**InBORe : Interface de gestion de BDD Orientée Recherche**

Documentation Developpeur

InBORe 1.0 (Symfony 5.4LTS / PHP 8.1)

Philippe Grison

20/06/2023

Table des matières

[1- Installation de Symfony 3](#_Toc153206765)

[2 – Installation d’un projet InBORe 3](#_Toc153206766)

[2a – copie /clone du projet InBORe 3](#_Toc153206767)

[2b– Installation de node.js et de Yarn package manager 3](#_Toc153206768)

[2c – Initialisation du projet (Windows) 4](#_Toc153206769)

[2d – Initialisation du serveur Web (WAMP) 4](#_Toc153206770)

[3 – Initialisation du projet : BDD, mapping ORM , table user\_db, langues 5](#_Toc153206771)

[3a – modélisation de la base : pré-requis 5](#_Toc153206772)

[3b- Paramétrage de la connexion à la base de données 5](#_Toc153206773)

[3c- Mapping ORM & création des Entités (id, contraints, getter et setter) 5](#_Toc153206774)

[3d- Ajout à la BBD la table « user\_db » 7](#_Toc153206775)

[3e- Langage(s) & bilinguisme de l’interface 8](#_Toc153206776)

[3f- Corriger les erreurs et les contraintes des Entités et de de la BDD 8](#_Toc153206777)

[4 - Génération et mise à jour des formulaires CRUD / Entity 10](#_Toc153206778)

[4a – génération des formulaires CRUD 10](#_Toc153206779)

[4b – ajuster les Repository 11](#_Toc153206780)

[4c – ajuster les formulaires 12](#_Toc153206781)

[4c – ajuster les templates TWIG ; index.html.twig, edit.html.twig 16](#_Toc153206782)

[4d – ajuster les versions bilingues des ; menus , libellés et mentions légales 16](#_Toc153206783)

[5 - Modifications/Evolutions des entités & formulaires 17](#_Toc153206784)

[5a – Ajout d’une Collection (relation 1-N ou N-N) 17](#_Toc153206785)

[5b – Ajout d’un attribut et MAJ du schéma de bdd 17](#_Toc153206786)

[5c – Ajout contrainte UNIQUE et MAJ du schéma de bdd 18](#_Toc153206787)

[6– Gestion centralisée des assets (js, css) avec Encore 19](#_Toc153206788)

[6a- Configuration du WebpackEncoreBundle 19](#_Toc153206789)

[6b- Ajout de modules Javascript 20](#_Toc153206790)

[6c- Composants et librairies js 21](#_Toc153206791)

[7 – Bundle Symfony – Usage & Déploiement 22](#_Toc153206792)

[7a – Info sur le bundle FOSJSRoutingBundle 2.x 22](#_Toc153206793)

[8 – Extension du projet InBORe : Services – Usage & Description 23](#_Toc153206794)

[8a – Service FileLoader 23](#_Toc153206795)

[Annexe 1 – Architecture logicielle – InBORe 25](#_Toc153206796)

# Installation de Symfony

L’installation peut s’effectuer de plusieurs manières via le composer

Pré-requis coté serveur Apache pour l’installation :

* Verifier ou installer le bon environnement de développement (PHP + Apache) , par exemple WAMP

<https://symfony.com/doc/5.4/setup.html>  
 verifier que certaines extensions sont “enabled” ; Ctype, iconv, JSON, PCRE, Session, SimpleXML, Tokenizer.Y ajouter en + les extensions php pour une base postgres : php\_pdo\_pgsql, php\_pgsql, php\_openssl

RQ – config pour php conseillé ; realpath\_cache\_size > 5M , accelerateur PHP enable (OPcache)

* installer le client Symfony : <https://symfony.com/download>  
  (i) download du binary amd64 à dezipper dans C:\Program Files   
  (ii) ajouter le PATH dans les variables d’environnement Windows des « Propriétés système » [Variables d’environnement] choisir dans la liste des « variables système » Path + [Modifier]+[Nouvelle] et ajouter le chemin du client Symfony.exe : C:\Program Files\Symfony

# 2 – Installation d’un projet InBORe

## 2a – copie /clone du projet InBORe

* cd path\_to\_project\_inbore
* **git clone** [git@gitlab.in2p3.fr:bbees/projets-bbees/inBORe.git](mailto:git@gitlab.in2p3.fr:bbees/projets-bbees/inBORe.git)  
  ou

**git clone** [git@github.com:philgris/inbore.git](mailto:git@github.com:philgris/inbore.git)

* renommer le dossier /inbore créé

## 2b– Installation de node.js et de Yarn package manager

Ouvrir une invite de commande (cmd) et #cd path\_to\_project. Lancer les deux commandes ;

1. Installation de Node.js en version v14.x  
   <https://blog.eldernode.com/install-node-js-on-windows>  
   <https://nodejs.org/download/release/v14.21.1/>

Pour Windows : choisir la version 64 bit .msi (ou .zip)  
RQ - via l’installer windows (.msi) choisir l’option “windows-build-tools”

* Tester l’installation **avec #node -v ou # npm –v**

1. Installation des binaires de Yarn<https://yarnpkg.com/getting-started/install> **# npm install -g yarn  
   # cd ~/path\_to\_project  
   # yarn set version classic**! il faut choisir la version classic car le bundle webpack.encore fonctionne uniquement pour le moment avec une version 1.x de yarn (# yarn -v) car la version 2.x est pour le moment incompatible avec Encore   
   ! la prise en compte des modifications de la variable system %PATH% nécéssite un redemarrage de Windows. Elle doit comprendre les paths vers ; nodejs , npm, Yarn\bin

## 2c – Initialisation du projet (Windows)

* **# cd path\_to\_the\_local\_project**

**# set PATH = path\_to\_version\_PHP\_8.1 ; %PATH%**TIP – verifier la version de PHP de la console : **# php –v**

* **# php composer.phar install**  
  => installe les dépendances php à partir du fichier composer.lock qui a été créée à partir du fichier composer.json  
  TIP- installer le client Symfony et tester l’installation avec la commande : **# symfony check:requirement**TIP- pour verifier quelle est la version de Symfony, PHP du projet : **# php bin/console about**TIP- pour verifier les failles de sécurité connues : **# symfony check:security**
* **# yarn install --force --frozen-lockfile**  
  => installe les dépendances js à partir du fichier yarn.lock qui a été créée à partir du fichier package.json. Toutes les librairies sont stockées dans le dossier node\_modules

RQ – Pour etre assuré qu’il n’y a pas de reecriture du fichier yarn.lock il faut rajouter **--frozen-lockfile**RQ  – le yarn.lock et le package.json sont compatible avec une version 18 de node.js  
Installe les dépendances listées dans le fichier package.json dans le dossier /node\_modules et crée ou met à jour le fichier yarn.lockEx. Installe deux packages supplémentaires utiles quand on utilise Encore avec Symfony ; core-js & webpack-notifier. Tous ces packages js sont stockés dans un dossier node\_modules/   
NOTE – cf. doc sur les commande yarn <https://classic.yarnpkg.com/en/docs/cli/install/>

* **# php bin/console assets:install**  
  => installe les assets des bundles dans le dossier /public/bundles/
* **# yarn encore dev**=> execute webpack.config.js ; compile les js une fois dans le dossier /public/build/ …

## 2d – Initialisation du serveur Web (WAMP)

- Déclarer un nouvel virtualhost pour l’accès aux applicatifs web du dossier partagé.

via le menu : CD **[WAMP icone] + [Vos virtualhost][Gestion Virtualhost]**

– il faut redémarrer Apache et le DNS pour la prise en compte de ce nouvel virtualhost :  
CD **[WAMP icone] + [Outils][redemarrage DNS]**

RQ – l’accès au projet via le virtualhost utilise le module rewrite d’Apache via le fichier /public/.htaccess

# 3 – Initialisation du projet : BDD, mapping ORM , table user\_db, langues

## 3a – modélisation de la base : pré-requis

Plusieurs pré-requis sur la modélisation de la base sont à respecter pour assurer un bon fonctionnement de l’ORM Doctrine :

* ! Toutes les tables et champs doivent être en minuscule.
* Toutes les tables ont des clés primaires et si besoin executer les 3 commandes SQL (postgreSQL) :  
  **CREATE SEQUENCE *public.nomtable\_id\_se*q START WITH 1 INCREMENT BY 1 NO MINVALUE NO MAXVALUE CACHE 1;  
  ALTER TABLE nomtable ADD COLUMN id INTEGER DEFAULT nextval(*'public. nomtable\_id\_seq*'::regclass) NOT NULL ;  
  ALTER TABLE *nomtable* ADD PRIMARY KEY (id) ;**
* Toutes les tables doivent disposer d’une clés primaire de préférence nommée ‘id’.   
  Si tel n’est pas le cas il faudra dans un deuxième temps renommer le nom des variables associées à la clès primaires (par ex $entityName\_pk) en $id pour toutes les entités (cf. §3c.c)
* Le nom des tables et des champs doivent de préférence être nommés sans caractère spéciaux pour éviter que Doctrine les renomme ce qui entraine une complexité supplémentaire pour la lecture des variables et du code (par ex la variable PHP associé à ‘nom\_champ’ sera traduit en $nomChamp )
* ! Les clés étrangères doivent être nommées suivant une dénomination commune du type :  
  *prefixe\_nomTableLiée\_suffixe*  
  avec un prefixe et suffixe paramétrable dans le fichier de configuration admin.yaml (cf. section fk\_name).   
  RQ – il faut au moins que l’un des deux soit définit, par ex. prefixe\_fk = ‘*id’* => fk = ‘*id\_nomTableLiee’*

## 3b- Paramétrage de la connexion à la base de données

Se reporter aux documentations:

Se repporter à la documentation « Database and the Doctrine ORM » :  
<https://symfony.com/doc/4.4/doctrine.html>

<https://symfony.com/doc/4.4/reference/configuration/doctrine.html>

<https://www.doctrine-project.org/projects/doctrine-dbal/en/current/reference/configuration.html>

* Ajuster les paramètres de connexion à la base de données définis dans le fichier **.env** . exemple :***DATABASE\_URL="* *postgresql://db\_user:db\_psw@127.0.0.1:5432/db\_name?serverVersion=X&charset=utf8 "***RQ – ce fichier peut être sauvegarder sous le nom .env.localRQ – pour verifier et debugger la config de Doctrine : **php bin/console debug:config doctri*ne***RQ – par defaut c’est le driver ‘pdo\_mysql’. Remplacer par ‘pdo\_pgsql’ pour une base postgreSQL  
  RQ – il faut avoir initialement activé les modules php relatifs à postgres (php\_pgsql, php\_pdo\_pgsql)

## 3c- Mapping ORM & création des Entités (id, contraints, getter et setter)

1. Génération des Entity :

**php bin/console doctrine:mapping:import "App\Entity" annotation --path=src/Entity**RQ ! cette commande supprime les entités existantes  
RQ - bien vérifier que la table user\_db en est exclu. Elle sera créée ultérieurement cf §3TIP – choisir ‘annotation’ comme argument est une bonne pratique  
RQ ! – la console execute la version de PHP spécifié par le PATH et charge le php.ini associé à cette version qui peut être différente de celle de WAMP. Mettre à jour cette version de php.ini identique de celle testée avec WAMP pour avoir les bonnes extensions (par ex. pdo\_pgsql)  
RQ – Si il existe des type non standard (ex geometry) ajouter dans le fichier de config de doctrine (doctrine.yaml) la sous-section dbal :mapping\_types: avec par exemple *geometry: string*

1. Ajout de la mention du « repositoryClass » pour les entités Table-Objet de la BDD-R

/\*\*  
\* *Table*  
\*  
 \* @ORM\Table(name="*table* ")  
 **\*** @ORM\Entity**(repositoryClass="App\Repository\Core\*TableRepository*")**  
 \*   
 \*/

RQ – inutile d’ajouter cette annotation pour les tables relationnelles

1. Si besoin ajuster dans les Entity le nom de variable et les annotations des clés primaires pour qu’elles soient nommée $id et avec des déclarations conforme à leur type (INT/VARCHAR,…)   
   (i) Le cas de clès primaire INTEGER : modifier l’intitulé de la variable associée à la clés primaire  
   # **private $*idtable*** => **private $id**(ii) Le casde clès de type VARCHAR : modifier l’intitulé de la variable associée à la clés primaire # **private $*idtable*** => **private $id**supprimer les annotations relatives à la génération automatisée de valeurs- **@ORM\GeneratedValue( …)**- **@ORM\SequenceGenerator( …)**
2. les contraintes des valeurs par default des sequences auto dans PostgreSQL (PK id) ne sont pas correctement transcrit pour PostgreSQL il faut changer la strategy="SEQUENCE" par "**IDENTITY**"  
   RQ – cette modification n’est plus necessaire en Symfony 5\*

@ORM\GeneratedValue(strategy="**IDENTITY**")

RQ : la modification de la valeur par defaut ( @ORM\Column(name="id", type="bigint", **nullable=false**, **options={"default"="nextval(*'adn\_est\_realise\_par\_id\_seq*'::regclass)"}**)) serait aussi possible mais conserve la commande SQL (SET DEFAULT ...) dans le dumpSQL

1. Ajouter ou ajuster/renommer toutes les variables des champs techniques dans les Entity
2. Cas ou elle n’existe pas : Ajouter le bloc de code à chaque Entity

**/\*\***

**\* @var \DateTime**

**\***

**\* @ORM\Column(name="date\_of\_creation", type="datetime", nullable=true)**

**\*/**

**private $dateCre;**

**/\*\***

**\* @var \DateTime**

**\***

**\* @ORM\Column(name="date\_of\_update", type="datetime", nullable=true)**

**\*/**

**private $dateMaj;**

**/\*\***

**\* @var int**

**\***

**\* @ORM\Column(name="creation\_user\_name", type="bigint", nullable=true)**

**\*/**

**private $userCre;**

**/\*\***

**\* @var int**

**\***

**\* @ORM\Column(name="update\_user\_name", type="bigint", nullable=true)**

**\*/**

**private $userMaj;**

(ii) Cas ou elles existent : Ajuster les nom des variable si besoin (Optionnel)

# **private $*usermaj*; => private $userMaj;**

# **private $*usercre*; => private $userCre;**

# **private $*datemaj*; => private $dateMaj;**

# **private $*datecre*; => private $dateCre;**

RQ - Si les champs existent déjà il est possible d’indiquer leur nom dans les annotations, soit d’ajouter les getters et setters sur les champs : setUserMaj($userMaj) { $this→myField = $userMaj ; }

1. Création des getter et setter des entity précédemment créés

**php bin/console make:entity --regenerate App**

RQ – ajouter l’Option –overwrite si il faut recréer l’ensemble des getter et setter

RQ - si une erreur de mémoire “Fatal error: Allowed memory size of …” arrive , il faut alouer plus de mémoire à PHP dans le php.ini (ex. : memory\_limit=512M, realpath\_cache\_size = 5M) et augmenter si besoin le paramètre : max\_execution\_time (ex. =300)

RQ – vérifier dans le cas des Id de type String non autoincremental que le Setter existe, le rajouter sinon :  
 **public function setId(?string $id): self**

**{**

**$this->id = $id;**

**return $this;**

**}**

## 3d- Ajout à la BBD la table « user\_db »

Lancer le script de création de la table user\_db (dans le dossier /docs) :   
PROCEDURE\_ADD\_USER\_DB.sql

RQ ! pour une base postgres la bdd et la table user\_db doivent être dans le schéma /public. Si ce n’est pas le cas le script de création et l’entité User.php sont à ajuster

La procédure ajoute une table user\_db à la BDD et charge une compte admin par défaut :  
login = admin / password = adminInBORe

RQ ! si l’information sur les utilisateurs de l’interface (user\_db) doit être enrichie par d’autres informations comme par ex. le(s) Site(s) qu’il gère, ou une information plus détaillée (table Person) il faut ajouter autant de tables relationnelles de type N-N ou 1-1 pour faire le lien vers ces tables spécifiques au projet. Le formulaire UserType relatif à la table user\_db sera à ajuster pour inclure les EMBED-FORM des tables à lier (par ex. UserSiteEmbedType , UserPersonEmbedType )

## 3e- Langage(s) & bilinguisme de l’interface

L’interface InBORe intègre la gestion bilingue de l’interface. Les traductions relatives aux éléments de l’interface ou des messages d’aide contextuel se trouvent dans le dossier /translations  
L’intitialisation des langues de l’interface nécessite d’ajuster 3 paramètres :  
(i) app\_locale : config/service.yaml – code ISO 3166 des langues  
**parameters:  
 app\_locales: *en|fr***(ii)default\_local & fallbacks : config/packages/translations.yaml  
 **default\_locale: *fr***  
et   
 **translator:**  
 **fallbacks :  
 - *fr***(iii) second\_language : templates/base.html.twig  
**{% set second\_language = '*en*' %}**

RQ - Dans le cas d’une interface monolingue (ex. de l’Anglais : ‘en’) , il faut ajuster les 4 paramètres comme suit :  
(i) **app\_locales: *en***(ii)**default\_locale: *en*  et fallback : - *en***(iii)**{% set second\_language = '' %}**

## 3f- Corriger les erreurs et les contraintes des Entités et de de la BDD

Lancer la commande qui montre les requetes SQL d’update sur la base (cf. /migrations/VersionX.php)**php bin/console make:migration**

RQ- le make :migration ajoute un contrôle sur la génération des séquences. Pour s’en abstraire il est toujours possible d’utiliser l’ancienne commande :  
**php bin/console doctrine:schema:update --dump-sql 1> dump-sql.txt**

Et adapter les contraintes manquantes qui ne sont pas généré automatiquement comme :  
  
(i) les contrainte CASCADE :  
- Entity/EntityN1N2.php

*\* @var \EntityN1*

*\**

*\* @ORM\ManyToOne(targetEntity=" EntityN1", inversedBy="* entityN1N2*s")*

*\* @ORM\JoinColumns({*

*\* @ORM\JoinColumn(name=" entityN1\_fk", referencedColumnName="id",* ***onDelete="CASCADE")***

*\* })*

*\*/  
 private $entityN1Fk;*

Rq – l’absence de la declaration de l’Action (ex. onDelete) au niveau de l’Entite génère un DROP CONSTRAINT + ADD CONSTRAINT FK\_\* sur la base

(ii) Ajuster (si necessaire) les contraintes NOT NULL sur les jointures des clès étrangère :   
Exemple :

\* @ORM\JoinColumns({

\* @ORM\JoinColumn(name=" *entityN1\_fk* ", referencedColumnName="id", **nullable=false**)

\* })

(iii) Ajouter les valeurs par defaut   
Exemple :

\* @ORM\JoinColumns({

\* @ORM\JoinColumn(… **options={"default":val\_default })**

\* })

(iv) Et corriger les erreurs possibles au niveau de la bdd ; erreur de contrainte, clès primaire, sequence …

relancer la commande et adapter la base de données en consequence. Reitérer la commande m:m et les modifications au niveau de la bdd autant de fois que necessaire:  
**php bin/console make:migration**

RQ – si les modifications sur la base de donnée le permette on peux appliquer les modification directement via la commande : **php bin/console doctrine:migration:migrate**

# 4 - Génération et mise à jour des formulaires CRUD / Entity

## 4a – génération des formulaires CRUD

Génération des formulaires CRUD pour chacune des tables. Elle va générer :

* le Repository  
  Cf. /Maker/Resources/skeleton/crud/repository/InboreRepository.tpl.php
* les vues twig (index, edit, new, show) dans un dossier « NomEntity » situé dans /templates  
  Cf. /Maker/Resources/skeleton/templates/edit.tpl.php & index.tpl.php
* les fichiers php de déclaration et construction des formulaires « NomEntity »Type.php situé dans un dossier /Form  
  Cf. /Maker/Resources/skeleton/form/Type.tpl.php
* le Controller associé à la table/entity : « NomEntity » Controller.php situé dans un dossier /Controller  
  Cf. /Maker/Resources/skeleton/crud/controller/InboreController.tpl.php

RQ  - Cette étape est à réitérer pour chacune des entités créées précédemment, exeption faite des tables relationnelles de type N-N (cf. §5a) .

RQ ! – Cette étape n’est à faire qu’une fois en début de projet et nécessite que les fichiers des dossiers /Controller, /Form & /templates pour l’Entité n’existent pas.

**php bin\console make:inbore-crud *NameEntity* --no-debug**  
(ex. *php bin/console make:inbore-crud Test --no-debug*)

RQ ! – respecter la casse (Majuscule/minuscule) pour le Nom-Du-Bundle (facultatif) et le NomEntity

RQ – l’Option –no-debug est optionnel mais est utile si il y a des warning qui s’affichent

## 4b – ajuster les Repository

1. Ajuster les 3 constantes**:**

// nom du champ ou expression à afficher pour le champ de recherche dans bootgrid

**const BOOTGRID\_SEARCH\_COLUMN = *'name\_or\_expression\_of\_the\_bootgrid\_search\_column'*;** RQ - ex. d’expression : 'CONCAT(*entity1.field1*,\':\', *entity2.field2*)';

// Name (Database) of the Autocomplete Field for Select2

**const DBNAME\_FIELD\_TO\_AUTOCOMPLETE = *'name\_of\_select2\_autocomplete\_field'*;**

// Max number of result to show in the auto-complete list (default = 20)

**const MAX\_RESULTS\_TO\_AUTOCOMPLETE = 20;**

1. Compléter la liste des champs à afficher dans la liste BootgridIl faut compléter les deux parties :// specific  
   Ex. Cas d’un champ de type time ou date . Il doit être convertit en String  
   **'CONCAT(\'\', entity.time, \'\')' => *'entity\_time’*,**Ex1. Cas d’un champ d’une table liée EntityRel  
   ***'entityRel.fieldName'* => *'entityRel\_fieldName'*,**RQ ! il faut également renseigner les points d) jointures et e) reduction  
     
   // aggregation  
   Ex2.   
   **'STRING\_AGG(entityRel.fieldName, \';\')' => 'list\_entityRel,**  
   RQ ! il faut également renseigner les points d) jointures et e) reduction  
     
   // links : liens vers la liste des N enregistrements d’une entités liée   
   Ex3. Cas d’un lien vers une liste d’enregistrement liés de l’EntityRel  
    **'CASE '.  
    'WHEN MAX(entityRel.id) IS NOT NULL '.  
    'THEN CONCAT(\'\',entity.id,\'\') '.  
    'ELSE \'\' '.  
    'END' => 'link\_entity\_rel',**
2. orderBy
3. jointure

// join entities   
Ex1.  
**->leftJoin(*'entity.fk\_entityRel ', 'entityRel*)**

Ex2. Cas d’un champ Aggregation  
**->leftJoin(‘entity.fk\_ entityRel, 'entityRel')**

1. reductionEx2. **->addGroupBy(*'entityRel\_fieldName'*)**

## 4c – ajuster les formulaires

1. Les Champs relationnel N-1 : clés étrangère

RQ  -  Les fichiers de déclaration des formulaires « NomEntity »Type.php sont par la suite à compléter ; les clés étrangères sont à préciser, il faut préciser dans les arguments de la fonction add() qu’il s’agit d’un champ qui pointe vers une entité ( 2eme argument : ‘entity’) et préciser dans un tableau 1) la class de l’entité 2) le nom du champ à afficher ( clef ‘property’) 3) des informations facultatives sur le type d’affichage (liste déroulante, case à cocher , sélection multiple etc.). exemple :  
(i) déclarer en haut de page les uses :  
**use App\Entity\*NomEntity*;  
use Symfony\Bridge\Doctrine\Form\Type\EntityType;**

(ii) Ajouter l’éntité liée // EntityType : Foreign key fields ex *EntityN1Fk*  
 **->add(*'EntityN1Fk'*, EntityType::class, array(  
 'class' => 'App\\Entity\\*EntityN1'*,  
 'query\_builder' => function (EntityRepository $er) {  
 return $er->createQueryBuilder(*'entityN1'*)  
 ->orderBy(*'entityN1.nomFieldToOrder'*, 'ASC');  
 },  
 'placeholder' => *'Choose a entityN1.nomFieldToOrder label'*,  
 'choice\_label' => *'nom\_field\_to\_order*', // nom\_field\_to\_order : name of field in the database  
 'multiple' => false,  
 'expanded' => false,  
 ))**

RQ -A vérifier qu’il y a l’appel du EntityRepository : **use Doctrine\ORM\EntityRepository;**

RQ – les clés étrangères et les informations relatives à la table liée ne sont pas affichées dans la vue twig générée par Doctrine (template show.html.twig) et doivent être ajoutées si on le souhaite ( via la syntaxe twig {{ *entity.idFk.nomAttribut* }} , avec *entity* et *idFk* le nom de l’entité et de la clés étrangère utilisée par Doctrine)

1. Les Champs Autocomplete

// SearchableSelectType : Auto complete fields from linked\_entityFk :   
// see /assets/Core/forms/InBORe\_entity-name.js for js example  
**use App\Form\Type\SearchableSelectType;**  
…  
 **->add('linked-entityFk', SearchableSelectType::class, [  
 'class' => 'App\\Entity\\*Linked-entity'*,  
 'choice\_label' => '** *SymfNameFieldToAutocomplete****'*,  
 'placeholder' => $this->translator->trans("*Linked-entity typeahead placeholder*"),  
 'attr' => [  
 "maxlength" => "255",  
 'readonly' => ($options['action\_type'] == Action::create->value && $relativeRecord->getLinked-entityFk()),  
 ],])**

RQ ! – les champs autocomplete font appel à la fonction findSearch() définit dans le Repository de l’entité associée (par ex. NomEntity) . Deux paramètres sont à ajuster dans NomEntityRepository :  
(i) DBNAME\_FIELD\_TO\_AUTOCOMPLETE : nom du champ (bdd) interrogé   
(ii) MAX\_RESULTS\_TO\_AUTOCOMPLETE : nombre maximum de résultat affiché par le champ autocomplete

RQ – ‘*SymfNameFieldToAutocomplete****'***: Le nom du champ de l’Entité correspondant au champ de la bdd DBNAME\_FIELD\_TO\_AUTOCOMPLETE

RQ – l’implémentation d’un champ autocomplete nécessite par ailleurs :  
(i) création/modification de l’entity-form.js : appel à la fonction initSearchSelect()  
(ii) ajout au fichier webpack.config.js de l’appel : .addEntry('entity-form', './assets/Core/forms/js/entity-form.js')  
(iii) l’appel du script dans entity/edit.html.twig : {{ encore\_entry\_script\_tags("prelevement-form") }}

1. Les Champs Date du type JJ-MM-AAAA

**use App\Form\Type\DateFormattedType;  
…**  
 **->add(*'nameOfdateField'*, DateFormattedType::class)**

1. Champs Date du type BaseVocabulary  
   **use App\Form\Type\BaseVocabularyType;  
   …**

**->add('parentCodeVocFk', BaseVocabularyType::class, [**

**'voc\_parent' => *'parent\_code'*,**

**'placeholder' => *'Choose a Parent’*,**

**])**

1. Champ imbriqué à valeur multiples (N-N)Ajout d’un champ de type Collection (N-N) . Cf. §5.a

**use Symfony\Component\Form\Extension\Core\Type\CollectionType;**

**…  
use App\Form\EmbedTypes\ *EntityRelN-NEmbedType*;**

**…** // EmbedType de type N-N ; entityRelN-Ns : array Collection defined in the Entity **->add(*'entityRelN-Ns'*, CollectionType::class, [  
 'entry\_type' => *EntityRelN-NEmbedType*::class,  
 'allow\_add' => true,  
 'allow\_delete' => true,  
 'prototype' => true,  
 'prototype\_name' => '\_\_name\_\_',  
 'by\_reference' => false,  
 'entry\_options' => [  
 'label' => false,  
 ],  
 ])**

1. Champ imbriqué à valeur multiples (N-N) avec création d’enregistrement « on-fly » par modaleAjout d’un champ de type Collection (N-N) . Cf. §5.a

**use Symfony\Component\Form\Extension\Core\Type\CollectionType;  
use App\Form\EmbedTypes\ *EntityRelN-NEmbedType*;**

**…**// EmbedType N-N with modal windows to create on-fly record ; entityRelN-Ns : array Collection defined in the Entity  
 **->add('entityRelN-Ns', CollectionType::class, [  
 'entry\_type' => *EntityRelN-NEmbedType*::class,  
 'allow\_add' => true,  
 'allow\_delete' => true,  
 'prototype' => true,  
 'prototype\_name' => '\_\_name\_\_',  
 'by\_reference' => false,  
 'attr' => [  
 "data-allow-new" => true,  
 "data-modal-controller" => 'App\\Controller\\Core\\*EntityRelN1Controller*::newmodalAction',  
 "choice\_label" => '*NAME\_OF\_Choice\_label\_USED\_IN\_EntityRelN-NEmbedType'*,  
 ],  
 'entry\_options' => [  
 'label' => false,  
 ],  
 ])**

1. Champ de type N-1 avec création d’enregistrement « on-fly » par modale  
     
   (i) ajout du bouton dans le formulaire via le ModalButonType   
   **->add('*nomDuBouton*', ModalButtonType::class, [  
    'label' => 'button. *nomDuBouton*,  
    'icon\_class' => 'fa-plus-circle',  
    'attr' => [  
    'class' => "btn-info btn-sm",  
    "data-modal-controller" => 'App\\Controller\\Core\\*EntityRelController*::newmodalAction',  
    ], ])**  
     
   (ii) ajout du code js dans entity-form.js   
    **import { modalFormSubmitCallback } from "./forms"  
    const $entity= $("#entity\_idEntityRel")  
    // EntityRel modal**

**const modalId = "#modal-suivi\_echouage\_ *nomDuBouton* "**

**const $modal = $(modalId)**

**$modal.find("form").off("submit").submit(function (event) {**

**event.preventDefault()**

**modalFormSubmitCallback(event, modalCallback)**

**})**

**function modalCallback(\_, response) {**

**$entity .append($('<option>', {**

**value: response.select\_id,**

**text: *response.select\_id* …**

**}))**

**.val(response.select\_id)**

**.selectpicker('refresh')**

**}**

1. Champ pour la gestion de fichier lié

use App\Form\Type\FileUploaderType;

...

->add('pdf', FileUploaderType::class, [

**'mapped' => false,** // champ non mappé

**'required' => false,**

**'data' => $builder→getData(),** // l’objet concerné

**'action\_type' => $options['action\_type'],**

'file\_path' => 'protocol\_file', // nom du path dans le controleur pour l’accès supervisé au fichier

'mime\_types' => ['application/pdf', 'application/x-pdf'], // types mimes acceptés

'accept' => '.pdf' // types de fichiers sélectionnable en front

'max\_size' => '4096k', // taille maximale autorisée

'setters' => [‘width’ => ‘setWidth’, ‘height’ => ‘setHeight’], // méthodes de l’objet

'ignore\_field\_name' => false // retro-compatibilité si fichier unique

]);

1. Autres type de champs spéciaux :
2. BaseVocabularyType : affichage de la liste de vocabulaire (Code : Name) pour un code parent donné   
   RQ – ce champ suppose une table Vocabulary de la forme {id, Parent, Code, Name} . Id un identifiant unique de type String
3. BooleanType : champ de type Booleen / case à cocher
4. DateFormattedType : champ de type date au format : JJ/MM/YYYY  
   RQ – DatePrecisionType (J/M/Y) si on veut préciser et controler la Precision dela date donnée
5. EntityCodeType : pour des champs de type Code alphanumérique
6. FloatType : champ de type float
7. HTMLType : champ de contenu HTML
8. JSONType : champ de type json

## 4c – ajuster les templates TWIG ; index.html.twig, edit.html.twig

1. Entity/index.html.twig  
     
   Deux paramètres peuvent être ajustés si necessaire ;   
   {# initialize the maximum number of fields list to show #}

**{% set max\_fields\_visible = 5 %}**  
RQ – augmenter la valeur si il y a besoin d’afficher plus de 5 colonnes par défaut dans la liste des enregistrements

{# set the search field column to show with an asterix #}

**{% set search\_field\_column = *'entity*\_bootgrid\_search\_column' %}**  
RQ – le nom du champ *'entity*\_bootgrid\_search\_column' correspond à celui définit dans *EntityRepository*.php

{% block action %} :  
Ajouter les boutons relatifs à de nouvelles actions ou pour modifier les boutons par défault (cf. /templates/index.html.twig)

1. Entity/edit.html.twig  
     
   {% block action %} :  
   Ajouter les boutons relatifs à de nouvelles actions ou pour modifier les boutons par défault.

{% block scripts %}  
Pour ajouter des comportements javascript il faut :  
 (i) créer le fichier *entity*-form.js dans /asset/Core/forms/js/

(ii) ajouter l’appel dans le fichier de configuration webpack.config.js :

**.addEntry(*'entity*-form', './assets/Core/forms/js/*entity*-form.js'**)

(iii ) ajouter l’appel TWIG : **{{ encore\_entry\_script\_tags("*entity*-form") }}**

## 4d – ajuster les versions bilingues des ; menus , libellés et mentions légales

Modifier et adapter la(les) version(s) française (.fr) et si l’interface est bilingue Englaise (.en):

* /config/package/twig.yaml  
  définition des libellés, routes, droit d’accès des menus & sous menus  (ex. text : core.nom\_menu)
* /translations/sections.fr.yml   
  traductions des menus & sous-menus définit dans le fichier de configuration twig.yaml
* /translations/messages.fr.yml

Traductions des éléments de l’interface (boutons, listes, …) et des libellées des champs de la bdd

* /translations/help.fr.yml   
  Traductions des messages d’aides
* /templates/misc/legal-notices.fr.html  
  templates des mentions légales

# 5 - Modifications/Evolutions des entités & formulaires

## 5a – Ajout d’une Collection (relation 1-N ou N-N)

Pour ajouter une collection EN1-N2 (relation N-N ) ou EN2 (relation 1-N) a une entité EN1 il faut :

1. Ajuster l’entité EN1 pour laquelle on souhaite ajouter la Collection  (ex. en1n2s ) :

**/\*\***

**\* @ORM\OneToMany(targetEntity=*EN1N2*::class, mappedBy="*idEN1*", cascade={"persist"}, orphanRemoval=true)**

**\* @ORM\OrderBy({"id" = "ASC"})**

**\*/**

**private $*en1n2s*;**

1. Créer les getter et setter et le ArrayCollection()

**# php bin/console make:entity --regenerate  
# App\Entity\*EN1***

1. Ajuster la relation inverse ManyToOne de l’entité EN1-N2 (ou EN2)

/\*\*

\* @var\*EN1*

\*

\* @ORM\ManyToOne(targetEntity=" *EN1*", **inversedBy=" *en1n2s* ")**

\* @ORM\JoinColumns({

\* @ORM\JoinColumn(name="*en1\_fk*", referencedColumnName="*iden1*", nullable=false, **onDelete ="CASCADE"**)

\* })

\*/

private $idEN1;

1. Générer et adapter le formulaire à imbriquer
2. générer le formulaire EN1NEmbedType (dans /Form/EmbedTypes/)

**# php bin/console make:inbore-embed *EN1N2* --no-debug**

1. Supprimer l’affichage de la foreignkey (idEN1) du formulaire EN1N2EmbedType et adapter le contenu
2. Ajouter l’appel du formulaire EN1NEmbedType dans le formulaire EN1

**use App\Form\EmbedTypes\ EN1NEmbedType ;**

…

**->add(*'en1n2s'*, CollectionType::class, array(**

**'entry\_type' => *EN1N2EmbedType*::class,**

**'allow\_add' => true,**

**'allow\_delete' => true,**

**'prototype' => true,**

**'prototype\_name' => '\_\_name\_\_',**

**'by\_reference' => false,**

**'entry\_options' => array('label' => false),**

**'required' => true,**

**))**

## 5b – Ajout d’un attribut et MAJ du schéma de bdd

Pour ajouter ou modifier des entités, comme ajouter un nouvel attribut. Il faut suivre les étapes suivantes :

1. Effectuer les modifications du fichier /Entity/NomEntiteAModifier.php

Ex. Ajout de l’attribut objectif : declaration de   
 /\*\*

\* @var string

\*

\* @ORM\Column(name="*objectif*", type="string", length=255, nullable=true)

\*/

private $*objectif*;RQ - il est possible de le créer de manière interactive via la console :   
**php bin/console make:entity**

1. Lancer la commande qui montre la commande SQL d’update de la base

**php bin/console make:migration**Ex. on obtient la commande SQL suivante dans encapsulé dans un fichier Version\*.php dans /migrations :  
*ALTER TABLE referentiel\_taxon ADD objectif VARCHAR(255) DEFAULT NULL ;*

1. Si la commande parait correcte on applique les modifications au niveau de la bdd :  
   **php bin/console doctrine:migrations:migrate**
2. Si il y a eu ajout d’un attribut ajouter les getter et setter : **php bin/console make:entity --regenerate App\Entity\NomEntiteAModifier**
3. Si besoin, dans le cas d’une clé étrangère. Il y a lieu de rajouter la mention de l’index créer sur la clé étrangère. Par ex. ajouter dans @ORM\Table … indexes = { @ORM\Index(name="IDX\_801C3911B669F53D", columns={"nom\_cles\_etrangere\_ajouter\_fk"}), …}   
   RQ – ici le nom IDX\_801C3911B669F53D est généré automatiquement par doctrine via la comande ((ii)

## 5c – Ajout contrainte UNIQUE et MAJ du schéma de bdd

Pour ajouter ou modifier des entités, comme ajouter une contrainte UNIQUE. Il faut suivre les étapes suivantes :

1. RQ - facultatif : A FAIRE si l’on souhaite choisir un nom de contrainte au niveau de la bdd :  
   Effectuer le(s) modification(s) du fichier /Entity/NomEntiteAModifier.php  
   EX. Ajouter dans l’entete de la class de l’Entity la déclaration de l’unicité pour un nom de contrainte choisie (par ex. cu\_lot\_materiel\_ext\_code\_lot\_materiel\_ext) :  
   *\* @ORM\Table(name="lot\_materiel\_ext", uniqueConstraints={@ORM\UniqueConstraint(name="cu\_lot\_materiel\_ext\_code\_lot\_materiel\_ext", columns={"code\_lot\_materiel\_ext"})}, … )*
2. Effectuer le(s) modification(s) du fichier /Entity/NomEntiteAModifier.php . Ajouter la mention *unique=true* au niveau de la déclaration du(es) champ(s) UNIQUE de l’Entity . Par exemple :  
   *\* @ORM\Column(name="code\_lot\_materiel\_ext", type="string", length=255, nullable=false, unique=true)*
3. Lancer la commande qui montre la commande SQL d’update de la base

**php bin/console make:migration**Ex. . on obtient la commande SQL suivante :

*CREATE UNIQUE INDEX cu\_lot\_materiel\_ext\_code\_lot\_materiel\_ext ON lot\_materiel\_ext (ccode\_lot\_materiel\_ext)*

1. Si la commande parait correcte on applique les modifications au niveau de la bdd :  
   **php bin/console doctrine:migrations:migrate**

# 6– Gestion centralisée des assets (js, css) avec Encore

La gestion des assets est géré par le bundle webpack-encore-bundle de Symfony : <https://symfony.com/doc/4.4/frontend/encore/installation.html>  
pour pouvoir gérer de manière centralisée l’ensemble des assets et beneficier d’un « composer » (yarn) qui facilitera la mise à jour et la cohérence des versions et dépendances.   
RQ – l’installation de webpack (# php -d memorycomposer require symfony/webpack-encore-bundle) crée :  
- un fichier webpack.config.js ; fichier de configuration générale qui permet de spécifier les assets à charger en fonction des différentes Pages (Entry) de l’application  
- un repertoire assets/avec les fichiers appelé par webpack.config.js

## 6a- Configuration du WebpackEncoreBundle

Cf. doc <https://symfonycasts.com/screencast/webpack-encore/first-webpack>

* Configuration de Encore via le fichier webpack.config.js (cf. racine du projet Symfony)

*// directory where compiled assets will be stored*

***.setOutputPath('public/build/')***

*// public path used by the web server to access the output path*

***.setPublicPath('/build')***

*// insertion des assets pour l’application app*

***.addEntry('app', './assets/js/app.js')***

* Construction des assets via les commandes :  
    
  Pour compiler les assets une fois (mode manuel)  
  **# yarn encore dev**

Ou pour recompiler les assets (mode automatique) dès qu’un fichier d’un asset est modifié   
**# yarn encore dev --watch**RQ – pour sortir de mode --watch : Ctrl+C

Pour produire la version de production

**# yarn encore production**RQ - ces commandes génère des fichiers js et css dans un dosiser public/build/ qui seront exploités par l’application web en dev ou production

* Pour optimiser le développement, Installer un serveur de ressources local (encore dev-server) pour les asset (js, css, images…) qui évite d’utiliser « encore dev—watch » avec des réécriture des fichiers à chaque modif  :

Pour travailler en https (si besoin), installer un certificat HTTPS   
**# PATH\_TO\_SYMFONY\_CLI/symfony server:ca:install**RQ – le client Symfony et l’executable symfony.exe à downloader via <https://symfony.com/download>

Lancer le serveur de ressource (webpack-dev-server) avant le développement : **# yarn encore dev-server --disable-host-check --port 8080 --https --pfx=%UserProfile%\\.symfony\\certs\\default.p12"**ou en mode http **# yarn encore dev-server --disable-host-check –port**

Pour arreter le serveur de ressources **# CTRL+C**

RQ – le serveur de ressources des assets qui fait tourner Webpack (via Symfony Encore) est indépendant du serveur web de Symfony. Il n’y a donc pas lieu de lancer la commande symfony start :server au préalable  
RQ – on peut ajouter des racourcis aux commandes dans le fichier .packages.js (section scripts), par exemple :   
"dev-server-win": "encore dev-server --disable-host-check --port 8080 --https --pfx=%UserProfile%\\.symfony\\certs\\default.p12",

* Ajouter dans le template base.html.twig les appels aux assets crées par Encore :

*{# templates/base.html.twig #}*

<head>

<!-- ... -->

{% block stylesheets %}

{# 'app' must match the first argument to addEntry() in webpack.config.js #}

**{{ encore\_entry\_link\_tags('app') }}**

<!-- Renders a link tag (if your module requires any CSS) <link rel="stylesheet" href="/build/app.css"> -->  
 {% endblock %}

</head>

<body>

<!-- ... -->

{% block javascripts %}

**{{ encore\_entry\_script\_tags('app') }}**

<!-- Renders app.js & a webpack runtime.js file -->

{% endblock %}

</body>

## 6b- Ajout de modules Javascript

Deux étapes sont à suivre pour ajouter un nouveau module javascript via Encore (exemple pour le module jquery) :

- Installer le module via le gestionnaire de package yarn    
RQ – le package js sera downloadé dans le dossier /node\_module  
**# yarn add *jquery* –dev**  
- ajouter la ligne qui explicite l’import du module dans le fichier de config (cf. « addEntry » dans webpack.config.js)  :  
// loads the jquery package from node\_modules  
**var $ = require(*'jquery'*);**ou **import $ from 'jquery';**

## 6c- Composants et librairies js

1. FosJsRoutingBundle : expose les routes Symfony pour les développements js  
   cf <https://github.com/FriendsOfSymfony/FOSJsRoutingBundle>
2. BazingaJsTranslationBundle: expose les traductions Symfony pour les développements js  
   cf. <https://github.com/willdurand/BazingaJsTranslationBundle/blob/master/Resources/doc/index.md>
3. Jquery, bootstrap, fontawesome, select2, bootgrid, mustache, moment

# 7 – Bundle Symfony – Usage & Déploiement

Ajout de Bundle au projet InBORe

## 7a – Info sur le bundle FOSJSRoutingBundle 2.x

* download et installation du bundle  
  cf. <https://symfony.com/doc/2.x/bundles/FOSJsRoutingBundle/installation.html>

cf. https://github.com/FriendsOfSymfony/FOSJsRoutingBundle/blob/master/Resources/doc/usage.rst

RQ ! ajouter un fichier fos\_js\_routing.yaml dans /config/packages/ comprenant :

*fos\_js\_routing:*

*routes\_to\_expose: [^.\*$]*

* ajout des appels des scripts relatifs au bundle dans base.html.twig (block script)  
  **<script src="{{ asset('bundles/fosjsrouting/js/router.min.js') }}"></script>**

**<script src="{{ path('fos\_js\_routing\_js', { callback: 'fos.Router.setData' }) }}"></script>**

# 8 – Extension du projet InBORe : Services – Usage & Description

Ajout de nouveaux services dans le projet InBORe

## 8a – Service FileLoader

Le service FileLoader permet de gérer le download de fichiers stocké dans un dossier à la racine du serveur web.  
Il permet de gérer via un formulaire (i) 1/N fichier(s) media (fichier binaire, image, audio, video) sans métadonnées associées (ii) N fichiers multimedia avec des métadonnées associées qui sont référencés et gérés via la table Media .  
  
**(1) cas d’un champ de formulaire / 1 fichier d’un format multimedia prédéfinit**   
 Les fichiers sont stockés dans un dossier /files/nom\_entite/numero\_incremental/nom\_du\_champ  
Le nom\_entite correspond à l’entité auquel est attaché le formulaire qui contient le champ d’upload.   
La configuration de se service s’effectue directement dans la définition du champ dans le formulaire:

**->add(**'*nom\_champ\_upload'***, FileUploaderType::class, [**

**'mapped' => false,**

# il s’agit d’un fichier lié, le champ n’est pas une propriété de l’entité

**'required' => false,**

# au choix

**'data' => $builder→getData(),**

# l’objet concerné, nécessaire pour la determination du chemin de stockage du fichier

**'action\_type' => $options['action\_type'],**

# repport de l’action courante

'file\_path' => 'nom\_du\_path',

# Optionnel: force le nom du path dans le controleur pour l’accès supervisé au fichier, ex: ‘protocol\_file\_special’

**'mime\_types' => [***'mime\_1', …, 'mime\_n’***]**

# types mimes acceptés (côté serveur), ex1 fichier : ['application/x-pdf', ‘application/zip’, ‘text/csv’],  
# ex2 images : ['image/gif', 'image/png', 'image/jpeg'], ex3 audio : ['audio/wav', 'audio/mpeg', 'audio/x-wav']

**'accept' => '***.ext1, …, .extn'*

# types de fichiers sélectionnable en front, ex1 : ‘.pdf,.zip,.csv ‘, ex2 : '.gif,.png,.jpeg', ex3 : '.wav,.mpeg'

**'max\_size' =>** *'4096k'***,**

# taille maximale autorisée, ex : 4096k

'setters' => [‘property’ => ‘setter’, ...],

# propriétés > méthodes de l’objet, ex : [‘width’ => ‘setWidth’, ‘height’ => ‘setHeight’], permet de remplir les champs correspondant de l’objet définit dans la table Media (cf. gestion multi-fichier)

'ignore\_field\_name' => false

# pour retro-compatibilité dans le cas où le fichier est dans le dossier de l’objet et pas le sous-dossier au nom du champ.

**]);**

**(2) cas de la gestion multi-fichiers multimedia multi-format**

L’implémentation de ce service fait appel à l’entité Media et aux formulaires Core/MediaType et EmbedTypes/MediaLinkEmbedType. Pour utiliser ce service il faut:

1. Copier les fichiers du dossier /docs/FileUpLoader dans l’application :/Media.php -> /entity/Media.php

RQ – adapter l’annotation @ORM\Table(name="media") si besoin   
/MediaLinkEmbedType.php -> /Form/EmbedType/MediaLinkEmbedType.php

**php bin\console make:inbore-crud Media --no-debug**

1. Adapter /Form/MediaType.php

Ajouter l’appel au composant FileUploaderType (cf cas (1) ci-dessus)  
Rendre en ReadOnly les paramètres ; path, filename, mimeType, siez, width, height :  
ex. *->add('filename', null, ['attr' => ['readonly'=>'readonly']])*

1. Ajuster /Repository/Core/MediaRepository.php

Mettre ‘filename’ pour le les constantes BOOTGRID\_SEARCH\_COLUMN et DBNAME\_FIELD\_TO\_AUTOCOMPLETE  
et ajouter dans la section // specific du tableau $fields   
'media.filename' => 'filename',  
'media.name' => 'name',  
…

1. Ajuster les Entités Media et EntiteLieAuxMedia  
   Ajouter dans Media :  
   **/\*\***

**\* @ORM\ManyToMany(targetEntity=*EntiteLieAuxMedia*::class, mappedBy="medialinks")**

**\*/**

**private $medias;**

Ajouter dans EntiteLieAuxMedia si nécessaire (N-N bidirectionnelle) :  
**/\*\***

**\* @ORM\ManyToMany(targetEntity=Media::class, inversedBy="medias")**

**\*/**

**private $medialinks**

1. Mettre à jour les setter et getter des Media et EntiteLieAuxMedia **# php bin/console make:entity --regenerate App\Entity\Media  
   # php bin/console make:entity --regenerate App\Entity\*EntiteLieAuxMedia***
2. Ajouter la table Media à la bdd . Lancer la commande qui montre la commande SQL d’update de la base  
   **# php bin/console make:migration**  
    (ii) Si la commande parait correcte on applique les modifications au niveau de la bdd :  
   **# php bin/console doctrine:migrations:migrate**
3. Ajouer L’appel à la Collections medialinks dans le formulaire de l’entité *EntiteLieAuxMedia*

**use App\Form\EmbedTypes\MediaLinkEmbedType;  
…**

**// mediallinks**

**->add('medialinks', CollectionType::class, array(**

**'entry\_type' => MediaLinkEmbedType::class,**

**'allow\_add' => true,**

**'allow\_delete' => true,**

**'prototype' => true,**

**'prototype\_name' => '\_\_name\_\_',**

**'by\_reference' => false,**

**'entry\_options' => array('label' => false),**

**'required' => false,**

**'attr' => [**

**"data-allow-new" => true,**

**"data-modal-controller" => 'App\\Controller\\Core\\MediaController::newmodalAction',**

**"entity" => 'media',**

**"choice\_label" => 'filename',**

**],**

**))**

Ajouter le menu au Media (cf. config/packages/twig.yaml)

# Annexe 1 – Architecture logicielle – InBORe

|  |  |  |
| --- | --- | --- |
| **dossier** | **fichier** | **commentaire** |
| /assets |  | ensemble des assets du projet (js, css, images, …) |
| /assets/Core |  | ensemble des assets du cœur InBORe (/ js & /css ) et des formulaires /forms |
| /assets/Core/css |  | Fichiers .css et .less du Core InBORe (navbars, core, custom, dashboard, login, tables) |
| /assets/Core/forms |  | Fichiers .js, .css , .less relatifs au formulaires du projet |
| /assets/Core/forms/ components |  | fichiers .vue |
| /assets/Core/forms/css |  | Css general aux formulaires |
| /assets/Core/forms/js |  | Javascript attachés aux formulaires. |
|  | InBORe\_entity-name.js | modèle généraliste / formulaire à modifier/compléter  RQ ! les fichiers InBORe\_entity-name.js devront être appelé et ajouté dans webpack.config.js |
|  | date-mask.js | contrôle des format d’entrée pour les champs de type {date/date-precision} |
|  | field-suggestions.js | fichier d’inittialisation des champs autocomplete Select2  RQ ! utilise le bundle FOSJsRoutingBundle (gestion des routes Symfony dans du Js) => **AV** – implémentation du bundle / plugin webpack FosRouting (cf. core.js, base.html.twig, field-suggestions.js) <https://github.com/FriendsOfSymfony/FOSJsRoutingBundle/blob/fb188bfe352ccd7ecff06c08eca05fed47f30de4/Resources/doc/usage.rst> |
|  | form-tools.js | Action de type refresh() de l’url lorsqu’un champ d’une liste de formulaire est changée |
|  | forms.js | js du Core ; gestions du prototype pour les collections ; boutons [Add], [Delete] , [add new] via une modal, champs autocomplete (select2) , gestion des champs du type {date/date-precision} (cf. date-mask.js) |
|  | vocabulary-form.js | renommage automatique des id des vocabulaires suivant les règles ; (i) espace => underscore (ii) Tous les caractères en minuscule |
| /assets/Core/js |  | javascripts généraux à InBORe |
|  | core.js | js principal du projet qui appel ; jquery, bootstrap, /nav.js, /options.js. Intègre également la config de bootgrid => **AV** – pour harmoniser mettre la config bootgrid dans /options.js |
|  | dashboard.js | javascript du dashboard (bootgrid) |
|  | index.js | jquery fonction de conversion d’un tableau en fichier csv |
|  | nav.js | évènements et animations pour la barre de navigation (menu/sous-menus ..) |
|  | options.js | fichiers de configuration (plotly, … |
| /assets/components |  | fichiers .vue et affichage Leaflet |
| /assets/directives |  | directive pour fichier vueJs |
| /assets/images |  | images , logos, icones général à l’interface InBOre + autres images à ajuster en fonction du projet |
| /assets/imports |  | template CSV de migration ; vocabulary.csv, taxon.csv. A compléter |
| /assets/styles |  |  |
| /config |  | fichiers de configuration de l’application |
|  | bundle.php | déclaration des bundles associés au projet. Et pour quels modes (all, dev, prod) |
|  | admin.yaml | configuration de paramètre généraux à InBORe RQ – paramètres OBLIGATOIRE : fixe\_fk\_delimiter, prefixe\_fk, suffixe\_fk RQ – initialement prévu pour intégrer les paramètres utilisés dans les Repository (cf. AdminRepository) **ATESTER** – le passage des paramètres ; fixe\_fk\_delimiter, prefixe\_fk, suffixe\_fk dans Services/GenericFunction .php |
|  | route.yaml | configuration de la route par default (dashboard) et du path (/) |
|  | services.yaml | configuration de ; (i) parametres de l’application (app\_locales : en|fr)  (ii) imports (ressources : admin.yaml) (iii) services (\_default : autowire, App\ , maker) |
| /config/packages |  | fichiers de configuration des différents packages et bundle de l’application |
|  | asset.yaml | config du json\_manifest\_path |
|  | cache.yaml | config du cache de Symfony |
|  | doctrine.yaml | config de doctrine (dbal, orm, orm.mapping, orm.dql, …) RQ ! – il existe un bug dans la gestion des alias dans la config doctrine.orm.mappings  <https://github.com/symfony/symfony/issues/44355> par conséquent les racourcis du type App :EntityName ne fonctionne pas et il faut pour le moment faire un appel avec le chemin complet du type App\\Entity\\EntityName |
|  | doctrine\_migrations.yaml | config du module migration de doctrine (migration\_paths, …) |
|  | fos\_js\_routing.yaml | config du bundle FOSJsRouting (routes\_to\_expose) |
|  | framework.yaml | config de Symfony (secret, session, php\_errors, …) |
|  | security.yaml | configuration de la sécurité , des rôles , de la npage de login et d’accès, des firewalls … |
|  | translation.yaml | config de default\_locale (fr) et du dossier ou devront se trouve les traductions |
|  | twig.yaml | Config de twig et menus/sous-menus ( default\_path /templates, app\_name, navbar\_menu\_entries) |
|  | webpack\_encore.yaml | Config de webpack (output\_path, script\_attributes) |
| /src |  | sources de l’application ; Controller/, Entity/ , Form/, Maker/, Query/, Repository/, Services/ Il s’agit du dossier qui doit contenir toutes les Class à charger en autoloading via la directive psr-4 du fichier composer.json (section autoload : "App\\": "src/"). "App" prefixe des Namespaces des Class. RQ ! Le fichier Kernel.php ne doit pas être modifié |
| /src/Controller |  | Controller de l’application |
|  | DefaultController.php | @Route("/legals/", name="legals") : misc/legal-notices.{fr/en}.html.twig |
|  | SecurityController.php | @Route("/login", name="login") , @Route("/logout", name="logout") |
| /src/Controller/Admin | AdminController.php | SuperController @Route("/admin/<entity>, ..., priority=1)  RQ – non utilisé dans la v1 d’InBORe . A conserver en prévision d’une v2 / v3 |
| /src/Controller/Core | DashboardController.php | @Route("/", name="dashboard") ~/Core/dashboard/index.html.twig |
| /src/Controller/User | UserController.php | Controller de l’Entity User (table user\_db) |
| /src/Entity | User.php | Entity de la table user\_db (cf. §3d script de creation de la table user\_db) |
| /src/Form |  | Ensemble des Class FormType génératrice des formulaires |
|  | UserType | formulaire de la Class User |
| /src/Form/ChoiceLoader | AutocompleteChoiceLoader.php | Gestion des choix dynamique des listes via les champs de type autocomplete (cf. Type/SearchableSelectType.php) |
| /src/Form/Admin | AdminType & AdminCollectionType | formulaire associé au supercontroller (inBORe v2) |
| /src/Form/EventListener | AddUserDateFields.php | (i) affecte des valeurs aux champs UserCre, UserMaj, DateCre, DateMaj et affiche quand le formulaire est validé (onSubmit) et (ii) affiche les champs dateCre et dateMaj en mode show (onPreSetData) |
| /src/Form/ DataTransformer | UppercaseTransformer | interface qui implémente DataTransformerInterface avec deux fonctions ; transform() et reverseTransform($text) qui retourne la chaine en Majuscule |
| /src/Form/EmbedTypes |  | dossier pour les formulaires imbriqués |
| /src/Form/Enum | Action.php | definition des listes d’enumerations |
| /src/Form/Type |  | Definition d’Objet prédéfinit pour les formulaires |
|  | ActionFormType & UserDateTraceType | Class générale qui extends la Class UserDateTraceType et charge les composants ; Security, EntityManagerInterface, TranslatorInterface et les Class AddUserDateFields\* , UppercaseTransformer. Elle implémente 3 fonctions ; (i) upperCaseFields(FormBuilderInterface $builder, array $fields) qui pourra s’appeler dans un FormType via : $this->upperCaseFields($builder, $uppercase\_fields); (ii) buildview() qui expose la variable action\_type dans les templates (iii) canEditAdminOnly() : retourne en mode edit 1/0 si l’utilisateur est Admin ou pas. A appeler dans un FormType via : $this->canEditAdminOnly($options) (\*) AV - AddUserDateFields : semble redondant avec UserDateTraceType ? |
|  | BaseVocabularyType.php | composant qui affiche la liste des vocabulaire (table Vocabulary {id\*:String, code\* :String, name\* :String, parent\* :String/Vocabulary RQ – cette version n’intègre pas la possibilité de traduction des champs ; Code et Name |
|  | BooleanType.php | champ de type Bolleen (1/0) |
|  | DateFormatedType.php | composant d’affichage des champs date au format dd-MM-yyyy |
|  | DatePrecisionType.php | composant d’affichage des champs de type datePrecision introduit pour préciser s’il s’agit du Jour/Mois/Annee **RQ / AV** – Actuellement ces codes doivent être stockés dans la table Vocabulary |
|  | EntityCodeType.php | estion du style de l’affichage des champs de type ‘Code’ |
|  | FloatType.php | Normalisation de l’affichage des champs de type Float |
|  | GpsBatchType.php & GpsImportType.php | Formulaires Batch & Import de point GPS. |
|  | HTMLType.php | composant pour l’affichage de contenu HTML **AV :** utilité de conserver ce composant ? |
|  | JSONType.php | composant pour l’affichage de champ Postgres de type Json |
|  | ModalButtonType.php | composant pour l’affichage d’une fenetre Modal via un bouton « New .. » |
|  | SearchableSelectType.php | composant pour l’affichage de champ de type autocomplete (Select2) |
| /src/Maker |  | dossier assosié au MakerBundle ; commande make + templates (.tpl.php) |
|  | MakeInboreCrud.php | commande make:inbore-crud |
|  | MakeInboreEmbed.php | commande make:inbore-crud |
|  | MakeInboreRepository.php | commande make:inbore-repository |
| /src/Maker/ Resources/help/ |  | fichier d’aide associé au commande make |
| /Resources/skeleton/crud | InboreController.tpl.php | template Controller d’InBORe |
|  | InboreRepository.tpl.php | template Repository d’InBORe |
| /Resources/skeleton/crud/template |  | template twig edit.tpl.php & index.tpl.php |
| /Resources/skeleton/form | EmbedType.tpl.php | template des formulaire Embed du dossier Form/EmbedTypes |
|  | Type.tpl.php | template des formulaires du dossier Form/ |
| /src/Query |  | Dossier de déclaration de Class et fonction spécifique pour Doctrine |
| /src/Repository |  | Dossier des Repository des Entity. L’appel du Repository s’effectue en tëte de fichier de l’Entity via l’annotation @ORM\Entity(repositoryClass="path\_to\_repository") |
|  | UserRepository.php | Repository de l’Entity User |
| /src/Repository/Core |  | ensemble des Repository des Entity |
| /src/Services |  | Dossier des Services génériques au projet InBORe |
|  | FileUploader.php | service d’Upload de fichier cf §8.a documention de l’implémentation du service |
|  |  |  |
| /src/Services/Core | GenericFunction.php | definition de fonctions globales ; GetNameToSymfony(), GetFkName(), GetUserCreId(), GetUserCreUsername(), GetUserMajUsername(), GetUserCreUserfullname(), GetUserMajUserfullname() |
|  | ImportFileCsv.php | definition de fonctions utiles à l’import de fichiers csv ; getCsvPath(), readCSV(), explodeCSV(), readColumnByTableSV(), testNameColumnCSV(), checkNameCSVfile2Template(), suppCharSpeciaux(), GetCurrentTimestamp() |
| /src/Services/Twig |  | declaration des extensions à Twig |
| /templates |  | templates twig de l’application |
| /translations |  | Fichiers des traductions |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |