep  -v "string" zzzisa/listType >zzzisa/listType1
 798  cat zzzisa/listType1
 799  grep  -v "integer" zzzisa/listType1 >zzzisa/listType2
 800  cat zzzisa/listType2
 801  grep  -v "boolean" zzzisa/listType2 >zzzisa/listType3
 802  cat zzzisa/listType3
 803  grep  -v "datetime" zzzisa/listType3 >zzzisa/listType4
 804  cat zzzisa/listType4
 805  grep  -v "text" zzzisa/listType4 >zzzisa/listType5
 806  cat zzzisa/listType5
 807  grep  "enum" zzzisa/listType5 >zzzisa/listTypeEnum
 808  cat zzzisa/listTypeEnum
 809  grep  -v "enum" zzzisa/listType5 >zzzisa/listType6
 810  cat zzzisa/listType6
 811  grep  -v "date" zzzisa/listType6 >zzzisa/listType7
 812  cat zzzisa/listType7
 813  grep  -v "decimal" zzzisa/listType7 >zzzisa/listType8
 814  cat zzzisa/listType8
 815  grep  -v "amount" zzzisa/listType8 >zzzisa/listType9
 816  cat zzzisa/listType9
 817  grep  -v "preFlush" zzzisa/listType9 >zzzisa/listType10
 818  cat zzzisa/listType10
 819  cat zzzisa/listType9
 820  grep  -v "json_array" zzzisa/listType9 >zzzisa/listType10
 821  cat zzzisa/listType10
 822  grep "array" zzzisa/listType10
 823  grep  -v "array" zzzisa/listType10 >zzzisa/listType11
 824  cat zzzisa/listType11
```