**I) Les convertisseurs de données avec Symfony2 : les dataTransformers.**

Il existe deux types de dataTransformers : les modelTransformers et les viewTransformers.

Les premiers permettent de transformer la donnée modèle en donnée normalisée alors utilisable par la vue, et les seconds permettent de transformer la donnée normalisée en donnée vue.

Les données modèle sont celles que l’on récupère par la méthode getData, ou qui sont settées par la méthode setData (méthodes du formulaire)

Les données normalisées sont généralement identiques aux données modèles.

Les données vues sont utilisées pour remplir les champs. La méthode bind($data) du formulaire utilise le format de la vue lors de la soumission des formulaires.

Tout dépend de ce que nous voulons faire. S’il s’agit uniquement de modifier l’information soumise (enregistrer un champ en lettres capitales, par exemple, ou changer le format d’une date), alors ce sont les viewTransformers qu’il faut utiliser.

S’il faut convertir une donnée soumise par le formulaire en un objet, ou en un autre type, alors les modelTransformers sont plus appropriés.

**Cas pratique : un champ texte avec un datepicker que nous voulons convertir en date pour la base de données.**

Dans un premier temps, il faut déclarer le dataTransformer dans le formType :



Le dataTransformer associé à notre champs dans le formulaire, il faut à présent coder ses deux fonctions : transform et reverseTransform.

1) Le dataTransformer va récupérer le champs texte lors de la soumission du formulaire et le convertir en objet dateTime par la fonction reverseTransform
2) Le dataTransformer, lors de l’édition d’un objet (url du type {objectID}/edit), va récupérer l’information en base de donnée (donc le datetime) et le convertir en champ texte par la fonction transform.



D’autres cas intéressants peuvent nécessiter l’utilisation d’un dataTransformer : transformer des données soumises sous forme d’array en une chaine de caractère séparée par des virgules (pour le cas de tags d’un article, par exemple), convertir un numéro d’un champ int en un objet enregistré en base, etc.

Symfony2 comporte un certain nombre de dataTransformers natifs :



**II) Les événements**

Les **événements** du formulaire permettent de **modifier/filtrer le** **contenu** (liste déroulante, changer casse, etc.) **de la donnée** ou bien de modifier la **structure du formulaire**. (ajout/suppression de champs par ex.)

Il existe cinq types d’événements qui interviennent à différents moments de la vie d’un formulaire :

* PRE\_SET\_DATA
* POST\_SET\_DATA
* PRE\_SUBMIT
* SUBMIT
* POST\_SUBMIT

Imaginons que nous voulons pouvoir uploader 3 fichiers maximum par article. Pour cela, nous devons faire trois choses :

1. créer le type File et faire une relation oneToMany entre l’entité Article et l’entité File
2. créer le formType FileType et l’enregistrer en tant que service
3. rajouter le champ en tant que collection et y placer un listener de type PRE\_SET\_DATA pour limiter le nombre de champs d’upload à 3
4. Le fichier File et la relation entre Article et File





1. L’enregistrement en tant que service



1. La collection et l’event



Nous remarquerons que le type de la collection est le nom de l’alias du formType (appbundle\_file)

Le listener va vérifier (à la création ou à l’édition d’un article) qu’il y a bien trois fichiers au maximum qui peuvent être ajoutés à l’article (en se basant sur la constante FILES\_LIMIT définie à 3 dans l’entité Article)

**III) Les extensions de type**

Les extensions de type de formulaire ont deux utilisations principales :

1. Vous voulez ajouter une **fonctionnalité générique sur plusieurs types** (comme ajouter un texte d'« aide » sur tout les types de champ);
2. Vous voulez ajouter une **fonctionnalité spécifique sur un type** (comme ajouter une fonctionnalité « téléchargement » sur un type de champ « file »).

Imaginons que nous voulons qu’il y ait deux auteurs maximum pour chaque article, mais que l’on laisse le choix à l’utilisateur d’en mettre un ou deux (code js de collection). Il faut pouvoir passer à la vue le paramètre 2.

Pour cela, nous devons faire 5 choses :

1. créer le type Author et faire une relation OneToMany entre l’entité Article et l’entité Author.
2. Créer le formType AuthorType et l’enregistrer en tant que service
3. Créer le formType AuthorCollectionType, l’enregistrer en tant que service et implémenter la fonction buildView pour passer des paramètres à la vue
4. Rajouter la collection à ArticleType
5. Créer un form\_theme pour mettre en forme la collection
6. le fichier Author et la relation entre Article et Author





1. Le formType AuthorType et son service





1. Le fichier AuthorCollectionType, son service et la fonction buildView





1. Rajouter la collection à ArticleType



1. Créer un form\_theme pour mettre en forme la collection







On remarquera que le nom des blocs correspond aux noms des formType (fonction getName) qui est également celui de leur alias (dans les services) suffixé de \_row.

La variable max\_limit étant passée dans la vue, on peut la réutiliser dans en js pour lancer une erreur si plus de 2 auteurs essaient d’être ajoutés :

