

|  |
| --- |
|  |



|  |  |
| --- | --- |
| **EMETTEUR :** | **DIRECTION GENERALE** |
| **DESTINATAIRES :** | **PERSONNELS** |
| **VERSION :** | **1.0 DU …** |

**Guide développeur**

**Muuska FW**

Table des matières

[I. Introduction 13](#_Toc43399410)

[II. Déploiement 14](#_Toc43399411)

[III. Architecture des dossiers 14](#_Toc43399412)

[1. app 14](#_Toc43399413)

[2. public 14](#_Toc43399414)

[3. vendor 15](#_Toc43399415)

[4. storage 15](#_Toc43399416)

[5. Index.php 15](#_Toc43399417)

[IV. Configuration de base 15](#_Toc43399418)

[1. Configuration principale 15](#_Toc43399419)

[a. dao\_source 15](#_Toc43399420)

[b. cookie\_iv 15](#_Toc43399421)

[c. cookie\_key 15](#_Toc43399422)

[d. default\_lang 15](#_Toc43399423)

[e. dev\_mode 15](#_Toc43399424)

[f. app\_installed 15](#_Toc43399425)

[2. Configuration de la base de données 16](#_Toc43399426)

[a. host 16](#_Toc43399427)

[b. user 16](#_Toc43399428)

[c. password 16](#_Toc43399429)

[d. database 16](#_Toc43399430)

[e. table\_prefix 16](#_Toc43399431)

[f. driver 16](#_Toc43399432)

[g. dsn 16](#_Toc43399433)

[3. Configuration des sous applications 16](#_Toc43399434)

[a. url\_path 17](#_Toc43399435)

[b. lang\_in\_url 17](#_Toc43399436)

[c. host 17](#_Toc43399437)

[d. authentification\_required 17](#_Toc43399438)

[e. access\_checking\_required 17](#_Toc43399439)

[V. Code de base 18](#_Toc43399440)

[1. Classe principale de l’application 19](#_Toc43399441)

[a. La methode **registerMainDAOSources** 19](#_Toc43399442)

[b. La méthode createSubProject 19](#_Toc43399443)

[c. La méthode createAppSetup 19](#_Toc43399444)

[2. La classe du setup 19](#_Toc43399445)

[3. La classe de la sous application 20](#_Toc43399446)

[VI. Installation 20](#_Toc43399447)

[VII. Architecture du système 20](#_Toc43399448)

[1. Architecture logique 20](#_Toc43399449)

[a. Le modèle 20](#_Toc43399450)

[b. La vue 20](#_Toc43399451)

[c. Le contrôleur 20](#_Toc43399452)

[d. La persistance ou DAO (Data Access Object) 20](#_Toc43399453)

[2. Schémas global 21](#_Toc43399454)

[a. Requête 21](#_Toc43399455)

[b. Index.php 21](#_Toc43399456)

[c. Application 21](#_Toc43399457)

[d. Routeur 21](#_Toc43399458)

[e. Sous application 21](#_Toc43399459)

[f. Contrôleur 22](#_Toc43399460)

[g. Réponse 22](#_Toc43399461)

[3. Diagramme de séquence du système 22](#_Toc43399462)

[VIII. Contrôleur 24](#_Toc43399463)

[1. Création d’un contrôleur 24](#_Toc43399464)

[2. Url d’un contrôleur 25](#_Toc43399465)

[3. La notion d’action 27](#_Toc43399466)

[4. Controller Input 29](#_Toc43399467)

[getName 29](#_Toc43399468)

[getFullName 30](#_Toc43399469)

[checkName 30](#_Toc43399470)

[checkFullName 30](#_Toc43399471)

[getAction 30](#_Toc43399472)

[getRequestType 30](#_Toc43399473)

[isAjaxRequest 30](#_Toc43399474)

[hasQueryParam 30](#_Toc43399475)

[getQueryParam 31](#_Toc43399476)

[hasPostParam 31](#_Toc43399477)

[getPostParam 31](#_Toc43399478)

[hasParam 31](#_Toc43399479)

[getParam 32](#_Toc43399480)

[getLang 32](#_Toc43399481)

[getLanguageInfo 32](#_Toc43399482)

[getLanguages 33](#_Toc43399483)

[getSubAppName 34](#_Toc43399484)

[getProject 34](#_Toc43399485)

[getSubProject 34](#_Toc43399486)

[getSubApplication 35](#_Toc43399487)

[hasTheme 35](#_Toc43399488)

[getTheme 35](#_Toc43399489)

[getDAO 35](#_Toc43399490)

[getDaoFactory 35](#_Toc43399491)

[createSelectionConfig 35](#_Toc43399492)

[createSaveConfig 35](#_Toc43399493)

[createDeleteConfig 35](#_Toc43399494)

[getRequest 35](#_Toc43399495)

[getResponse 35](#_Toc43399496)

[getVisitorInfoRecorder 35](#_Toc43399497)

[getCurrentUser 36](#_Toc43399498)

[5. Controller result 36](#_Toc43399499)

[getTitle 36](#_Toc43399500)

[setTitle 36](#_Toc43399501)

[setContent 37](#_Toc43399502)

[hasContent 37](#_Toc43399503)

[getContent 37](#_Toc43399504)

[addError 37](#_Toc43399505)

[addErrors 38](#_Toc43399506)

[hasErrors 38](#_Toc43399507)

[addSuccess 38](#_Toc43399508)

[addAlert 38](#_Toc43399509)

[addAlerts 39](#_Toc43399510)

[getAssetSetter 39](#_Toc43399511)

[setRedirection 39](#_Toc43399512)

[hasRedirection 39](#_Toc43399513)

[6. Controller param 39](#_Toc43399514)

[a. Paramètre standard 40](#_Toc43399515)

[b. Paramètre lié à un model 41](#_Toc43399516)

[7. Controller URL creator 42](#_Toc43399517)

[8. Méthodes utile du contrôleur 42](#_Toc43399518)

[IX. Model 42](#_Toc43399519)

[1. Notion de « Model Définition » 45](#_Toc43399520)

[a. Exemple de code 45](#_Toc43399521)

[b. Obtention d’une instance unique 47](#_Toc43399522)

[c. La méthode « createModel » 47](#_Toc43399523)

[d. La méthode «createDefinition» 47](#_Toc43399524)

[e. Définition des associations multiples 49](#_Toc43399525)

[f. Définition d’un champ 49](#_Toc43399526)

[g. Utilisation d’un tableau associatif comme model 53](#_Toc43399527)

[2. Notion de model 55](#_Toc43399528)

[a. setAssociated 57](#_Toc43399529)

[b. getAssociated 57](#_Toc43399530)

[c. hasAssociated 57](#_Toc43399531)

[d. addMultipleAssociated 57](#_Toc43399532)

[e. setMultipleAssociatedModels 57](#_Toc43399533)

[f. getMultipleAssociatedModels 57](#_Toc43399534)

[3. Test des model 57](#_Toc43399535)

[a. Création du contrôler **test-model** 57](#_Toc43399536)

[**b.** Modification de la classe **FrontSubApplication** 58](#_Toc43399537)

[c. Code de test 59](#_Toc43399538)

[4. Déploiement des models 60](#_Toc43399539)

[a. Créer le processus de mise à jour 60](#_Toc43399540)

[b. Modifier la version de l’application 61](#_Toc43399541)

[c. Code finale de l’application 62](#_Toc43399542)

[d. Démarrer la mise à jour 63](#_Toc43399543)

[X. La persistence (DAO) 64](#_Toc43399544)

[1. Récapitulatif des méthodes utiles pour la manipulation des données 64](#_Toc43399545)

[2. Manipulation de données 70](#_Toc43399546)

[a. Initialisation 70](#_Toc43399547)

[b. Enregistrement 71](#_Toc43399548)

[c. Sélection 74](#_Toc43399549)

[d. Suppression 90](#_Toc43399550)

[e. Méthodes communes 91](#_Toc43399551)

[3. Utiliser une autre source d’accès aux données 91](#_Toc43399552)

[4. Implémentation d’un nouveau support d’accès aux données 91](#_Toc43399553)

[5. Enregistrer un nouveau support d’accès au données 91](#_Toc43399554)

[XI. Back office 91](#_Toc43399555)

[1. Initialisation 91](#_Toc43399556)

[a. Configuration 91](#_Toc43399557)

[b. Création de la sous application pour le Back office 91](#_Toc43399558)

[c. Association du back office à la classe que vous venez de créer 92](#_Toc43399559)

[d. Configuration du thème pour le back office 92](#_Toc43399560)

[e. Activer l’authentification 93](#_Toc43399561)

[f. Activer la vérification des droits d’accès 94](#_Toc43399562)

[XII. Les projets 95](#_Toc43399563)

[1. L’application 95](#_Toc43399564)

[a. Création d’une nouvelle application. 95](#_Toc43399565)

[b. Obtention de l’instance de l’application depuis une autre classe 98](#_Toc43399566)

[c. Méthodes utiles de l’interface Application 98](#_Toc43399567)

[2. Le framework 102](#_Toc43399568)

[3. Les modules 102](#_Toc43399569)

[a. Création d’un module 102](#_Toc43399570)

[b. Déploiement d’un module 104](#_Toc43399571)

[c. Installation d’un module 104](#_Toc43399572)

[d. Administration des modules installés 105](#_Toc43399573)

[4. Méthodes de l’interface \muuska\project\Project 107](#_Toc43399574)

[a. getName 107](#_Toc43399575)

[b. getType 107](#_Toc43399576)

[c. getVersion 107](#_Toc43399577)

[d. hasSubProject 107](#_Toc43399578)

[e. getSubProject 107](#_Toc43399579)

[f. getInstalledInfo 107](#_Toc43399580)

[g. isInstalled 107](#_Toc43399581)

[h. isActive 107](#_Toc43399582)

[i. getCorePath 107](#_Toc43399583)

[j. getCoreDir 107](#_Toc43399584)

[k. getTranslator 108](#_Toc43399585)

[l. createAssetGroup 108](#_Toc43399586)

[m. createAsset 108](#_Toc43399587)

[n. createHtmlImage 108](#_Toc43399588)

[o. createTemplate 108](#_Toc43399589)

[p. createJSTranslation 108](#_Toc43399590)

[q. createResourceTree 108](#_Toc43399591)

[XIII. Les sous projets 108](#_Toc43399592)

[1. Les méthodes de l’interface **SubProject** 109](#_Toc43399593)

[a. getTranslator 109](#_Toc43399594)

[b. createAssetGroup 109](#_Toc43399595)

[c. createAsset 109](#_Toc43399596)

[d. createHtmlImage 109](#_Toc43399597)

[e. createTemplate 109](#_Toc43399598)

[f. createController 109](#_Toc43399599)

[2. Création d’un sous projet 109](#_Toc43399600)

[a. Création de la classe du sous projet 109](#_Toc43399601)

[b. Association de la classe du sous projet au projet 110](#_Toc43399602)

[3. Les sous applications 110](#_Toc43399603)

[4. Création d’un contrôleur 111](#_Toc43399604)

[5. Url permettant d’accéder au contrôleur d’un module 112](#_Toc43399605)

[XIV. Internationalisation 113](#_Toc43399606)

[1. Traducteur 113](#_Toc43399607)

[a. Exemple 113](#_Toc43399608)

[b. Multiple Loader 116](#_Toc43399609)

[2. Traducteur de projet 117](#_Toc43399610)

[a. Configuration principale 118](#_Toc43399611)

[b. Configuration des alertes 119](#_Toc43399612)

[c. Configuration des contrôleurs 121](#_Toc43399613)

[d. Configuration des options 122](#_Toc43399614)

[e. Configuration des models 124](#_Toc43399615)

[f. Configuration des templates 126](#_Toc43399616)

[g. Configuration des JS 128](#_Toc43399617)

[h. Configuration personnalisé 129](#_Toc43399618)

[i. Configuration supplémentaires 130](#_Toc43399619)

[3. Gestion du multilingue des models 131](#_Toc43399620)

[a. Activer le multilingue pour un model 131](#_Toc43399621)

[b. Activer le multilingue pour un champ 131](#_Toc43399622)

[c. Mise à jour du support d’accès aux données 131](#_Toc43399623)

[d. Méthode de récupération des valeurs traductibles 133](#_Toc43399624)

[4. Gestion du multilingue avec le DAO 134](#_Toc43399625)

[Sélection 134](#_Toc43399626)

[Enregistrement 134](#_Toc43399627)

[CRUD 135](#_Toc43399628)

[List 135](#_Toc43399629)

[Form 135](#_Toc43399630)

[View 135](#_Toc43399631)

[Contrôleur virtuel 135](#_Toc43399632)

[Composant HTML 136](#_Toc43399633)

[Html Page 137](#_Toc43399634)

[Html Custom Element 137](#_Toc43399635)

[Commande 138](#_Toc43399636)

[Bouton 138](#_Toc43399637)

[Lien 138](#_Toc43399638)

[Label 138](#_Toc43399639)

[Input 138](#_Toc43399640)

[Default input 138](#_Toc43399641)

[Input Date 138](#_Toc43399642)

[Input hidden 138](#_Toc43399643)

[TextArea 138](#_Toc43399644)

[Rich text editor 138](#_Toc43399645)

[File upload 138](#_Toc43399646)

[Checkbox 138](#_Toc43399647)

[Select 138](#_Toc43399648)

[Radio 138](#_Toc43399649)

[Radio Switch 138](#_Toc43399650)

[Formulaire 139](#_Toc43399651)

[Form 139](#_Toc43399652)

[Field 139](#_Toc43399653)

[Grid field 139](#_Toc43399654)

[Translatable Field 139](#_Toc43399655)

[Grid Translatable Field 139](#_Toc43399656)

[Quick search form 139](#_Toc43399657)

[Dropdown 139](#_Toc43399658)

[Default 139](#_Toc43399659)

[Split 139](#_Toc43399660)

[Custom 139](#_Toc43399661)

[Icon 139](#_Toc43399662)

[Panel 139](#_Toc43399663)

[Default 139](#_Toc43399664)

[List panel 139](#_Toc43399665)

[Navigation 139](#_Toc43399666)

[Nav 139](#_Toc43399667)

[Nav Item 139](#_Toc43399668)

[Full navigation 139](#_Toc43399669)

[Liste 139](#_Toc43399670)

[Definition 139](#_Toc43399671)

[Item 139](#_Toc43399672)

[Field 139](#_Toc43399673)

[Tree 139](#_Toc43399674)

[Image 139](#_Toc43399675)

[Relative Image 139](#_Toc43399676)

[Children Container 139](#_Toc43399677)

[Renderer 140](#_Toc43399678)

[HtmlContentRenderer 140](#_Toc43399679)

[Concept **Erreur ! Signet non défini.**](#_Toc43399680)

[Template 140](#_Toc43399681)

[Callback renderer 140](#_Toc43399682)

[ValueRenderer 140](#_Toc43399683)

[Concept **Erreur ! Signet non défini.**](#_Toc43399684)

[Model File 140](#_Toc43399685)

[Option Label 140](#_Toc43399686)

[Simple Value renderer 140](#_Toc43399687)

[Image renderer 140](#_Toc43399688)

[Class Icon renderer 140](#_Toc43399689)

[Asset 140](#_Toc43399690)

[Absolute 140](#_Toc43399691)

[Relative 140](#_Toc43399692)

[Group 140](#_Toc43399693)

[Contenu JS 140](#_Toc43399694)

[Variable JS 140](#_Toc43399695)

[Contenu CSS 140](#_Toc43399696)

[Conteneur 140](#_Toc43399697)

[Scope 140](#_Toc43399698)

[Meta 140](#_Toc43399699)

[Link 140](#_Toc43399700)

[JS 140](#_Toc43399701)

[Ajax 141](#_Toc43399702)

[Controlleur 141](#_Toc43399703)

[Getter 141](#_Toc43399704)

[Concept 141](#_Toc43399705)

[Classes exist ante 141](#_Toc43399706)

[Création 141](#_Toc43399707)

[Checker 141](#_Toc43399708)

[Concept 141](#_Toc43399709)

[Classes existante 141](#_Toc43399710)

[Création 141](#_Toc43399711)

[Validation 141](#_Toc43399712)

[Gestion des options 141](#_Toc43399713)

[Concept 141](#_Toc43399714)

[Création d’un fournisseur d’option 141](#_Toc43399715)

[Gestion du cache 141](#_Toc43399716)

[Configuration 142](#_Toc43399717)

[L’interface \muuska\config\Configuration 142](#_Toc43399718)

[Récupération de la configuration principale 142](#_Toc43399719)

[Création d’une configuration 142](#_Toc43399720)

[JSON 142](#_Toc43399721)

[DAO 142](#_Toc43399722)

[Ajout d’une configuration à l’application 142](#_Toc43399723)

[Obtention d’une Configuration enregistré à l’application 142](#_Toc43399724)

[Mail 142](#_Toc43399725)

[Concept 142](#_Toc43399726)

[Envoi de mail 142](#_Toc43399727)

[URL 142](#_Toc43399728)

[Controller URL creator 142](#_Toc43399729)

[Object URL 142](#_Toc43399730)

[Model URL 142](#_Toc43399731)

[Array URL 142](#_Toc43399732)

[Pagination URL 142](#_Toc43399733)

[List Limiter URL 142](#_Toc43399734)

[Gestion des utilisateurs 142](#_Toc43399735)

[Current User 142](#_Toc43399736)

[Person resolver 142](#_Toc43399737)

[Gestion des Evènements 143](#_Toc43399738)

[Concept 143](#_Toc43399739)

[Ecouter un évènement 143](#_Toc43399740)

[Helpers 143](#_Toc43399741)

[HTTP 143](#_Toc43399742)

[Request 143](#_Toc43399743)

[Response 143](#_Toc43399744)

[Cookie 143](#_Toc43399745)

[Visitor info recorder 143](#_Toc43399746)

[Session 143](#_Toc43399747)

[Router 143](#_Toc43399748)

[Evenement 143](#_Toc43399749)

[Route loading event 143](#_Toc43399750)

[Request parsing event 143](#_Toc43399751)

[Utilitaires 143](#_Toc43399752)

[Tools 143](#_Toc43399753)

[StringTools 143](#_Toc43399754)

[FileTools 143](#_Toc43399755)

[ArrayTools 143](#_Toc43399756)

[Développement de thème 144](#_Toc43399757)

[Concept 144](#_Toc43399758)

[Création d’un thème 144](#_Toc43399759)

[Utilisation d’un thème 144](#_Toc43399760)

[Surcharge 144](#_Toc43399761)

[Concept 144](#_Toc43399762)

[Procedure 144](#_Toc43399763)

# Introduction

Muuska est un framework PHP très maniable permettant le développement d’applications Web rapide. Il est construit autour du modèle MVC.

Fonctionnalités de Muuska Framework

* Système de génération de contenu HTML (Formulaires, tableaux, lien, arbre, éditeur de texte enrichie).
* Système d’upload sophistiqué pouvant uploader des fichiers de grandes capacités peu importe les limites du serveur (upload\_max\_filesize).
* Support du multilingue.
* Support des modules.
* Support des thèmes.
* Implémentation du Back office.

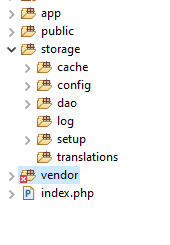
Avantages de Muuska Framework

* Framework purement orienté objet.
* Routage URI flexible et simple.
* Séparation entre les traitements et le support d’accès aux données. Ce qui permet qu’a un moment données ont peut décider d’utiliser les fichiers JSON plutôt qu’une base de donnée MYSQL pour le stockage des données. Le système permettra de faire ce changement sans avoir à modifier les contrôleurs mais en permettant au développeur de créer une stratégie de manipulation des données en utilisant les fichiers JSON.
* Manipulation facile des templates.
* Code réutilisable et plus facile à maintenir.
* Mise à disposition d’un ensemble de fonctions pour traiter certaines tâches récurrentes.
* Gestion des droits d’accès sophistiqués.
* Système d’installation, et configuration facile.
* Mise en place d’un système permettant de faire les opérations CRUD (Create, Read, Update, Delete).

# Déploiement

Décompressez l’archive dans le répertoire de votre serveur.

# Architecture des dossiers



## app

Dossier contenant les sources propre à votre projet.

## public

Dossier content les ressources publiques de votre application (Fichiers CSS, JS, Images, etc…) c’est le seule dossier accessible depuis l’extérieur.

## vendor

Dossier contenant les bibliothèques utilisées par votre application.

## storage

Dossier contenant les données utilisé par votre application (configuration, cache, traductions, log, etc…).

## Index.php

Point d’entrée de l’application

# Configuration de base

## Configuration principale

Ouvrez le fichier « **storage/config/main.json** ». Ce fichier contient les configurations de base de votre application.



### dao\_source

Représente le nom du principal support d’accès aux données.

### cookie\_iv

Représente l’une des clés utilisées pour crypter les cookies.

### cookie\_key

Représente l’une des clés utilisées pour crypter les cookies. Elle est aussi utilisée pour crypter les mots de passes.

### default\_lang

Représente la langue par default.

### dev\_mode

Indique si vous êtes en mode développeur ou pas.

### app\_installed

Indique que l’application est installée.

## Configuration de la base de données

Ouvrez le fichier « **storage/config/db.json** ». Ce fichier contient les informations de configuration de votre base de données.



### host

Représente l’adresse du serveur de base de données

### user

Représente le nom d’utilisateur.

### password

Représente le mot de passe.

### database

Représente le nom de la base de données.

### table\_prefix

Représente le préfix des tables.

### driver

Représente le nom du driver de base de données. La valeur par défaut est « **mysql** ». Les valeurs possibles sont **oracle**, **pgsql**, etc…

### dsn

Représente le **dsn** complet d’accès à la base de données. Ce champ est facultatif. Car les informations précédentes seront utilisées pour créer le **dsn**. Ce champ peut être utile au cas où vous voulez utiliser un driver qui n’utilise pas les informations précédentes (Par exemple **sqllite**, **odbc**).

## Configuration des sous applications

Ouvrez le fichier « **storage/config/sub\_apps.json** ». Ce fichier contient les informations de configuration des sous applications disponibles (Front office, Back office, API, etc…).



Dans ce fichier, la clé représente le nom de la sous application (**FRONT**, **ADMIN**, **API**, etc…).

### url\_path

Représente le chemin relatif (par exemple **admin**, **api**) permettant d’accéder à la sous application. Le chemin complet sera sous la forme {**baseUrl**}/{**url\_path**} où **baseUrl** représente l’url de base par exemple http:localhost/muuska ou https:www.domain.com.

**Exemple de chemin complet** : http:localhost/muuska/**admin** (où **url**\_**path** = **admin**), http:localhost/muuska/**api** (où **url**\_**path** = **api**), http:localhost/muuska (où url\_**path** est vide), https:www.domain.com/**admin**.

### lang\_in\_url

Indique si la langue doit être ajoutée dans l’URL finale ou pas. Si la valeur de ce champ est **true**, le chemin complet sera sous la forme {**baseUrl**}/{**url\_path**}/{**lang**} où **lang** représente le code de la langue.

**Exemple de chemin complet** : http:localhost/muuska/admin/en (où **lang** = **en**), http:localhost/muuska/admin/**fr** (où **lang** = **fr**), http:localhost/muuska/admin (où **lang** est vide), https:www.domain.com/**fr** (où **url\_path** est vide et **lang** = **fr**).

### host

Permet de définir une adresse spécifique pour la sous application. Vous pouvez par exemple utiliser un sous domaine comme **host** de votre sous application (admin.domain.com, api.domain.com).

### authentification\_required

Indique si l’authentification est requise pour accéder à la sous application ou pas.

### access\_checking\_required

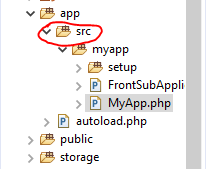
Indique si la vérification des droits d’accès est activée.

1. cookie\_lifetime

Représente le nombre d’heure pendant lesquelles les cookies du visiteur seront conservés.

# Code de base

Les namespaces doivent être crée en suivant le principe utilisé en **JAVA** pour la gestion des packages. Le dossier « **src** » est le dossier racine des classes.



Donc à l’intérieur du dossier « **src** » tout sous-dossier sera inclut dans le namespace.

* La classe « **MyApp** » qui se trouve dans le dossier « **myapp** » aura pour namespace « **myapp** ». Le nom complet de la classe sera « \**myapp\MyApp** ».
* La classe « **AppSetup** » qui se trouve dans le dossier « **myapp/setup** » aura pour namespace « **myapp\setup** ». Le nom complet de la classe sera « \**myapp\setup\AppSetup** ».

## Classe principale de l’application



### La methode **registerMainDAOSources**

Permet d’enregistrer les sources d’accès aux données disponibles.

### La méthode createSubProject

Permet de créer l’instance de la sous application.

### La méthode createAppSetup

Permet de créer le setup de l’application.

## La classe du setup



## La classe de la sous application



# Installation

Ouvrez votre navigateur et accédez au chemin de votre projet Muuska (par exemple, **http://localhost/muuska/en/install**) pour démarrer l'installation.

# Architecture du système

## Architecture logique

Cette partie explique le motif de conception du système (Design pattern).

Les différentes couches du système sont :

### Le modèle

Représente Les classes persistantes du système. Par exemple Une classe Livre.

**NB** : Les classes du modèle ne s’occupe pas des traitements concernant la persistance des données.

### La vue

Concerne la forme (représentation) elle interagit avec le modèle. La vue concerne principalement la représentation des données du modèle à l'écran (ou sur tout autre périphérique de sortie).

### Le contrôleur

Le contrôleur gère les interactions avec l'utilisateur et détermine quels traitements doivent être réalisés.

### La persistance ou DAO (Data Access Object)

Chargée de gérer les objets persistants c'est à dire gère les interactions avec le support d'accès aux données (par exemple la base de données).

## Schémas global



### Requête

Représente tout appel effectué vers le serveur. Ça peut être une requête provenant du navigateur, un appel depuis un autre serveur, une requête AJAX…

**Exemple** : **localhost/muuska/en/home**

### Index.php

Point d’entrée principal du système son rôle est de crée le routeur et l’initialiser.

### Application

Classe principale du système. Elle délègue l’analyse des requêtes au Routeur.

### Routeur

Classe permettant d’analyser les requêtes et gérer le routage des url.

Son rôle est de récupérer la requête de l’utilisateur, la formater. À partir de cette requête, détecter la langue de navigation, savoir quel sous application (Back office ou Front office, API, etc..) est concerné, de trouver le contrôleur à exécuter. Dès qu’il a trouvé le contrôleur correspondant, sont travail restant consiste à récupérer l’instance de la sous application et lui demander d’exécuter le contrôleur.

### Sous application

Classe de chargé de gérer une sous partie (Front office, Back office, API, etc…) de l’application. Lors du traitement d’une requête, c’est elle qui doit instancier le contrôleur approprié et l’exécuter.

### Contrôleur

Analyse la requête de l’utilisateur et réalise le traitement approprié puis retourne la réponse.

### Réponse

Résultat retourné sur à l’utilisateur.

## Diagramme de séquence du système



**Résumé** :

* **La requête** : L’utilisateur initie la requête (Exemple : localhost/muuska/en/home) vers le serveur.
* **Le fichier index.php** : instancie l’application et exécute sa méthode « **run** ».
* **L’application** : instancie le routeur et exécute sa méthode « **run** ».
* **Routage** : Le routeur construit le requête et initialise le flux de réponse ensuite, il lance l’analyse de la requête afin de déterminé les paramètres nécessaires à l’identification du contrôleur (Le nom du contrôleur, l’action, la langue de navigation, la sous application concerné, etc…). Une fois l’analyse de la requête terminée, le routeur doit récupérer l’instance de la sous application concerné par la requête puis demander à la sous application d’exécuter le contrôleur.
* **Exécution de contrôleur** : La sous application doit créer l’instance du contrôleur pour traiter la requête de l’utilisateur. Une fois l’instance du contrôleur crée, la sous application doit vérifier l’authentification puis vérifier les accès de l’utilisateur. Une fois toutes les vérifications effectuées, la sous application appellera la méthode « **executeAction** » du contrôleur trouvé.
* **Exécution de l’action par le contrôleur** : Le contrôleur effectue le traitement approprié en fonction de l’action de l’utilisateur puis appelle la méthode « **outputResult** » pour le retour de la réponse.

# Contrôleur

Analyse la requête de l’utilisateur et réalise le traitement approprié puis retourne la réponse.

## Création d’un contrôleur

Pour créer un contrôleur, allez dans le dossier « **myapp/controller/front**», ensuite ajouté votre fichier par exemple « **HelloWorldController.php** ».

Le nom de la classe doit être identique à celui du fichier ; cette classe doit hériter de la classe **« muuska\controller\AbstractController »**.



Une fois la classe crée, il faut fournir le moyen de créer une instance de cette classe. Pour le faire, ouvrez la classe « **FrontSubApplication** » ( **\myapp\FrontSubApplication** » et redéfinissez la méthode « **createController** »



Votre contrôleur est enfin prêt il ne reste plus qu’à entrer l’URL pour y accéder.

## Url d’un contrôleur

L’URL permettant l’accès à un contrôleur doit être sous la forme {**baseUrl**}/ {**subAppUrlPath**}{**lang**}/{**controllerName**}/{**action**}

* {**baseUrl**} : variable représentant l’url de base. Exemple http:localhost/muuska ou https:www.domain.com. Si la sous application à laquelle le contrôleur est lié possède dans sa configuration une valeur pour ce champ « **host** », c’est cette valeur qui sera utilisé.
* {**subAppUrlPath**} : correspond au champ « **url\_path** » présent dans la configuration de la sous application courante suivi du « **/** ». Au cas où la valeur de « **url\_path** » est vide, vous ne devez pas ajouter le « / ».
* {**lang**} : Le code de la langue. Exemple « **fr** » ou « **en** ». Si la sous application à laquelle le contrôleur est lié à définit dans sa configuration le champ « lang\_in\_url » avec la valeur « false », vous ne devez pas spécifier le code de langue dans l’url.
* {**controllerName**} : Le nom du contrôleur. Cette valeur doit être en minuscule donc si le nom du contrôleur est composé de plusieurs mots (Par exemple « **HelloWorld** »), il faudra utiliser un trait d’union (**-**) pour séparer les mots (Le nom du contrôleur sera « **hello-world**»). Au cas où vous ne le spécifiez pas, le nom du contrôleur correspondra à « **home** ».
* {**action**} : Le nom de l’action. Cette valeur doit être en minuscule donc si le nom de l’action est composé de plusieurs mots (Par exemple « **UpdatePassword** »), il faudra utiliser un trait d’union (**-**) pour séparer les mots (Le nom du contrôleur sera « **update-password**»). Au cas où vous ne le spécifiez pas, le nom d l’action correspondra à « **default** ».

Exemple d’URL :

http:localhost/muuska/en où {**baseUrl**} = **http:localhost/muuska/**, {**subAppUrlPath**} est vide, {**lang**} = **en**, {**controllerName**} est vide (Mais la valeur considéré sera « **home** »), {**action**}  est vide (Mais la valeur considéré sera « **default** »).

http:localhost/muuska/en/hello-world où {**baseUrl**} = **http:localhost/muuska/**, {**subAppUrlPath**} est vide, {**lang**} = **en**, {**controllerName**} = **hello-world**, {**action**}  est vide (Mais la valeur considéré sera « **default** »).

http:localhost/muuska/en/hello-world/goodbye où {**baseUrl**} = **http:localhost/muuska/**, {**subAppUrlPath**} est vide, {**lang**} = **en**, {**controllerName**} = **hello-world**, {**action**} = **goodbye**.

http:localhost/muuska/admin/en/hello-world où {**baseUrl**} = **http:localhost/muuska/**, {**subAppUrlPath**} = **admin**, {**lang**} = **en**, {**controllerName**} = **hello-world**, {**action**}  est vide (Mais la valeur considéré sera « **default** »).

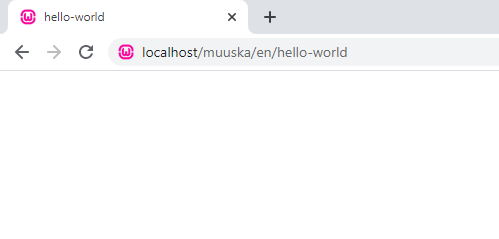
http:localhost/muuska/admin/hello-world où {**baseUrl**} = **http:localhost/muuska/**, {**subAppUrlPath**} = **admin,** {**lang**} n’est pas renseigné car dans la configuration de la sous application courante, le champ **lang\_in\_url** a la valeur « **false »**, {**controllerName**} = **hello-world**, {**action**}  est vide (Mais la valeur considéré sera « **default** »).

https:www.domain.com/en/hello-world/goodbye où {**baseUrl**} = **https:www.domain.com**, {**subAppUrlPath**} est vide, {**lang**} = **en**, {**controllerName**} = **hello-world**, {**action**} = **goodbye**.

https:www.api.domain.com/en/hello-world/goodbye où {**baseUrl**} = https:www.api.domain.com car dans la configuration de la sous application courante le champ « **host** » à la valeur **www.api.domain.com**, {**subAppUrlPath**} est vide, {**lang**} = **en**, {**controllerName**} = **hello-world**, {**action**} = **goodbye**.

https:www.domain.com/en/hello-world,

Pour accéder au contrôleur que vous venez de créer, Entrez L’url du contrôleur (http:localhost/muuska/en/hello-world) dans votre navigateur ; Vous allez avoir une page similaire à la suivante :



La page est encore vide.

## La notion d’action

Le contrôleur effectue des traitements en fonction de l’action spécifié dans la requête.

* Si le nom de l’action dans la requête est composé de plusieurs mots, il faut utiliser un tiret (-) pour séparer les mots. Exemple « **update-password** » (http:localhost/muuska/en/hello-world/update-password)
* Si aucune action n’est spécifiée, le nom de l’action aura la valeur « **default** »
* La méthode à exécuter pour le traitement d’une action dans un contrôleur doit être nommé ainsi : process{**actionName**} où **actionName** Représente le nom de l’action en nomenclature « **Camel case** » (Par exemple « **UpdatePassword** » avec la première lettre de l’action en majuscule.

Exemple : **processUpdatePassword**, **processAdd**, **processDefault**.

Pour notre contrôleur **HelloWorld**, nous allons utiliser la méthode **processDefault**

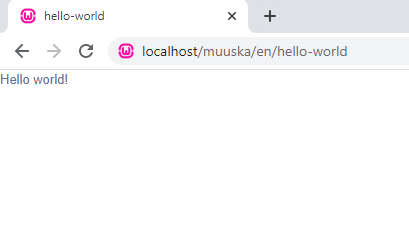


Pour spécifier le contenu qui sera affiché, il faut ajouter l’instruction suivante :



Le code complet de la classe **HelloWorld** sera le suivant :



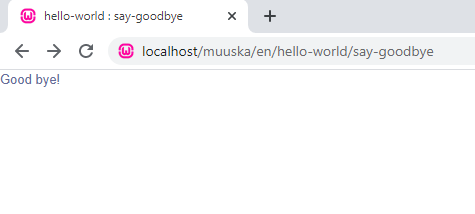


Ajout d’une nouvelle action dans le contrôleur.

Nous allons ajouter l’action « **Say goodbye** » au contrôleur **HelloWorld**.



Allez dans votre navigateur et entrez l’url en spécifiant l’action « **say-goodbye** » (http:localhost/muuska/en/hello-world/say-goodbye).



## Controller Input

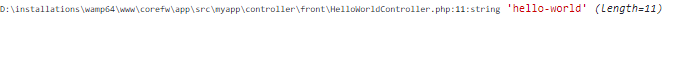
Est un objet de type **\muuska\controller\ControllerInput** fournit au contrôleur au moment de son instanciation. Cette classe contient les ressources dont le contrôleur a besoin pour réaliser ses traitements.

Pour accéder à cet objet depuis votre contrôleur, écrivez **$this->input**.

Voici la liste des méthodes disponibles :

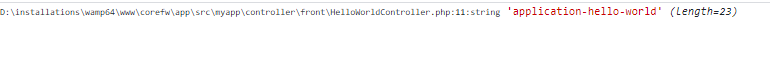
### getName

Retourne le nom du contrôleur courant.

### getFullName

Retourne le nom complet du contrôleur.

### checkName

Vérifie si le nom du contrôleur est égal au nom que vous passez en paramètre.

### checkFullName

Vérifie si le nom complet du contrôleur est égal au nom que vous passez en paramètre.

### getAction

Retourne l’action courante.

### getRequestType

Retourne le type de requête.

### isAjaxRequest

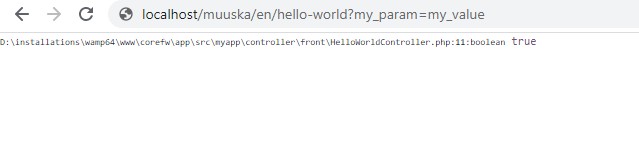
Vérifie s’il s’agit d’une requête AJAX ou pas. Les requêtes AJAX doivent avoir dans l’URL le un paramètre « **ajax** » qui aura la valeur 1. Exemple <http://localhost/muuska/en/hello-world?ajax=1>.

### hasQueryParam

Vérifie si un paramètre est présent dans l’URL.



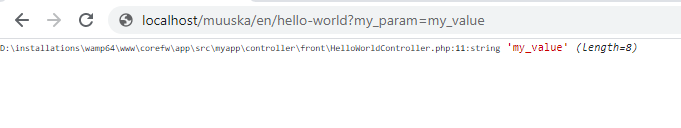
Ajoutez le paramètre **my\_param** dans l’URL pour tester. (<http://localhost/muuska/en/hello-world?my_param=my_value>).



### getQueryParam

Retourne la valeur d’un paramètre présent dans l’URL. Elle prend en paramètre le nom du paramètre et la valeur à retourner au cas où le paramètre n’existe pas.





### hasPostParam

Vérifie si un paramètre est présent dans le **POST**.

### getPostParam

Retourne la valeur d’un paramètre présent dans le **POST**. Elle prend en paramètre le nom du paramètre et la valeur à retourner au cas où le paramètre n’existe pas.

### hasParam

Vérifie si un paramètre est présent dans l’URL ou dans le POST.

### getParam

Retourne la valeur d’un paramètre présent dans l’URL ou dans le **POST**. Elle prend en paramètre le nom du paramètre et la valeur à retourner au cas où le paramètre n’existe pas.

### getLang

Retourne le code de la langue courante.



Modifier le code de langue dans l’URL et validez.



### getLanguageInfo

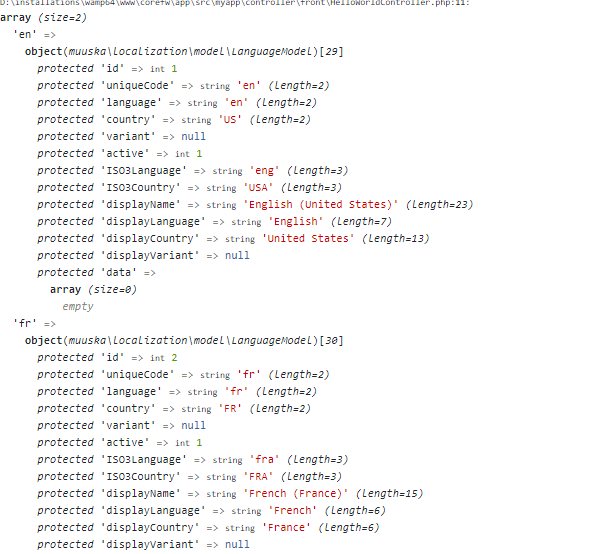
Retourne l’instance de la langue courante. L’objet retourné est de type **\muuska\localization\languageInfo**.



### getLanguages

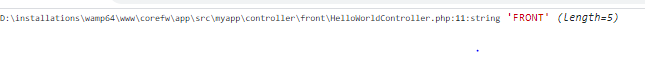
Retourne la liste des langues actives. Elle retourne un tableau où chaque element est un objet de type **\muuska\localization\languageInfo**.





### getSubAppName

Retourne le nom de la sous application courante.



### getProject

Retourne l’instance du projet courant. L’objet retourné est de type **\muuska\project\Project**.

### getSubProject

Retourne l’instance du sous projet courant. L’objet retourné est de type **\muuska\project\SubProject**.

### getSubApplication

Retourne l’instance de la sous application courante. L’objet retourné est de type **\muuska\project\SubApplication**.

### hasTheme

Vérifie si la sous application courante a un thème.

### getTheme

Retourne l’instance du thème utilisé par la sous application courante. Elle retourne « **null** » si la sous application n’a pas de thème. L’objet retourné est de type **\muuska\util\theme\Theme**.

### getDAO

Retourne l’instance du DAO en fonction de la définition du model fourni en paramètre. L’objet retourné est de type **\muuska\dao\DAO**.

### getDaoFactory

Retourne le gestionnaire d’accès aux données. L’objet retourné est de type **\muuska\dao\DAOFactory**.

### createSelectionConfig

Crée une instance du configurateur de sélection de donnée. L’objet retourné est de type **\muuska\dao\util\SelectionConfig**.

### createSaveConfig

Crée une instance du configurateur d’enregistrement de donnée. L’objet retourné est de type **\muuska\dao\util\SaveConfig**.

### createDeleteConfig

Crée une instance du configurateur de suppression de donnée. L’objet retourné est de type **\muuska\dao\util\DeleteConfig**.

### getRequest

Retourne l’instance de la requête de l’utilisateur. L’objet retourné est de type **\muuska\http\Request**.

### getResponse

Retourne l’instance du flux de réponse de l’utilisateur. L’objet retourné est de type **\muuska\http\Response**.

### getVisitorInfoRecorder

Retourne l’objet permettant d’enregistrer les informations du visiteur courant (Cookie ou Session). L’objet retourné est de type **\muuska\http\VisitorInfoRecorder**.

### getCurrentUser

Retourne l’instance du gestionnaire de l’utilisateur courant. L’objet retourné est de type **\muuska\security\CurrentUser**.

## Controller result

Est un objet de type **\muuska\controller\DefaultControllerResult**. Cet objet contient le résultat du d’exécution d’une action.

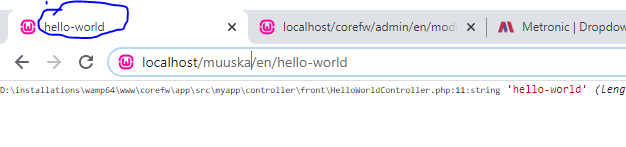
Pour accéder à cet objet depuis votre contrôleur, écrivez **$this->result**.

Voici la liste des méthodes disponibles :

### getTitle

Retourne le titre de la page.

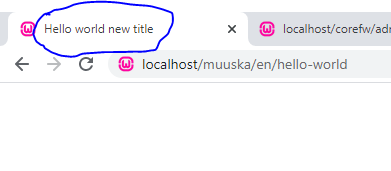




### setTitle

Modifie le titre de la page.





### setContent

Modifie le contenu principal de la page. Elle prend en paramètre un objet de type **\muuska\html\HtmlContent** qui représente un composant HTML. Vous verrez la liste des composants disponibles dans le namespace **\muuska\html\.**





### hasContent

Vérifie si la page a un contenu principal.

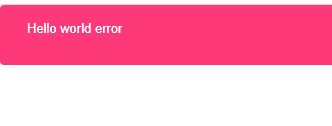
### getContent

Retourne le contenu principal de la page. L’objet retourné est de type **\muuska\html\HtmlContent**. Si le contenu principal n’existe pas elle retourne « **null** »

### addError

Ajoute un message d’erreur à la page.

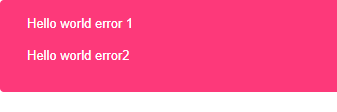




### addErrors

Ajoutes des messages d’erreurs à la page.





### hasErrors

Vérifie si la page à des erreurs.

### addSuccess

Ajoute un message de succès à la page.

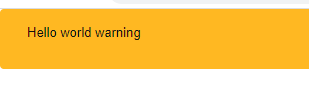




### addAlert

Ajoute une alerte à la page. Elle prend en paramètre le type de l’alerte (les types d’alertes disponibles sont dans la classe **\muuska\html\constants\AlertType**) et le message de l’alerte.





### addAlerts

Ajoute plusieurs messages d’alerte en fonction du type.





### getAssetSetter

Retourne le gestionnaire des assets (css, js, etc…). L’objet retourné est de type **\muuska\asset\AssetSetter**.

### setRedirection

Renseigne une redirection. Elle prend en paramètre un objet de type **\muuska\http\redirection\Redirection**.

### hasRedirection

Vérifie si une redirection a été renseignée.

## Controller param

Est un objet de type **\muuska\controller\param\ControllerParamResolver**. Cet objet vous permet de définir les paramètres de base de votre contrôleur et de récupérer facilement leur valeur. Ainsi, vous n’aurez plus à vérifier vous-même si un paramètre obligatoire (Par l’identifiant d’un objet au moment de Edition) est présent.

Pour accéder à cet objet depuis votre contrôleur, écrivez **$this->paramResolver**.

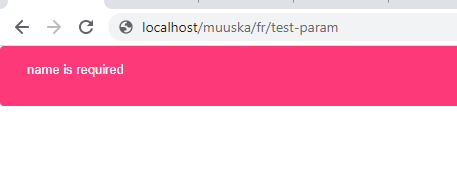
Pour configurer votre gestionnaire de paramètre, vous devez redéfinir la méthode « **initParamResolver** » dans votre contrôleur.

Pour définir un paramètre, l’interface **\muuska\controller\param\ControllerParamParser** énumère la liste des méthodes attendues. Il existe trois types de paramètre :

### Paramètre standard

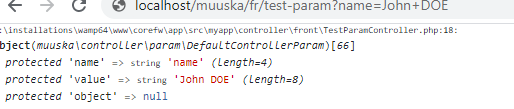
La classe concernée est **\muuska\controller\param\DefaultControllerParamParser**.





La méthode « **processDefault** » ne s’exécute pas car le paramètre « **name** » n’a pas été renseigné dans l’URL.

Si vous ajoutez le paramètre « **name** » dans l’url vous verrez le contenu de la variable « **param** » :



### Paramètre lié à un model

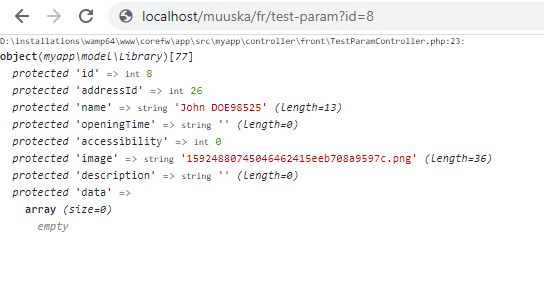
La classe concernée est **\muuska\controller\param\ModelControllerParamParser**.





L’erreur affiché est dû au fait qu’il y-a pas de bibliothèque enregistré avec l’identifiant « **10** ».

Si vous entrez un identifiant valide, vous aurez l’affichage suivant :



## Controller URL creator

Est un objet de type **\muuska\url\ControllerUrlCreator**. Cet objet permet de créer les URL.

Pour accéder à cet objet depuis votre contrôleur, écrivez **$this->urlCreator**.

## Méthodes utile du contrôleur

# Model

Représente Les classes persistantes du système. Par exemple Une classe Livre.

Nous allons utiliser un système de gestion des bibliothèques pour expliquer cette partie.



**Modèle physique de données**



Dans votre application, créez un dossier « **model** » qui contiendra vos classes.

Pour chaque classe, vous devez créer sa définition.

## Notion de « Model Définition »

### Exemple de code



### Obtention d’une instance unique

Le bloc de code suivant doit être ajouté dans toutes vos définitions c’est pour s’assurer qu’on ait qu’une seule instance de la classe dans l’application.



### La méthode « createModel »

La méthode « **createModel** » permet fournir le moyen de créer une nouvelle instance d’un model.

### La méthode «createDefinition»

La méthode «**createDefinition**» retourne un tableau associatif définissant la classe. Le tableau suivant explique ces différents champs.

|  |  |  |  |
| --- | --- | --- | --- |
| **Champs** | **Description** | **Type de donnée** | facultatif |
| **name** | Représente le nom de la classe qui sera utilisée par le support de persistance (Par exemple pour la base de données ça représentera le nom de la table). | Chaine | non |
| **primary** | Le nom du champ représentant l’identifiant de l’objet. | Chaine | Oui (Si l’objet a plusieurs identifiant) |
| **primaries** | Nom des champs permettant d’identifier l’objet. | tableau | Oui (Si l’objet a un seul identifiant) |
| **autoIncrement** | Indique si l’on veut que la valeur de l’identifiant d’un objet soit attribué automatiquement.  Ce champ n’est utile que pour les Classes ayant un seul identifiant. | booléen | oui |
| **multilingual** | Permet de spécifier si l’objet prend en compte le multilingue. | booléen | oui |
| **uniques** | Permet de spécifier un groupe d’attributs pour lesquelles on ne veut pas avoir plusieurs enregistrements avec les mêmes couples de valeurs. Par exemple si je ne veux pas avoir deux auteurs avec le même nom et prénom  'uniques' => **array**(  **array**('firstName', 'lastName')  ), | Tableau associatif | oui |
| **fields** | Représente la définition des attributs persistant de la classe. | Tableau associatif | non |
| **presentationFields** | Champs utilisés pour la description de l’objet et aussi pour l’auto complétion.  Si vous ne renseignez pas cette information, le système cherchera les champs appropriés parmi les valeurs suivantes (dans cet order) : « **firstName** + **lastName** »,  « **label** », « **displayName** », « **name** », « **title** », « **code** », « **reference**». | Tableau | oui |
| **presentationFields**  **Separator** | Le séparateur à utiliser au cas où « **presentationFields »** contient plus d’un élément.  La valeur pas défaut est l’espace. | Chaine | oui |
| **projectType** | Le type de projet auquel l’objet est associé.  La valeur par défaut est **\muuska\project\constants\ProjectType::Application** | Chaine | oui |
| **projectName** | Le nom du projet auquel l’objet est associé. | Chaine | oui |
| **associations** | Permet de définir les associations multiples.  C’est tableau associatif où la clé représente le nom de l’association et la valeur est un tableau définissant l’association. La définition de l’association est un tableau associatif avec les infos suivantes :   * **reference** : objet représentant la définition de la classe externe. * **field** : nom du champ représentant l’identifiant de l’objet courant dans la classe externe | tableau | oui |

### Définition des associations multiples



### Définition d’un champ

**Le champ « fields » de la définition du Model retourné par la méthode « createDefinition»**

Le champ « **fields** » est un tableau associatif représentant la définition des attributs persistant de la classe.

**NB** : Si dans la méthode **« createDefinition»**, le champ « **autoIncrement**» est définit avec la valeur « **true**», la propriété représentant l’identifiant de la classe ne doit pas figurer dans le tableau « **fields** ».

Le tableau « **fields** » est de la forme « **clé-valeur** » où :

* La « **clé** » est une chaine de caractère représentant le nom d’une propriété de la classe exemple « **firstName** ».
* La « **valeur** » est un tableau associatif représentant la définition de la propriété. Le tableau suivant liste des champs disponibles :

|  |  |  |  |
| --- | --- | --- | --- |
| **Champs** | **Description** | **Type de donnée** | optionnel |
| **type** | Le type de donnée manipulé par l’attribut.  Ce type est le type sous lequel la donnée sera persistée. (entier, chaine de caractère, booléen, etc…)  Un ensemble de constante sont définit dans la classe **muuska\constants\DataType**. | entier | non |
| **nature** | Représente la nature de la donnée. C’est-à-dire le type d’information que la variable est censé contenir ça peut être un numéro de téléphone, une adresse email, un mot de passe, un nom, etc…  Un ensemble de constante sont définit dans la classes **muuska \constants\FieldNature**. | entier | oui |
| **required** | Permet de spécifier si l’attribut est obligatoire ou pas. | booléen | oui |
| **validationRule** | Permet de spécifier le nom de la méthode qui sera utilisé pour la validation des données de l’attribut. Ce nom doit correspondre à un nom de méthode se trouvant dans la classe **muuska\validation\ValidationRuleManager**.  Exemple « **isPhoneNumber** » | Chaine | oui |
| **validationRules** | Liste des règles de validation | tableau | oui |
| **maxSize** | Le nombre de caractère maximum autorisé par l’attribut. | entier | oui |
| **minSize** | Le nombre de caractère minimum autorisé par l’attribut. | entier | oui |
| **maxValue** | Valeur maximum autorisé | entier | oui |
| **minValue** | Valeur minimum autorisé | entier | oui |
| **unique** | Permet de spécifier la valeur de l’attribut doit être unique par exemple je ne veux pas avoir deux livres dans mon système avec le même code. | booléen | oui |
| **reference** | N’est requis qu’au cas où la nature du champ à la valeur **FieldNature::EXISTING\_MODEL\_ID.**  Il représente l’objet contenant la définition de la classe vers laquelle l’attribut pointe.  Cet objet doit être une instance de la classe  **« muuska\model\ AbstractModelDefinition ».** | Objet | oui |
| **onDelete** | N’est requis qu’au cas où la nature du champ à la valeur **FieldNature::EXISTING\_MODEL\_ID.**  Permet de spécifier le comportement (CASCADE, RESTRICT, etc…) en cas de suppression de la référence de l’objet.  Un ensemble de constante sont définit dans la classe **muuska\dao\constants\ReferenceOption**. | entier | oui |
| **optionProvider** | N’est requis qu’au cas où la nature du champ à la valeur **FieldNature::OPTION**.  Représente l’objet qui fournira les options à utiliser.  Cet attribut est utile au cas où l’on a une liste de valeur prédéfinit pour un champ en d’autre terme ça représente les énumérations.  Cet objet doit être une instance de la classe **« muuska\option\ AbstractOptionProvider ».** | objet | oui |
| **validator** | Permet de spécifier un validateur personnalisé pour le champ.  Cet objet doit implémenter l’interface **\muuska\validation\Validator**.  La classe **\muuska\validation\DefaultValidator**  Implémente cette interface et vous fournit un moyen de créer une instance en définissant une fonction de rappel. | objet | oui |

#### Implémentation d’un validateur personnalisé pour un champ



#### Création des options pour un champ

Ajouter un dossier option dans votre application puis ajouter La classe de votre option



La méthode **initOptions** vous permet d’initialiser vos options. Grace à la méthode **addArrayOption**, vous pouvez ajouter vos options, le premier paramètre représente la valeur de l’option, et le deuxième le label de l’option. Il faut utiliser la méthode « **l** » pour obtenir la traduction d’un texte.

#### Définition d’un champ externe



### Utilisation d’un tableau associatif comme model

Vous n’êtes pas obligé de créer une classe model pour toutes vos définitions. Vous pouvez spécifier dans la définition que vous voulez qu’un tableau associatif soit créé. Pour le faire vous devez ajouter l’instruction suivante dans la définition :



Une fois cette instruction ajouté, à chaque fois qu’une voudra une instance du model, une instance de la classe **muuska\model\ArrayModel** sera retourné.

Cette classe contient les méthodes suivantes :

#### getPropertyValue

Permet d’obtenir la valeur d’une propriété. Elle prend en paramètre le nom de la propriété.

#### setPropertyValue

Permet de modifier la valeur d’une propriété, elle prend en paramètre de nom de la propriété et la nouvelle valeur.

La définition finale de l’objet sera :



## Notion de model

C’est une classe avec les attributs, méthode, les getters et les setters.

Elle doit hériter de la classe **\muuska\model\AbstractModel**



La classe **AbstractModel** contient les méthodes suivantes :

### setAssociated

Permet de modifier l’instance d’un objet externe. Elle prend en paramètre le nom du champ sur lequel la référence de l’objet est définit et l’instance du nouvel objet.

### getAssociated

Permet d’obtenir l’instance d’un objet externe. Elle prend en paramètre le nom du champ sur lequel la référence de l’objet est définit.

### hasAssociated

Permet de vérifier si l’instance d’un objet externe existe. Elle prend en paramètre le nom du champ sur lequel la référence de l’objet est définit.

### addMultipleAssociated

Permet d’ajouter un objet à une association multiple. Elle prend en paramètre le nom de l’association multiple et l’instance de l’objet.

### setMultipleAssociatedModels

Permet modifier les instances un d’une association multiple. Elle prend en paramètre le nom de l’association multiple et un tableau d’objet.

### getMultipleAssociatedModels

Permet d’obtenir les instances d’une association multiple. Elle prend en paramètre le nom de l’association multiple.

## Test des model

Nous allons créer un contrôler « **test-model** » pour tester nos Models.

### Création du contrôler **test-model**

Allez dans votre dossier **controller/front** et créer une classe **TestModelController**



### Modification de la classe **FrontSubApplication**



Entrez l’url <http://localhost/muuska/en/test-model> pour y accéder

### Code de test



## Déploiement des models

Pour déployer les models sur le support d’accès aux données, vous devez faire une mise à jour de votre application en suivant les étapes suivantes :

### Créer le processus de mise à jour

Ouvrez la classe **MyApp** et redéfinissez la méthode **createUpgrade**



Puis ajouter le code suivant :



### Modifier la version de l’application

Il faut modifier la version de l’application afin que la mise à jour puisse se lancer automatiquement à la prochaine exécution de l’application. Pour le faire il faut ajouter l’instruction suivante dans la classe **MyApp**

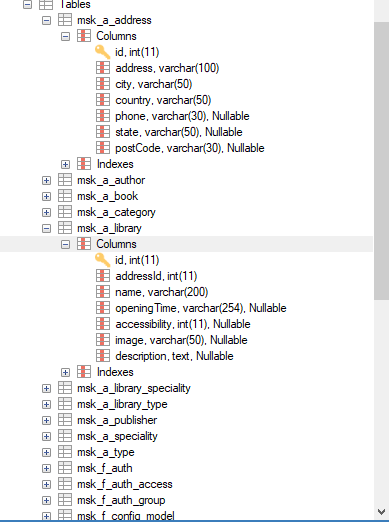


### Code finale de l’application



### Démarrer la mise à jour

Lancer n’importe quel contrôleur de votre application. Le support d’accès aux données courant (Base de données) fera la mise à jour de votre application. Pour vérifier, ouvrez votre client de base de donnée vous verrez vos tables ajouté avec le préfix « **msk\_a\_** ».



# La persistence (DAO)

C’est la partie qui s’occuper de gérer les opérations liées au support d’accès aux données.

## Récapitulatif des méthodes utiles pour la manipulation des données

Nous avons l’interface « **\muuska\dao\DAO** » qui s’occupe de la persistance. Elle offre par défaut un ensemble de méthode que nous avons l’habitude d’utiliser.

|  |  |  |  |
| --- | --- | --- | --- |
| Méthode | Description | Paramètres | Type de retour |
| **add** | Ajoute un objet.  Si l’objet a les attributs « **creationDate** » ou « **lastModifiedDate** », les valeurs seront renseignées automatiquement.  Si la classe de l’objet est définit en auto-incrément, la valeur de l’identifiant de l’objet sera automatiquement renseigné sur l’objet.  Cette méthode lève une exception si l’objet est invalide. | - **model** : L’objet à ajouter.  - **saveConfig** (facultatif) : c’est un objet de la classe **\muuska\dao\util\SaveConfig** permettant de spécifier les options pour l’enregistrement de données. | booléen |
| **update** | Modifie un objet existant.  Si l’objet a un attribut « **lastModifiedDate** », sa valeur sera renseignée automatiquement.  Cette méthode lève une exception si l’objet est invalide. | **- model** : L’objet à modifier.  - **saveConfig** (facultatif) : c’est un objet de la classe **\muuska\dao\util\SaveConfig** permettant de spécifier les options pour l’enregistrement de données. | booléen |
| **save** | Enregistre un objet.  Si l’identifiant de l’objet est renseigné, ça fera une modification au cas contraire, ça fera l’ajout.  Cette méthode n’est fonctionnelle que pour les classes possédant un identifiant auto-incrément.  Cette méthode lève une exception si l’objet est invalide. | **- model** : L’objet à enregistrer.  - **saveConfig** (facultatif) : c’est un objet de la classe **\muuska\dao\util\SaveConfig** permettant de spécifier les options pour l’enregistrement de données. | booléen |
| **updateMultipleRows** | Modifie les informations de plusieurs lignes à partir des données d’ligne à partir des données d’un modèle.  Cette méthode lève une exception les données du model sont invalide. | **- model** : L’objet à partir duquel les données seront récupérées.  - **saveConfig** (facultatif) : c’est un objet de la classe **\muuska\dao\util\SaveConfig** permettant de spécifier les options pour l’enregistrement de données. | booléen |
| **updateMultipleRowsFromArray** | Modifie les informations de plusieurs lignes à partir des données d’ligne à partir des données présentes dans un tableau associatif.  Cette méthode lève une exception les données sont invalide. | **- data** : tableau associatif représentant les données à modifier.  - **saveConfig** (facultatif) : c’est un objet de la classe **\muuska\dao\util\SaveConfig** permettant de spécifier les options pour l’enregistrement de données. | booléen |
| **changeValue** | Modifie la valeur d’un attribut spécifique.  Lève une exception si la classe de l’objet ne possède pas l’attribut spécifié. | **- model** : L’objet sur lequel on veut changer la valeur.  - **field** : Le nom de l’attribut.  - **value** : La nouvelle valeur. | booléen |
| **changeValueOnMultipleRows** | Modifie la valeur d’un attribut spécifique sur plusieurs lignes.  Lève une exception si la classe de l’objet ne possède pas l’attribut spécifié. | - **field** : Le nom de l’attribut.  - **value** : La nouvelle valeur.  - **saveConfig** (facultatif) : c’est un objet de la classe **\muuska\dao\util\SaveConfig** permettant de spécifier les options pour l’enregistrement de données. |  |
| **activate** | Active un objet.  Lève une exception si la classe de l’objet ne possède pas l’attribut « **active** ». | **- model** : L’objet à activer. | booléen |
| **deactivate** | Désactive un objet.  Lève une exception si la classe de l’objet ne possède pas l’attribut « **active** ». | **- model** : L’objet à désactiver. | booléen |
| **setModelNewPosition** | Modifie automatiquement la position d’un objet. En lui donnant la valeur de la position du dernier objet + 1  Lève une exception si la classe de l’objet ne possède pas l’attribut « **position** ». | **- model** : L’objet à modifier. |  |
| **delete** | Supprime un objet. | **- model** : L’objet à supprimer.  - **deleteConfig** (facultatif) : c’est un objet de la classe **\muuska\dao\util\DeleteConfig** permettant de spécifier les options pour la suppression de données. | booléen |
| **deleteMultipleRows** | Supprime plusieurs enregistrements. | - **deleteConfig** (facultatif) : c’est un objet de la classe **\muuska\dao\util\DeleteConfig** permettant de spécifier les options pour la suppression de données. |  |
| **clearData** | Efface toute les données puis réinitialise le DAO. |  | boolean |
| **getById** | Récupère un objet par son identifiant.  Retourne « **null** » si l’objet n’existe pas. | - **id** : Identifiant de l’objet.  - **selectionConfig** (facultatif) : c’est un objet de la classe **\muuska\dao\util\SelectionConfig** permettant de spécifier les options de sélection de données.  - **returnEmptyModelIfNotFound** (facultatif) : Permet de spécifier qu’on veut qu’un Model vide soit retourné si aucun objet n’a été trouvé. Si vous donnez la valeur « **true** » à ce paramètre, il faudra appeler la méthode « **isLoaded** » sur la définition de l’objet pour vérifier si l’objet a été trouvé.  Valeur par défaut « **false** » | objet |
| **getUniqueModel** | Récupère un objet unique.  Retourne « **null** » si l’objet n’existe pas. | - **selectionConfig** (facultatif) : c’est un objet de la classe **\muuska\dao\util\SelectionConfig** permettant de spécifier les options de sélection de données.  - **returnEmptyModelIfNotFound** (facultatif) : Permet de spécifier qu’on veut qu’un Model vide soit retourné si aucun objet n’a été trouvé. Si vous donnez la valeur « **true** » à ce paramètre, il faudra appeler la méthode « **isLoaded** » sur la définition de l’objet pour vérifier si l’objet a été trouvé.  Valeur par défaut « **false** » |  |
| **getModelValue** | Récupère la valeur d’un champ spécifique. | - **selectionConfig** (facultatif) : c’est un objet de la classe **\muuska\dao\util\SelectionConfig** permettant de spécifier les options de sélection de données.  - **field**: Le nom du champ |  |
| **getData** | Récupère une liste d’objet. | - **selectionConfig** (facultatif) : c’est un objet de la classe **\muuska\dao\util\SelectionConfig** permettant de spécifier les options de sélection de données. | Collection d’objets |
| **getDataTotal** | Récupère le nombre de données. | - **selectionConfig** (facultatif) : c’est un objet de la classe **\muuska\dao\util\SelectionConfig** permettant de spécifier les options de sélection de données. |  |
| **loadAssociatedObject** | Charge un objet externe | - **model** : l’objet courant.  - **field** : Le champ auquel l’objet externe est associé.  - **selectionConfig** (facultatif) : c’est un objet de la classe **\muuska\dao\util\SelectionConfig** permettant de spécifier les options de sélection de données.  - **returnEmptyModelIfNotFound** (facultatif) | Objet |
| **loadMultipleAssociatedObjects** | Charge les objets d’une association multiple | - **model** : l’objet courant.  - **associationName** : Le nom de l’association multiple.  - **selectionConfig** (facultatif) : c’est un objet de la classe **\muuska\dao\util\SelectionConfig** permettant de spécifier les options de sélection de données. | Collection d’objets |
| **getParents** | Retourne la liste des objets parent de l’objet courant.  Lève une exception si la classe de l’objet ne possède pas l’attribut « **parentId** ». | - **model** : l’objet courant.  - **selectionConfig** (facultatif) | Collection d’objets |
| **getChildren** | Retourne la liste des objets enfants de l’objet courant.  Lève une exception si la classe de l’objet ne possède pas l’attribut « **parentId** ». | - **model** : l’objet courant.  - **selectionConfig** (facultatif) | Collection d’objets |
| **getForeignDAO** | Retourne le **DAO** d’un objet externe. | - **field** : Le champ auquel le **DAO** externe est associé. | **DAO** |
| **getMultipleAssociationDAO** | Retourne le **DAO** d’une association multiple | - **associationName** : Le nom de l’association multiple. | **DAO** |

## Manipulation de données

Pour cette partie, nous allons créer un contrôleur « **TestDao** »

Nous allons créer un contrôler « **test-dao** ».

### Initialisation

#### Création du contrôler **test-dao**

Allez dans votre dossier **controller/front** et créer une classe **TestDAOController**



#### Modification de la classe **FrontSubApplication**



Entrez l’url http://localhost/muuska/en/test-dao pour y accéder

#### Récupération d’une instance du DAO

Pour Récupérer l’instance d’un DAO, il faut ajouter l’instruction suivante au niveau du contrôleur :



La méthode « **getDAO** » prend en paramètre la définition de l’objet dont on veut récupérer le DAO.

### Enregistrement

#### La classe \muuska\dao\util\SaveConfig

Cette classe possède plusieurs méthodes qui permettent de définir les options d’enregistrement :

##### setExludedFields

Indiquer les champs à exclure pendant l’enregistrement.

##### setSpecificFields

Indiquer les champs spécifiques à enregistrer.

##### createAssociatedFieldSaveConfig

Crée et ajoute la configuration d’enregistrement d’un objet associé. Elle prend en paramètre le nom du champ sur lequel la référence de l’objet externe est définit.

##### createMultipleSaveAssociation

Crée une configuration pour une association multiple

#### L’ajout



Exécuter le contrôleur puis ouvrez votre base de données pour vérifier les enregistrements.

#### La modification



Exécuter le contrôleur puis ouvrez votre base de données pour vérifier les enregistrements.

#### Modification de plusieurs



#### Modification d’une valeur



### Sélection

#### La classe « muuska \dao\util\SelectionConfig »

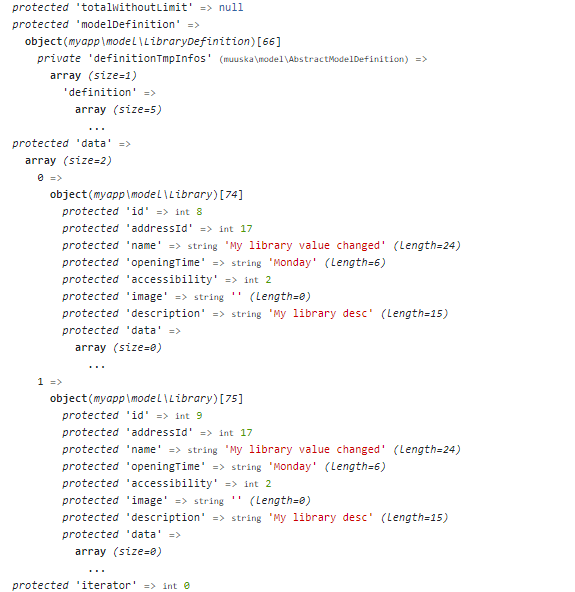
Cette classe permet de définir la façon dont les données seront récupérées (les critères de sélection, les critères de trie, la limite de données à récupérer).

Le tableau suivant décrit les attributs de la classe « **muuska\ \dao\util\SelectionConfig** »

|  |  |  |  |
| --- | --- | --- | --- |
| Attribut | Description | Type de donnée | défaut |
| **langEnabled** | Indique si l’on veut récupérer les valeurs des champs traductibles | booléen | true |
| **allLangsEnabled** | Indique si l’on veut récupérer les valeurs des champs traductibles dans toutes les langues disponibles. | booléen | false |
| **lang** | La langue dans laquelle les données seront récupérées  (Si Le Model concerné prend en compte le multilingue) | Chaine |  |
| **dataCountingEnabled** | Indique si l’on veut aussi récupérer le nombre total de données. | booléen | false |
| **start** | L’index où la lecture de données doit commencer. (Le premier index est 0). | entier | 0 |
| **limit** | Le nombre maximum de données à récupérer. Si ce champ n’est pas spécifié, toutes les données seront récupérées. | entier | 0 |
| **logicalOperator** | L’opérateur logique (**AND** ou **OR**) à utiliser entre les restrictions. La classe « **muuska\constants\operator\ LogicalOperator** » possède les constantes à utiliser pour cet attribut. | entier | 1 :  AND\_ |
| **virtualDeletedEnabled** | Indique si les éléments qui ont été supprimé virtuellement seront sélectionnés. | booléen | false |
| **onlyActive** | Indique si l’on veut uniquement les éléments actif (Uniquement si le Model à un attribut « **active** »). | booléen | false |
| **restrictionFields** | Liste des champs où sont appliquées les restrictions.  Les éléments de ce tableau sont de type « **muuska\dao\util\FieldRestriction** ». | Tableau |  |
| **sortOptions** | Liste des champs où sont configurés le options de trie.  Les éléments de ce tableau sont de type « **muuska\dao\util\SortOption**. | Tableau |  |
| **selectionAssociations** | Liste des champs où sont configurées les options de sélection de données associées.  Les éléments de ce tableau sont des objets de la classe «**muuska\dao\util\SelectionAssociation**». | Tableau |  |

#### Sélection standard

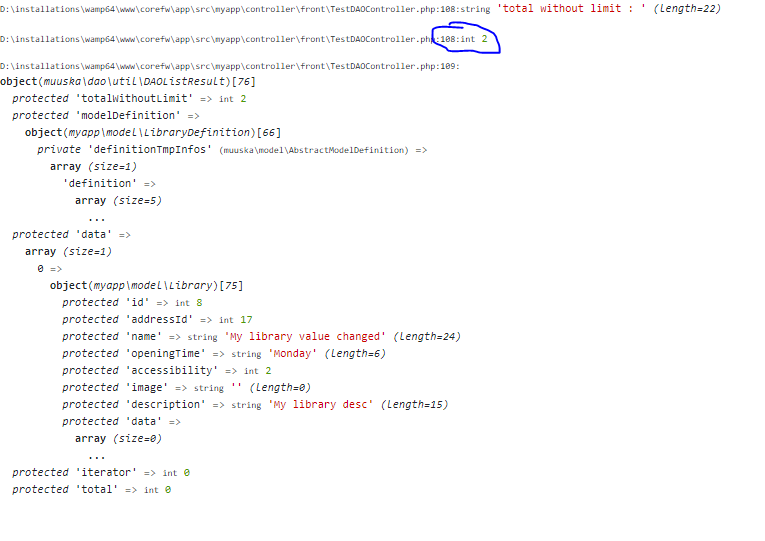




#### Sélection avec nombre total de données



**Résultat**



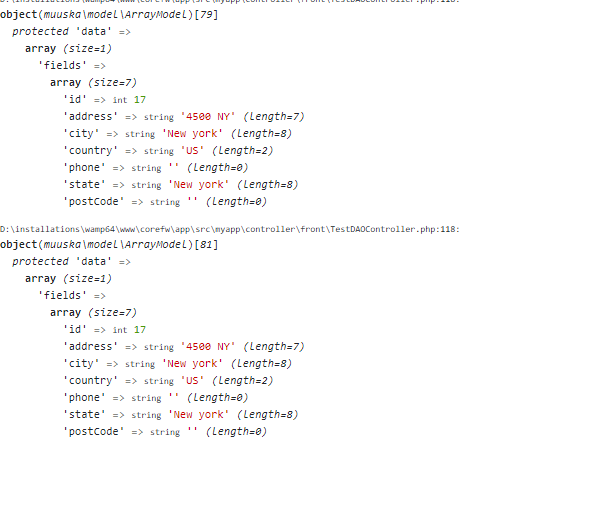
#### Récupération des informations sur les objets associés au Model

Voici une liste des méthodes de la classe « **core\utilities\dao\SelectionConfig** » qui permet de gérer les objets associés à un Model.

|  |  |  |  |
| --- | --- | --- | --- |
| Méthode | Description | Paramètres | Type de retour |
| **addSelectionAssociation** | Ajoute une configuration d’association à la liste (**selectionAssociations)** | - **selectionAssociation**: c’est un objet de la classe «**muuska \dao\util\SelectionAssociation**».  - **key** (facultatif) : la clé de l’objet dans la liste. Si cette valeur n’est pas renseignée, le nom du champ concerné sera utilisé comme clé. |  |
| **hasAssociation** | Vérifie si une clé existe dans la liste des associations. | - **key** : la clé recherchée. | booléen |
| **getSelectionAssociationByKey** | Retourne une association en fonction d’une clé.  Si la clé cherchée n’existe pas, « **null** »sera retourné. | - **key** : la clé recherchée.  - **getNewIfNotExist** (facultatif) : Permet d’indiquer si l’on veut qu’un nouvel objet soit retourné au cas où celui cherché n’existe pas. (Valeur par défaut : « **false** »).  - **addToListIfNotExist** (facultatif) : Permet d’indiquer si l’on veut que le nouvel objet soit créé au cas où celui cherché n’existe pas soit ajouté à la liste. (Valeur par défaut : « **false** »). | SelectionAssociation |
| **setSelectionAssociationParams** | Ajoute ou Modifie (Si elle existe déjà) une association à partir des paramètres. | - **fieldName** : le nom du champ sur lequel il y-a la référence.  - **joinType** (facultatif) : Le type de jointure à utiliser. La classe « **muuska\dao\constants\ JoinType**» possède les constantes à utiliser pour ce paramètre.  (Valeur par défaut : **JoinType::LEFT**).  - **retrievingEnabled** (facultatif) : Indique si l’on veut récupérer ou pas les infos de l’objet associé.  (Valeur par défaut : **true**).  - **langEnabled** (facultatif) : Indique si l’on veut récupérer les valeurs des champs traductibles.  (Valeur par défaut : **true**).  - **langEnabled** (facultatif) : Indique si l’on veut récupérer les valeurs des champs traductibles dans toutes les langues disponibles.  (Valeur par défaut : **false**). |  |

**Exemple**





#### Trie des données

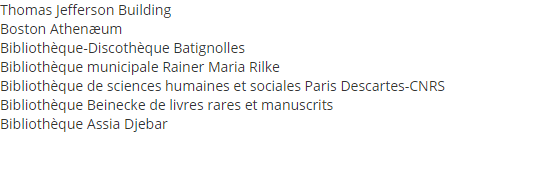
Voici une liste des méthodes de la classe « **muuska\dao\util\SelectionConfig** » qui permet de gérer le trie.

|  |  |  |  |
| --- | --- | --- | --- |
| Méthode | Description | Paramètres | Type de retour |
| **addSortOption** | Ajoute une option de trie à la liste (**sortOptions)** | - **sortOption**: c’est un objet de la classe «**muuska\dao\util**\**SortOption**».  - **key** (facultatif) : la clé de l’objet dans la liste. Si cette valeur n’est pas renseignée, le nom du champ concerné sera utilisé comme clé. |  |
| **hasSortOption** | Vérifie si une clé existe dans la liste des options de trie. | - **key** : la clé recherchée. | booléen |
| **getSelectionSortByKey** | Retourne une option de trie en fonction d’une clé.  Si la clé cherchée n’existe pas, « **null** »sera retourné. | - **key** : la clé recherchée.  - **getNewIfNotExist** (facultatif) : Permet d’indiquer si l’on veut qu’un nouvel objet soit retourné au cas où celui cherché n’existe pas. (Valeur par défaut : « **false** »).  - **addToListIfNotExist** (facultatif) : Permet d’indiquer si l’on veut que le nouvel objet soit créé au cas où celui cherché n’existe pas soit ajouté à la liste. (Valeur par défaut : « **false** »). | SortOption |
| **setSortOptionParams** | Ajoute ou Modifie (Si elle existe déjà) une option de trie à partir des paramètres. | - **fieldName** : le nom du champ sur lequel on veut que le trie soit basé.  - **way** (facultatif) : Le sens du trie (**ASC** ou **DESC**). La classe « **muuska\dao\constants\dao\SortDirection**» possède les constantes à utiliser pour ce paramètre.  (Valeur par défaut : **SortDirection::DESC**).  - **key** (facultatif) : la clé de l’objet dans la liste. Si cette valeur n’est pas renseignée, le nom du champ concerné sera utilisé comme clé. |  |

#### Trie standard



**Résultat**



#### Trie en Utilisant les champs des objets associés

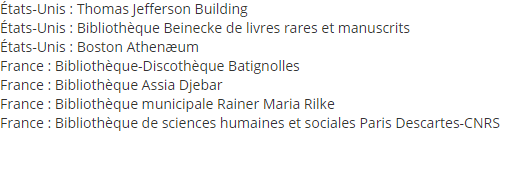
Pour le faire, il faut crée une instance de la classe «**muuska\dao\util\SortOption**» ; ensuite donner la valeur « **true** » à l’attribut « **foreign** » ; puis renseigner l’attribut « **externalField** » avec le nom du champ externe (par exemple ‘country’).

Une fois l’objet de type « **SortOption**» créé, il faut l’ajouter à la liste des options de trie de l’objet de type « **SelectionConfig** » en utilisant la méthode « **addSortOption**».

Si dans la suite de vos traitements vous souhaitez utiliser l’objet associé en question, il faut ajouter le champ qui le référencie à la liste des associations de l’objet de type « **SelectionConfig** ». (Exemple : $selectionConfig->setSelectionAssociationParams('addressId');



**Résultat**



#### La restriction de données

Voici une liste des méthodes de la classe « **muuska\dao\util\SelectionConfig** » qui permet de gérer la restriction de données.

|  |  |  |  |
| --- | --- | --- | --- |
| Méthode | Description | Paramètres | Type de retour |
| **addRestrictionField** | Ajoute une restriction la liste (**restrictionFields)** | - **restrictionField**: c’est un objet de la classe «**muuska\dao\util**\**FieldRestriction**».  - **key** (facultatif) : la clé de l’objet dans la liste. Si cette valeur n’est pas renseignée, le nom du champ concerné sera utilisé comme clé. |  |
| **hasRestrictionField** | Vérifie si une clé existe dans la liste des restrictions. | - **key** : la clé recherchée. | booléen |
| **getRestrictionFieldByKey** | Retourne une restriction en fonction d’une clé.  Si la clé cherchée n’existe pas, « **null** »sera retourné. | - **key** : la clé recherchée.  - **getNewIfNotExist** (facultatif) : Permet d’indiquer si l’on veut qu’un nouvel objet soit retourné au cas où celui cherché n’existe pas. (Valeur par défaut : « **false** »).  - **addToListIfNotExist** (facultatif) : Permet d’indiquer si l’on veut que le nouvel objet soit créé au cas où celui cherché n’existe pas soit ajouté à la liste. (Valeur par défaut : « **false** »). | FieldRestriction |
| **setRestrictionFieldParams** | Ajoute ou Modifie (Si elle existe déjà) une restriction à partir des paramètres. | - **fieldName** : le nom du champ sur lequel sur lequel la restriction est basé.  - **value** : la valeur qui sera utilisée pour la restriction (ça peut être aussi un tableau en fonction de l’opérateur choisit).  - **operator** (facultatif) : L’opérateur à utiliser pour la restriction (**EQUALS, CONTAINS**, **START\_WITH**, **END\_WITH**, **DIFFERENT**, **BETWEEN**, **IN\_LIST**, **NOT\_IN\_LIST**). La classe « **muuska\constants\operator\Operator**» possède les constantes à utiliser pour ce paramètre.  (Valeur par défaut : **Operator::EQUALS**).  - **key** (facultatif) : la clé de l’objet dans la liste. Si cette valeur n’est pas renseignée, le nom du champ concerné sera utilisé comme clé.  - **foreign** (facultatif) : Indique qu’on fait la restriction sur un champ externe.  (Valeur par défaut : **false**)  - **externalField** (facultatif) : Nom du champ externe. |  |
| **createRestrictionFieldsFromArray** | Ajoute des restrictions à la liste à partir d’un tableau associatif. | - **fields**: tableau associatif des restrictions. |  |

#### Restriction standard



**Résultat**



#### Restriction en Utilisant les champs des objets associés

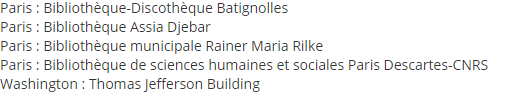
Pour le faire, il faut crée une instance de la classe «**muuska\dao\util\** **FieldRestriction**» ; ensuite donner la valeur « **true** » à l’attribut « **foreign** » ; puis renseigner l’attribut « **externalField** » avec le nom du champ externe (par exemple ‘country’).

Une fois l’objet de type « **FieldRestriction**» créé, il faut l’ajouter à la liste des restrictions de l’objet de type « **SelectionConfig** » en utilisant la méthode « **addRestrictionField**».

Si dans la suite de vos traitements vous souhaitez utiliser l’objet associé en question, il faut ajouter le champ qui le référencie à la liste des associations de l’objet de type « **SelectionConfig** ». (Exemple : $selectionConfig->setSelectionAssociationParams('addressId');



**Résultat**



#### Restriction en groupe

L’on peut se trouver dans des cas où l’on veut faire des restrictions en utilisant des opérateurs logiques différents entre certains champs. Par exemple si je veux sélectionner toutes les bibliothèques de « France » qui ouvrent le « Jeudi » et toutes les bibliothèques de la ville de « Boston » qui ouvre également le « Jeudi ».

La classe « **muuska\dao\util\FieldRestriction**» possède les attributs et méthodes suivantes pour gérer les restrictions par groupe.

**Attributs**

* **logicalOperator** : L’opérateur logique qui sera utilisé par les sous-restrictions.
* **subFields** : La liste des sous-restrictions.

**Méthodes**

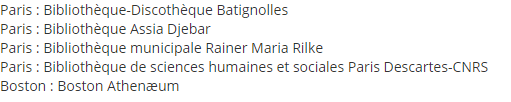
|  |  |  |  |
| --- | --- | --- | --- |
| Méthode | Description | Paramètres | Type de retour |
| **addSubField** | Ajoute un sous élément à la liste (**subFields)** | - **subField**: c’est un objet de la classe «**muuska\dao\util\** **FieldRestriction**».  - **key** (facultatif) : la clé de l’objet dans la liste. Si cette valeur n’est pas renseignée, le nom du champ concerné sera utilisé comme clé. |  |
| **hasSubField** | Vérifie si une clé existe dans la liste des sous-restrictions. | - **key** : la clé recherchée. | booléen |
| **getSubFieldByKey** | Retourne une sous-restriction en fonction d’une clé.  Si la clé cherchée n’existe pas, « **null** »sera retourné. | - **key** : la clé recherchée.  - **getNewIfNotExist** (facultatif) : Permet d’indiquer si l’on veut qu’un nouvel objet soit retourné au cas où celui cherché n’existe pas. (Valeur par défaut : « **false** »).  - **addToListIfNotExist** (facultatif) : Permet d’indiquer si l’on veut que le nouvel objet soit créé au cas où celui cherché n’existe pas soit ajouté à la liste. (Valeur par défaut : « **false** »). | FieldRestriction |
| **setSubFieldParams** | Ajoute ou Modifie (Si elle existe déjà) une sous-restriction à partir des paramètres. | - **fieldName** : le nom du champ sur lequel sur lequel la restriction est basé.  - **value** : la valeur qui sera utilisée pour la restriction (ça peut être aussi un tableau en fonction de l’opérateur choisit).  - **operator** (facultatif) : L’opérateur à utiliser pour la restriction (**EQUALS, CONTAINS**, **START\_WITH**, **END\_WITH**, **DIFFERENT**, **BETWEEN**, **IN\_LIST**, **NOT\_IN\_LIST**). La classe « **muuska\constants\operator\ Operator**» possède les constantes à utiliser pour ce paramètre.  (Valeur par défaut : **Operator::EQUALS**).  - **key** (facultatif) : la clé de l’objet dans la liste. Si cette valeur n’est pas renseignée, le nom du champ concerné sera utilisé comme clé.  - **foreign** (facultatif) : Indique qu’on fait la restriction sur un champ externe.  (Valeur par défaut : **false**)  - **externalField** (facultatif) : Nom du champ externe. |  |
| **createSubFieldsFromArray** | Ajoute des sous-restrictions à la liste à partir d’un tableau associatif. | - **fields**: tableau associatif des restrictions. |  |

**Remarque** : Vous pouvez ajouter des sous-restrictions à une sous-restriction autant de fois que nécessaire en fonction de votre besoin (ça fonctionne comme un arbre).

Voici le code qui va résoudre notre cas évoqué plus haut :

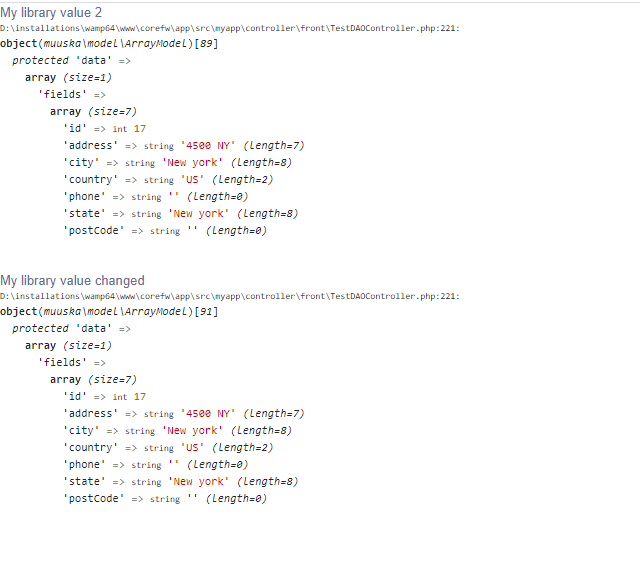


**Résultat**



#### Restriction en utilisant les fonctions





### Suppression

#### Standard



#### Suppression multiple



### Méthodes communes

Toutes les méthodes utilisées pour la restriction avec le **SelectionConfig** peuvent aussi être utilisé pour les mises à jour avec le **SaveConfig** et aussi pour les suppressions avec le **DeleteConfig**.

## Utiliser une autre source d’accès aux données

## Implémentation d’un nouveau support d’accès aux données

## Enregistrer un nouveau support d’accès au données

# Back office

## Initialisation

### Configuration

Pour ajouter le Back office, ouvrez le fichier « **storage/config/sub\_apps.json** » et ajouter sa configuration.



### Création de la sous application pour le Back office

A la racine de votre dossier **src**, Ajoutez la classe **AdminSubApplication.**

Elle doit Hériter de la classe **\muuska\project\AbstractSubApplication.**



### Association du back office à la classe que vous venez de créer

Ouvrez votre classe **MyApp** et ajouter les instructions suivantes dans la méthode **createSubProject**



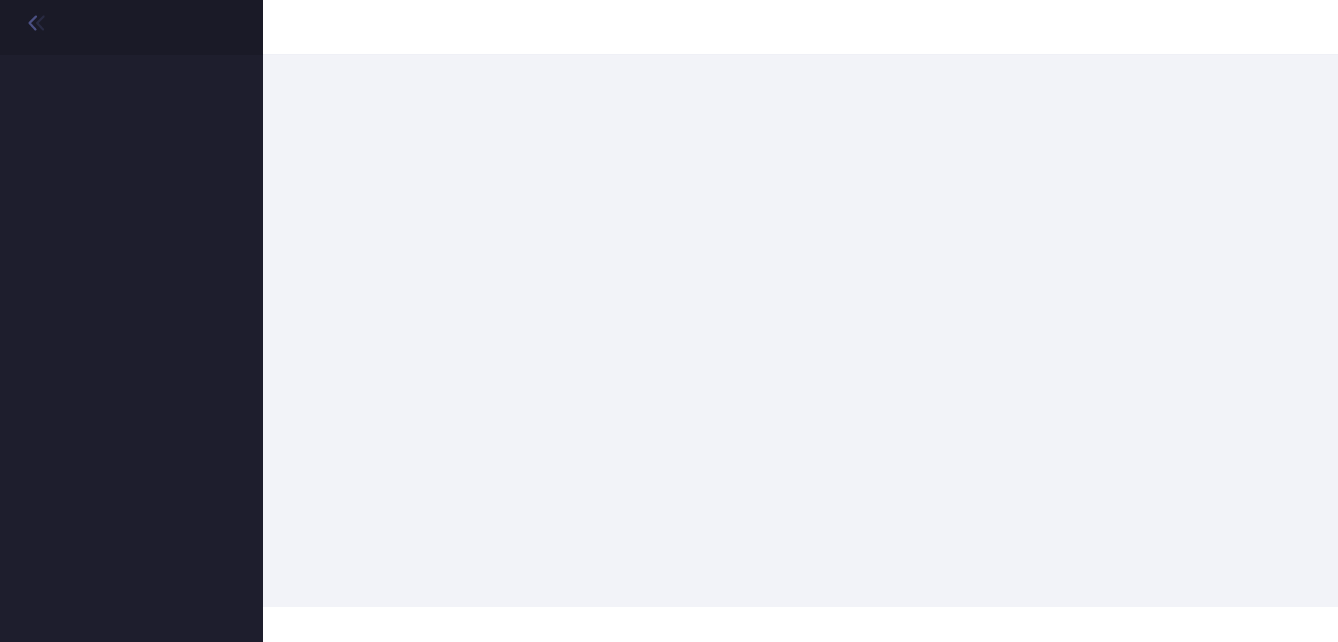
Votre back office est enfin prêt pour y accéder entrez l’url <http://localhost/muuska/admin/en>

### Configuration du thème pour le back office

Ouvrez le dossier « **storage/config/themes/default** » et ajouter un fichier « **admin.json** » avec le contenu suivant :



Ensuite actualisez la page et vous verrez le changement



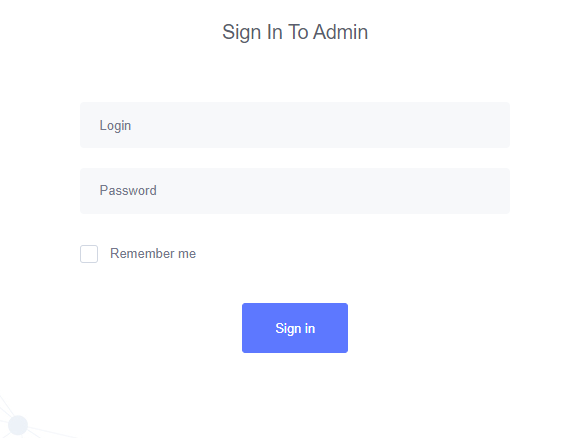
### Activer l’authentification

Pour activer l’authentification, ouvrez le fichier « **storage/config/sub\_apps.json** »

Et ajouter le contenu suivant :



Si vous actualiser la page, vous serez redirigé sur la page d’authentification



Les accès de test sont

* Login : [demo@demo.com](mailto:demo@demo.com)
* Password : **demo**

### Activer la vérification des droits d’accès

Pour activer la vérification des droits d’accès, ouvrez le fichier « **storage/config/sub\_apps.json** » et ajoutez le contenu suivant :



# Les projets

Un projet est un composant qui fournit un ensemble de fonctionnalités utilisable par l’application finale.

Les fonctionnalités attendues d’un projet sont définit dans l’interface « **\muuska\project\Project** ». La classe abstraite « **\muuska\project\AbstractProject** » fournit une implémentation standard de la plupart de ces fonctionnalités.

On distingue 3 types de projets (la classe **\muuska\project\constants\ProjectType** contient la liste des types de projet) :

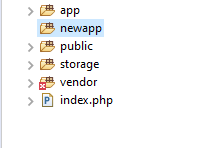
## L’application

C’est le projet principal. Il définit toutes la logique de système finale. Il doit implémenter l’interface **\muuska\project\Application** qui définit les fonctionnalités du système. La classe « **\muuska\project\AbstractApplication** » qui hérite de la classe **AbstractProject** fournir une implémentation standard de la plupart des fonctionnalités de l’interface **Application**. Donc la classe **MyApp** qui utilisé depuis le début est l’application principale.

### Création d’une nouvelle application.

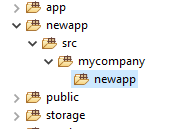
#### Etape 1 : création du dossier de l’application

Allez à la racine de votre projet et créer un nouveau dossier pour votre application



#### Etape 2 : création de la classe principale

Créez le sous dossier « **src** » puis ajouter à l’intérieur les sous dossier qui seront utilisé comme namespace de base de votre application (Il en faut au moins 1 sous dossier).



Créer une classe qui hérite de la classe **AbstractApplication**



Si vous utiliser un IDE il vous signalera qu’il y-a une méthode non implémenté : il s’agit de la méthode « **createAppSetup** » donc le rôle est de créer le setup de votre application.

#### Etape 3 : création du setup

Créer une nouvelle classe qui doit hériter de la class « **\muuska\project\setup\AbstractProjectSetup** » et ajouter le constructeur suivant :

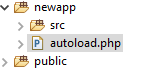


Maintenant vous pouvez implémenter la méthode « **createAppSetup** » dans votre Application



#### Etape 4 : création de l’autoloader

Créez un fichier **autoload.php** à la racine du dossier de votre nouvelle application



Contenu du fichier



La méthode « **registerNew** » de la classe « **\muuska\util\DefaultAutoloader** » prend comme premier paramètre le namespace de base de votre nouveau projet.

#### Etape 5 : modifier le fichier index.php

Ouvrez le fichier « index.php » principal et insérez le code suivant :



Si vous actualisé la page, vous aurez le message suivant « **Application not implemented** » car aucune sous application n’a été implémenté.

#### Etape 6 : création d’une sous application

Créer une classe qui herite de la classe **\muuska\project\AbstractSubApplication** :



#### Etape 7 : association de la sous application à l’application principale

Pour le faire, ouvrez la classe principale de votre application et redéfinissez la méthode « **createSubProject** » :



Si vous actualisé la page, vous aurez maintenant une page blanche.

### Obtention de l’instance de l’application depuis une autre classe

Pour obtenir l’instance de l’application à n’importe quel endroit de votre code, utilisez l’instruction suivante : **App::getApp()** . La classe **App** se trouve dans le namespace « **\muuska\util** ».

### Méthodes utiles de l’interface Application

#### getBaseUrl

Retourne l’URL de base de l’application

#### createUrl

Crée une nouvelle URL. Elle prend en paramètre un objet de type **\muuska\url\UrlCreationInput**.

#### getPublicUrl

Retourne l’URL de base du dossier public.

#### getUploadTmpUrl

Retourne l’URL de base du dossier d’upload temporaire.

#### getUploadTmpFullUrl

Retourne l’URL complète d’un fichier temporaire. Elle prend en paramètre de nom du fichier.

#### getModelFileUrl

Retourne l’URL du fichier d’un model. Elle prend en paramètre la définition du model, le model en question et le nom du champ au quel le ficher est associé.

#### getRootDir

Retourne le chemin physique du dossier racine de l’application.

#### getPublicDir

Retourne le chemin physique du dossier public de l’application.

#### getStorageDir

Retourne le chemin physique du dossier des stockages.

#### getRootConfigDir

Retourne le chemin physique du dossier des configurations.

#### getCacheDir

Retourne le chemin physique du dossier de cache.

#### getLibrariesDir

Retourne le chemin physique du dossier des bibliothèques (dossier **vendor**).

#### getUploadTmpDir

Retourne le chemin physique du dossier d’upload temporaire de l’application.

#### getUploadTmpFullFile

Retourne le chemin physique complet d’un fichier temporaire.

#### getModelFileDir

Retourne le chemin physique du dossier de stockage des fichiers d’un model. Elle prend en paramètre la définition du model et le nom du champ au quel le dossier est associé.

#### getModelFullFile

Retourne le chemin physique complet du fichier d’un model. Elle prend en paramètre la définition du model, le model en question et le nom du champ au quel le ficher est associé.

#### getLanguages

Retourne la liste des langues disponible. Elle prend en paramètre un booléen pour specifier si seul les langues actives seront retournées. Cette fonction retourne un tableau où chaque élément est une instance de l’interface **\muuska\localization\LanguageInfo**.

#### getLanguageInfo

Retourne une instance de la langue en fonction du code. Si aucune instance n’est trouvée, elle retournera « **null** ». L’instance retournée est de type **\muuska\localization\LanguageInfo.**

#### getDefaultLang

Retourne le code de la langue par défaut.

#### getDefaultMailSender

Retourne l’objet qui s’occupe de l’envoi de mail. Cette méthode retourne une instance de la classe **\muuska\mail\MailSender**.

#### isDevMode

Retourne un booléen pour indiquer si le mode développeur est activé ou pas.

#### getEnabledSubApplications

Retourne les noms des sous applications activé.

#### getSubApplicationConfig

Retourne la configuration d’une sous application en fonction de son nom. Elle retourne « **null** » si la sous application n’existe pas. Cette méthode retourne une instance de la classe **\muuska\config\Configuration**.

#### getCacheManager

Retourne une instance du système de gestion de cache par default. L’objet retourné est de type **\muuska\cache\CacheManager**.

#### getEventTrigger

Retourne une instance de l’exécuteur d’évènement.

#### registerDAOSource

Enregistrer un nouveau support d’accès aux données.

#### getMainConfiguration

Retourne la configuration principale de l’application. Cette méthode retourne une instance de la classe **\muuska\config\Configuration**.

#### registerConfiguration

Enregistre une nouvelle configuration afin qu’elle puisse être utilisé par d’autre processus. Elle prend en paramètre le nom de la configuration, et l’instance de la configuration (**\muuska\config\Configuration**).

#### hasConfiguration

Vérifie si une configuration a été enregistrée.

#### getConfiguration

Retourne l’instance d’une configuration enregistrée. Elle retourne « **null** » si la configuration n’existe pas.

#### hasSubApplication

Vérifié si l’instance d’un sous application existe. Elle prend en paramètre le nom de la sous application.

#### getSubApplication

Retourne l’instance d’une sous application. Elle prend en paramètre le nom de la sous application. Elle retourne « **null** » si l’instance n’existe pas.

#### getProject

Retourne l’instance d’un projet. Elle prend en paramètre le type du projet et son nom du projet. Si le type est APPLICATION ou FRAMEWORK, vous devez donner la valeur « **null** » au paramètre « **name** ». Elle retourne « **null** » si l’instance n’existe pas.

#### getModuleInstance

Retourne l’instance d’un module. Elle prend en paramètre le nom du module. Elle retourne « **null** » si l’instance n’existe pas.

## Le framework

C’est un projet donc le rôle principal est de fournir les fonctionnalités de base du système. (Gestion des droits d’accès, authentification, gestion des modules, traduction de base, CSS et JS de base, etc..).

Pour obtenir l’instance du framework, utilisez l’instruction **App::getFramework().**

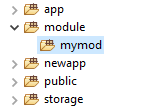
## Les modules

Ce sont les projets dont le type est **\muuska\project\ProjectType::MODULE**.

### Création d’un module

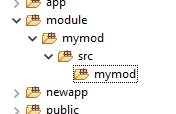
#### Etape 1 : création du dossier du module

Allez à la racine et créer un le dossier « **module** » s’il n’existe pas ensuite créer un nouveau dossier pour votre module (Le nom du dossier doit être en minuscule, sans espace et sans caractère accentué) le nom du dossier sera aussi le nom de votre module.



#### Etape 2 : création de la classe principale

Créez le sous dossier « **src** » puis ajouter à l’intérieur les sous dossier qui seront utilisé comme namespace de base de votre application (Il en faut au moins 1 sous dossier).



Créer une classe qui hérite de la classe **AbstractProject**



#### Etape 3 : création du setup

Créer une nouvelle classe qui doit hériter de la classe « **\muuska\project\setup\AbstractProjectSetup** » et ajouter le constructeur suivant :

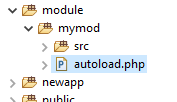


NB : l’instruction suivante permet de créer une nouvelle instance de votre projet



#### Etape 4 : création de l’autoloader

Créez un fichier **autoload.php** à la racine du dossier de votre module



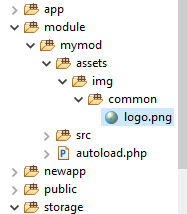
Contenu du fichier



La méthode « **registerNew** » de la classe « **\muuska\util\DefaultAutoloader** » prend comme premier paramètre le namespace de base de votre nouveau projet.

#### Etape 5 : Ajout du logo du module

A la racine de votre module, ajouter un dossier « **assets** » puis dans ce dossier créer un dossier « **img** » à l’intérieur du dossier « **img** » ajoutez un sous dossier « **common** » puis ajoutez un fichier « **logo.png** ».



### Déploiement d’un module

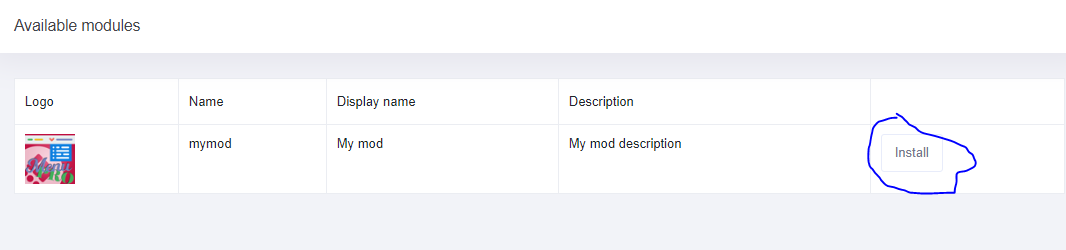
Vous devez ajouter le setup du module à la liste des setups de l’application pour le faire, ouvrez le fichier « **storage/setup/module.json** » et ajoutez la classe du setup de votre module :



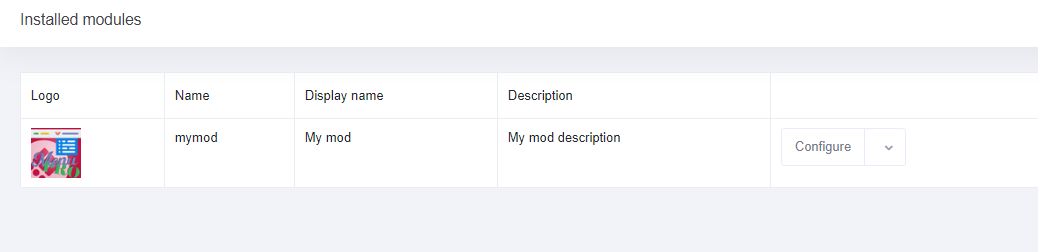
### Installation d’un module

Pour installer un module, ouvrez la page d’installation des modules en accédant au contrôleur « **module-installer** ». Dans notre cas L’URL complète sera la suivante : <http://localhost/muuska/admin/en/module-installer>.

Vous verrez la liste des modules disponibles. Cliquez sur le bouton « **install** » pour lancer l’installation d’un module.

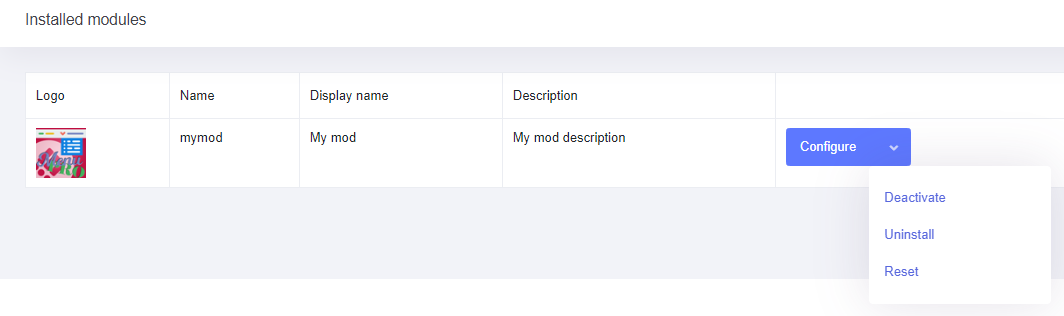


Après l’installation vous serez redirigé sur la page d’administration des modules installé :



### Administration des modules installés

Pour administrer un module, ouvrez la page d’administration des modules en accédant au contrôleur « **module-manager** ». Dans notre cas L’URL complète sera la suivante : <http://localhost/muuska/admin/en/module-manager>.



Pour chaque module installé, vous avez les options suivantes :

#### Configure

Permet de d’accéder à la page d’administration du module.

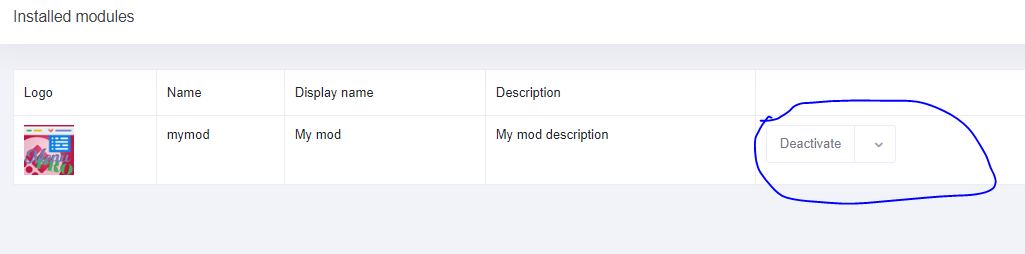
Cette option n’est visible qu’au cas où le module est configurable.

Si vous cliquez sur configure vous aurez le message « **Controller not found** » car vous n’avez pas encore créé le contrôler de configuration (**home**) de votre module.

Pour rendre un module non configurable, ouvrez la classe du setup du module et redéfinissez la méthode « **isConfigurable** » en retournant la valeur **false** :



**Résultat**



#### Activate

Active le module. Cette option n’est visible qu’au cas où le module a été désactivé.

#### Deactivate

Désactive le module. Cette option n’est visible qu’au cas où le module est activé.

#### Uninstall

Désinstalle le module.

#### Reset

Réinitialise le module (Le désinstalle puis le réinstalle).

## Méthodes de l’interface \muuska\project\Project

### getName

Retourne le nom du projet.

### getType

Retourne le type du projet.

### getVersion

Retourne la version du projet.

### hasSubProject

Vérifie si le projet à une instance pour la sous application dont le nom est fourni en paramètre.

### getSubProject

Retourne une instance du sous projet en fonction du nom de la sous application fourni en paramètre. Elle retourne « **null** » si l’instance n’existe pas.

### getInstalledInfo

Retourne un objet contenant les informations d’installation du projet. Elle retourne « **null** » si le projet n’est pas installé. Cette methode retoune un objet de type **\muuska\projet\ProjectInfo**.

### isInstalled

Vérifie si le projet est installé ou pas.

### isActive

Vérifie si le projet est activé ou pas.

### getCorePath

Retourne le chemin relatif où le projet a été déployé.

### getCoreDir

Retourne le chemin physique complet du projet.

### getTranslator

Retourne un traducteur du projet en fonction de la configuration fournit en paramètre. Elle prend en paramètre une instance de l’interface **\muuska\translation\config\TranslationConfig** et une instance de l’interface **\muuska\translation\Translator** qui sera utilisé comme traducteur alternatif (ce paramètre est facultatif). Elle retourne une instance de l’interface **\muuska\translation\Translator**.

### createAssetGroup

Créé un groupe d’asset en fonction du nom fournit en paramètre et l’ajoute à la liste des assets. Elle retourne **null** si elle ne reconnait pas le nom du groupe qui a été donné en paramètre. Elle retourne on objet de type **\muuska\asset\AssetGroup**.

### createAsset

Crée un asset relatif. Elle prend en paramètre le type de l’asset (les types disposible sont dans la classe \muuska\asset\constants\AssetType), le chemin relatif de l’asset. Elle retourne un objet de type **\muuska\asset\RelativeUriAsset**.

### createHtmlImage

Crée une image html relative en fonction du chemin relatif fournit en paramètre. Elle retourne un objet de type **\muuska\html\RelativeHtmlImage** qui est un composant HTML.

### createTemplate

Crée un template en fonction du chemin relatif fourni en paramètre. Elle retourne un objet de type **\muuska\renderer\template\Template**.

### createJSTranslation

### createResourceTree

# Les sous projets

Un sous projet est un composant qui fournit un ensemble de fonctionnalités utilisable par une sous application (Font office, Back office, API, etc…).

Les fonctionnalités attendus d’un sous projet sont définit dans l’interface « **\muuska\project\SubProject** ». La classe abstraite « **\muuska\project\AbstractSubProject** » fournit une implémentation standard de la plupart de ces fonctionnalités.

## Les méthodes de l’interface **SubProject**

### getTranslator

Retourne un traducteur du sous projet en fonction de la configuration fournit en paramètre. Elle prend en paramètre une instance de l’interface **\muuska\translation\config\TranslationConfig** et une instance de l’interface **\muuska\translation\Translator** qui sera utilisé comme traducteur alternatif (ce paramètre est facultatif). Elle retourne une instance de l’interface **\muuska\translation\Translator**.

### createAssetGroup

Créé un groupe d’asset en fonction du nom fournit en paramètre et l’ajoute à la liste des assets. Elle retourne **null** si elle ne reconnait pas le nom du groupe qui a été donné en paramètre. Elle retourne on objet de type **\muuska\asset\AssetGroup**.

### createAsset

Crée un asset relatif. Elle prend en paramètre le type de l’asset (les types disponibles sont dans la classe \muuska\asset\constants\AssetType), le chemin relatif de l’asset. Elle retourne un objet de type **\muuska\asset\RelativeUriAsset**.

### createHtmlImage

Crée une image html relative en fonction du chemin relatif fournit en paramètre. Elle retourne un objet de type **\muuska\html\RelativeHtmlImage** qui est un composant HTML.

### createTemplate

Crée un template en fonction du chemin relatif fourni en paramètre. Elle retourne un objet de type **\muuska\renderer\template\Template**.

### createController

Crée un contrôleur en fonction de l’input fournit en paramètre. Elle prend en paramètre un objet de type \**muuska\controller\ControllerInput** et retourne un objet de type **\muuska\controller\Controller**. Elle retourne « **null** » si le contrôleur n’existe pas.

## Création d’un sous projet

### Création de la classe du sous projet

Créer une classe qui hérite de la classe **\muuska\project\AbstractProject**



### Association de la classe du sous projet au projet

Ouvrez la classe principale de votre projet et redéfinissez la méthode **createSubProject**.



NB : Si votre projet est l’application principale votre sous projet doit plutôt hériter de la classe **\muuska\project\AbstractSubApplication** qui hérite de la classe **AbstractSubProject** et implémente l’interface **\muuska\project\SubApplication**.

## Les sous applications

Une sous application est une partie de l’application. Vous pouvez créer autant de sous application que vous souhaitez (Front office, Back office, API, etc…). C’est la sous application qui définit la logique de base des fonctionnalités publiques du système (Exécution des contrôleurs, gestion des utilisateurs et droit d’accès, création du thème courant). Une sous application est un sous projet avec des fonctionnalités supplémentaires. L’interface **\muuska\project\SubApplication** propose les fonctionnalités requise par une sous application. La classe **\muuska\project\AbstractSubApplication** fourni des implémentations standard pour la plupart de ces fonctionnalités.

## Création d’un contrôleur

La procédure est la même que celle utilisé depuis le début.

Vous créez le sous projet auquel le contrôleur sera associé s’il n’existe pas :



Ensuite vous associé le sous projet à votre projet dans la méthode « **createSubProject** » :



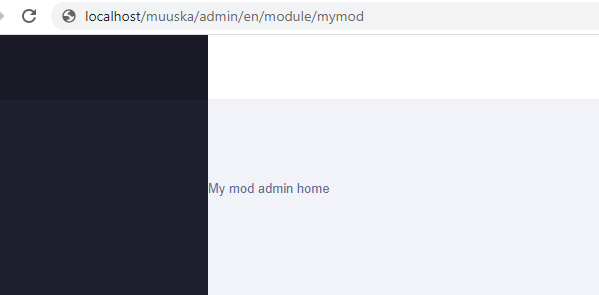
Ensuite vous créez une classe pour votre contrôleur :



Enfin redéfinissez la méthode « **createController** » dans la classe de votre sous projet :



Ouvrez à nouveau le gestionnaire de module et cliquez sur le bouton « **Configure** » de votre module et vous aurez l’affichage suivant :



## Url permettant d’accéder au contrôleur d’un module

L’URL permettant l’accès à un contrôleur doit être sous la forme {**baseUrl**}/ {**subAppUrlPath**}{**lang**}/module{**moduleName**}/{**controllerName**}/{**action**}

* {**moduleName**} : représente le nom du module.

Exemple d’URL http:localhost/muuska/en/module/mymod,

http:localhost/muuska/en/module/mymod/hello-world,

http:localhost/muuska/en/module/mymod/hello-world/goodbye,

http:localhost/muuska/admin/en/module/mymod

https:www.domain.com/en/module/mymod/hello-world/goodbye,

https:www.api.domain.com/en/module/mymod/hello-world/goodbye,

# Internationalisation

## Traducteur

Un traducteur est un objet permettant de traduire les textes dans n’importe quelle langue. C’est un objet de type **\muuska\translation\Translator**. Cet objet a une seule méthode : **translate** qui prend en paramètre le code de la langue, la chaine à traduire et le contexte (Chaine de caractère) dans lequel on souhaite traduire cette chaine (Le contexte est facultatif).

La classe **\muuska\translation\DefaultTranslator** implémente l’interface **\muuska\translation\Translator** et fourni une implémentation standard de la méthode translate. Cette classe a besoin d’un objet de type **\muuska\translation\loader\TranslationLoader** qui lui permettra de charger les textes en fonction d’une langue. Le constructeur de cette classe prend également un second paramètre (Qui est facultatif) représentant le traducteur alternatif : c’est un objet de type **\muuska\translation\Translator** (Il est utilisé au cas où la **TranslationLoader** n’arrive pas à trouver un texte).

Le **TranslationLoader** par défaut est le JSON la classe concerné est **\muuska\translation\loader\source\JSONTranslationLoader** elle prend en paramètre le chemin du fichier où se trouve les textes. Ce chemin est sous la forme « dossier1/dossier2/**{lang}/**fichier » (**{lang}** n’est pas une variable vous devez l’écrire exactement comme ça car au moment de charger les textes, il sera remplacé par la langue approprié). Vous ne devez pas ajouter le suffixe « **.json** » car il sera ajouté automatiquement.

### Exemple

Nous allons créer un nouveau contrôleur « test-translation » dans l’application **MyApp** pour tester les traductions.

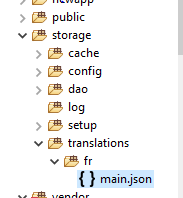
#### Création du contrôleur



Si vous testez vous verrez le message « **Hello world** » affiché.

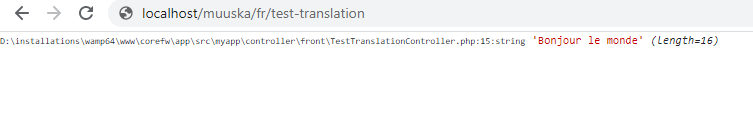
#### Création du fichier de traduction

Aller dans le dossier « **storage/translations** » depuis la racine. Et crée un dossier « **fr** » à l’intérieur de ce dossier, ajoutez un fichier **main.json**.





Modifier la langue dans l’URL puis testez à nouveau. (<http://localhost/muuska/fr/test-translation>).



#### Utilisation du contexte

Pour un même texte, vous pouvez avoir plusieurs traductions en fonction du contexte.

Nous allons modifier le contrôleur test-translation et spécifier le contexte lors de la traduction.

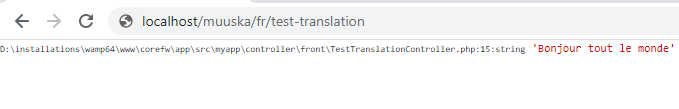


Si vous actualisez la page vous verrez toujours le texte « Bonjour le monde » car dans le fichier de traduction ce contexte n’existe pas.

Ajout du contexte dans le fichier de traduction



Lorsque vous utilisez les contextes, vous devez spécifier dans le fichier un contexte vide qui correspondra au contexte par défaut ; ainsi si le texte n’a pas de traduction dans un contexte, c’est lui que sera utilisé.



### Multiple Loader

Un multiple loader est un objet (de type **\muuska\translation\loader\MultipleLoader**) de permettant de charger les textes par groupe.

Cette interface possède la méthode « **getLoader** » qui retourne un objet de type TranslationLoader en fonction de la clé (nom du groupe) fourni en paramètre.

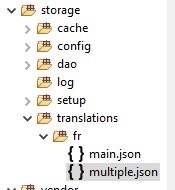
La classe **\muuska\translation\loader\DefaultMultipleLoader** fourni une implémentation standard de cette interface. Son constructeur prend en paramètre un objet de type **\muuska\translation\loader\TranslationLoader**.

#### Utilisation

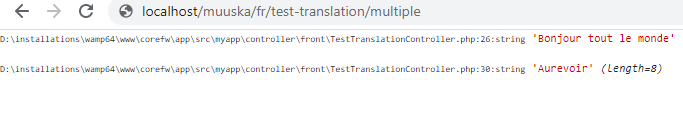
Nous allons créer une nouvelle action (**multiple**) dans notre contrôleur :



#### Création du fichier de traduction

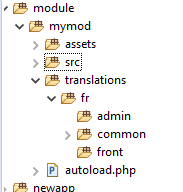






## Traducteur de projet

Tous les projets offres un moyen de récupérer l’instance d’un traducteur en fonction de la configuration fourni en paramètre. Les textes doivent être dans le dossier « **translations** » qui se trouve à la racine du projet concerné (S’il, n’existe pas il faut le Crée). A l’intérieur du dossier translation vous devez créer un sous pour chaque langue (**fr**, **en**, etc…). Dans le dossier de chaque langue, vous devez créer un sous dossier pour chaque sous application (Le nom du dossier doit correspondre au nom de la sous application en minuscule par exemple **admin**, **front**). Vous devez aussi créer un sous **common** pour les traductions qui ne sont pas liées à une sous application.



Vous pouvez récupérer l’instance d’un traducteur à partir de l’instance d’un projet ou d’un sous projet grâce à la méthode « **getTranslator** ». Cette méthode prend en paramètre un la configuration du traducteur qui est un objet de type **\muuska\translation\config\TranslatorConfig**.

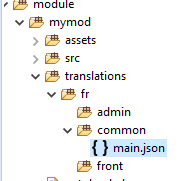
La configuration de traducteur permet d’indiquer au projet ou au sous projet comment il doit s’y prendre pour créer l’instance du traducteur. Il existe plusieurs types de configurations :

### Configuration principale

Permet de récupérer les traductions principales d’un projet ou d’un sous projet. La classe concernée est **\muuska\translation\config\MainTranslationConfig**.

Le nom du fichier doit être « **main** ».

#### Création du fichier de traduction

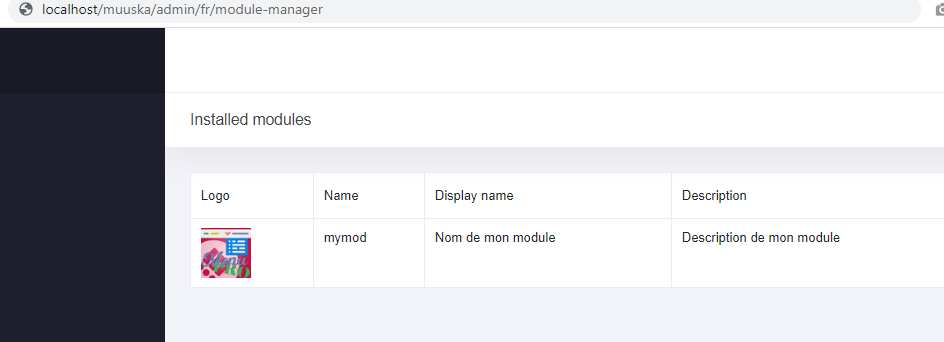


#### Contenu du fichier de traduction



#### Affichage

Si vous lancez la page « **module-manager** » en français (<http://localhost/muuska/admin/fr/module-manager>), vous verrez que les textes de votre module ont changés.

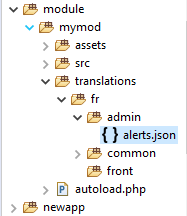


### Configuration des alertes

Permet de récupérer les traductions des alertes d’un projet ou d’un sous projet. La classe concernée est **\muuska\translation\config\AlertTranslationConfig**. Son constructeur prend en paramètre le type d’alerte.

Le nom du fichier doit être « **alerts** ». Le loader utilisé pour cette configuration est un **MultipleLoader** ce qui signifie que dans ce fichier, les textes doivent être structuré par groupe où chaque groupe où le nom du groupe correspondra au type de l’alerte.

#### Création du fichier de traduction



#### Contenu du fichier de traduction



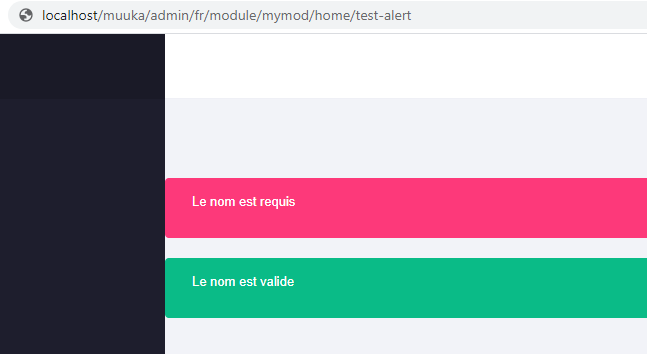
#### Utilisation au niveau du contrôleur

Modifions le contrôleur « **home** » du module « **mymod** » pour effectuer le test.



#### Affichage

Pour voir le résultat, exécutez l’action « **test-alert** » sur le contrôleur **home** de votre module (<http://localhost/muuka/admin/fr/module/mymod/home/test-alert>).

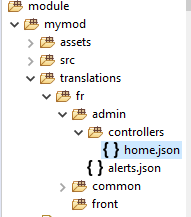


### Configuration des contrôleurs

Permet de récupérer les traductions des contrôleurs d’un sous projet. La classe concernée est **\muuska\translation\config\ControllerTranslationConfig**. Son constructeur prend en paramètre le nom du contrôleur.

Le fichier doit être dans le dossier « **controllers** » et son nom doit être le même que celui du contrôleur. Le loader utilisé pour cette configuration est un **MultipleLoader** ce qui signifie que dans ce fichier, les textes doivent être structuré par groupe où le groupe « **main** » contient les textes utilisée par la classe du contrôleur. Les groupes « **templates** » et « **js** » contiennent les textes utilisés par les templates et fichier JS. Dans les groupes « **templates »** et « **js »**, les textes sont structurés par groupe où le nom du groupe correspond au chemin relatif du fichier sans extension.

#### Création du fichier de traduction



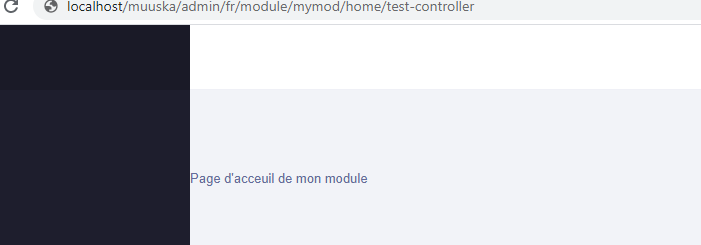
#### Contenu du fichier de traduction





Dans les contrôleurs, pour récupérer les traductions, il faut utiliser la méthode « **l** » qui prend en paramètre le texte à traduire et le contexte (facultatif).

#### Affichage



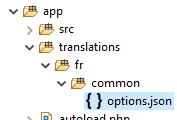
### Configuration des options

Permet de récupérer les traductions des fournisseurs d’options d’un projet ou d’un projet. La classe concernée est **\muuska\translation\config\OptionTranslationConfig**. Son constructeur prend en paramètre le nom du fournisseur d’option.

Le nom du fichier doit être « **options** ». Le loader utilisé pour cette configuration est un **MultipleLoader** ce qui signifie que dans ce fichier, les textes doivent être structuré par groupe où chaque groupe où le nom du groupe correspondra au nom du fournisseur d’option.

Nous allons faire les tests en utilisant la classe **\myapp\option\AccessibilityProvider** crée dans l’application « **myapp** ».

#### Création du fichier de traduction



#### Contenu du fichier de traduction



#### Définition du traducteur dans la classe du fournisseur l’option

Dans la classe de l’option, redéfinissez la méthode « **createTranslator** » :



La méthode « **getProjectTranslator** » prend en paramètre l’instance du projet auquel l’option appartient et le nom du fournisseur d’option.

Dans les fournisseurs d’options, pour récupérer les traductions, il faut utiliser la méthode « **l** » qui prend en paramètre le texte à traduire et le contexte (facultatif).

#### Utilisation au niveau du contrôleur



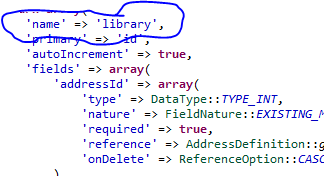
#### Affichage



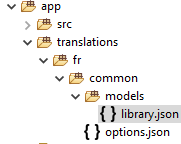
### Configuration des models

Permet de récupérer les traductions des models d’un projet ou d’un sous projet. La classe concernée est **\muuska\translation\config\ModelTranslationConfig**. Son constructeur prend en paramètre la définition du model.

Le fichier doit être dans le dossier « **models** » et son nom doit être le même que celui présent dans la définition du model.



#### Création du fichier de traduction



#### Contenu du fichier de traduction



#### Utilisation au niveau du contrôleur



#### Affichage

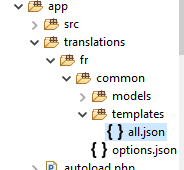


### Configuration des templates

Permet de récupérer les traductions des templates d’un projet ou d’un sous projet. La classe concernée est **\muuska\translation\config\TemplateTranslationConfig**. Son constructeur prend en paramètre le chemin relatif du fichier sans extension.

Le fichier doit être dans le dossier « **templates** **»** son nom doit être « **all** ». Le loader utilisé pour cette configuration est un **MultipleLoader** ce qui signifie que dans ce fichier, les textes doivent être structuré par groupe où chaque groupe où le nom du groupe correspondra au chemin relatif du template.

#### Création du fichier de traduction

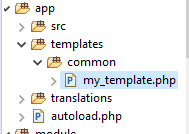


#### Contenu du fichier de traduction



#### Création du fichier du template

Créer le dossier « **templates** » à la racine de votre projet courant ; à l’intérieur de ce dossier, ajoutez un sous dossier « **common** » qui contiendra les templates qui ne sont pas liés à une sous application ; enfin, ajoutez votre fichier (**my\_template.php**).



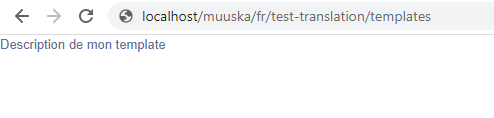


A l’intérieur des templates, vous avez un objet « $translator » qui est de type **\muuska\translation\LangTranslator**. Cet objet possède une méthode « l » qui vous permet de récupérer les traductions dans la langue courante. Cette méthode prend en paramètre le texte à traduire et le contexte (facultatif).

#### Utilisation au niveau du contrôleur



#### Affichage

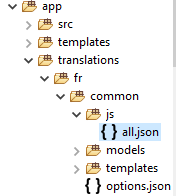


### Configuration des JS

Permet de récupérer les traductions des fichiers JavaScripts d’un projet ou d’un sous projet. La classe concernée est \muuska\translation\config\JSTranslationConfig. Son constructeur prend en paramètre le chemin relatif du fichier sans extension.

Le fichier doit être dans le dossier « **js** » son nom doit être « **all** ». Le loader utilisé pour cette configuration est un **MultipleLoader** ce qui signifie que dans ce fichier, les textes doivent être structuré par groupe où chaque groupe où le nom du groupe correspondra au chemin relatif du fichier JS.

#### Création du fichier de traduction

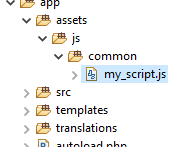


#### Contenu du fichier de traduction



#### Création du fichier JS

Créer le dossier « **assets** » à la racine de votre projet courant ; à l’intérieur de ce dossier, ajoutez un sous dossier « **js** » ; à l’intérieur du dossier « **js** », ajoutez un sous dossier « **common** » qui contiendra les fichiers **JS** qui ne sont pas liés à une sous application ; enfin, ajoutez votre fichier (**my\_script.js**).

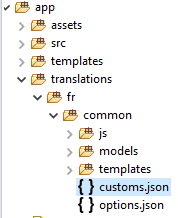


### Configuration personnalisé

Permet de récupérer les traductions personnalisées d’options d’un projet ou d’un projet. La classe concernée est **\muuska\translation\config\CustomTranslationConfig**. Son constructeur prend en paramètre le nom de la configuration personnalisée.

Le nom du fichier doit être « **customs** ». Le loader utilisé pour cette configuration est un **MultipleLoader** ce qui signifie que dans ce fichier, les textes doivent être structuré par groupe où chaque groupe où le nom du groupe correspondra au nom de la configuration personnalisée.

#### Création du fichier de traduction



#### Contenu du fichier de traduction



#### Utilisation au niveau du contrôleur



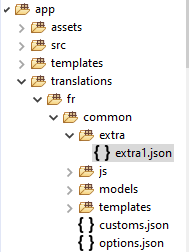
#### Affichage



### Configuration supplémentaires

Permet de récupérer les traductions des supplémentaires d’un projet ou d’un sous projet. La classe concernée est **\muuska\translation\config\DefaultTranslationConfig**. Son constructeur prend en paramètre le nom du dossier et le nom du fichier de traduction (sans extension).

#### Création du fichier de traduction



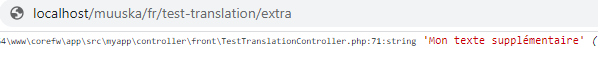
#### Contenu du fichier de traduction



#### Utilisation au niveau du contrôleur



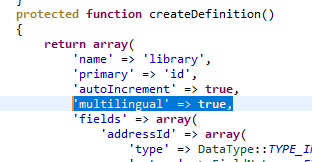
#### Affichage



## Gestion du multilingue des models

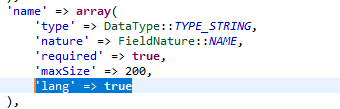
### Activer le multilingue pour un model

Pour activer le multilingue sur un model, allez dans la méthode « **createDefinition** » de sa définition et ajoutez l’option « **multilingual** » avec la valeur « **true** »:



### Activer le multilingue pour un champ

Pour activer le multilingue sur un champ d’un model, allez dans définition de ce champ et ajoutez l’option « **lang** » avec la valeur « **true** »:



### Mise à jour du support d’accès aux données

Puisque la définition des models a changé, vous devez faire une mise à jour du support d’accès aux données.

#### Modifier le processus de mise à jour



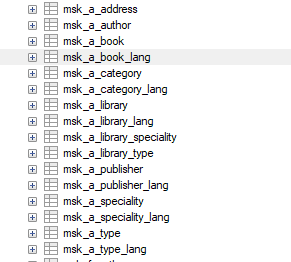
#### Modifier la version de l’application

Il faut modifier la version de l’application afin que la mise à jour puisse se lancer automatiquement à la prochaine exécution de l’application. Pour le faire il faut ajouter l’instruction suivante dans la classe **MyApp**



#### Démarrer la mise à jour

Lancer n’importe quel contrôleur de votre application. Le support d’accès aux données courant (Base de données) fera la mise à jour de votre application. Pour vérifier, ouvrez votre client de base de donnée vous verrez vos tables ajouté avec le préfix « **msk\_a\_** ».



### Méthode de récupération des valeurs traductibles

Pour manipuler les champs traductibles, la classe **\muuska\model\AbstractModel** met à disposition les méthodes suivantes :

#### setPropertyValueByLang

Modifie la valeur d’un champ dans une langue précise. Elle prend en paramètre le nom du champ, la nouvelle valeur et le code de la langue.



#### getPropertyValueByLang

Retourne la valeur d’un champ dans une langue précise. Elle prend en paramètre le nom du champ et le code de la langue.





#### setAllLangsPropertyValues

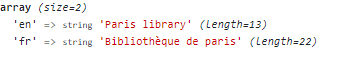
Modifie la valeur d’un champ dans plusieurs langues. Elle prend en paramètre le nom du champ, les nouvelles valeurs (tableau associatif où la clé représente le code de la langue).



#### getAllLangsPropertyValues

Retourne la valeur d’un champ dans toutes les langues. Elle prend en paramètre le nom du champ. Elle retourne un tableau associatif





## Gestion du multilingue avec le DAO

### Sélection

### Enregistrement

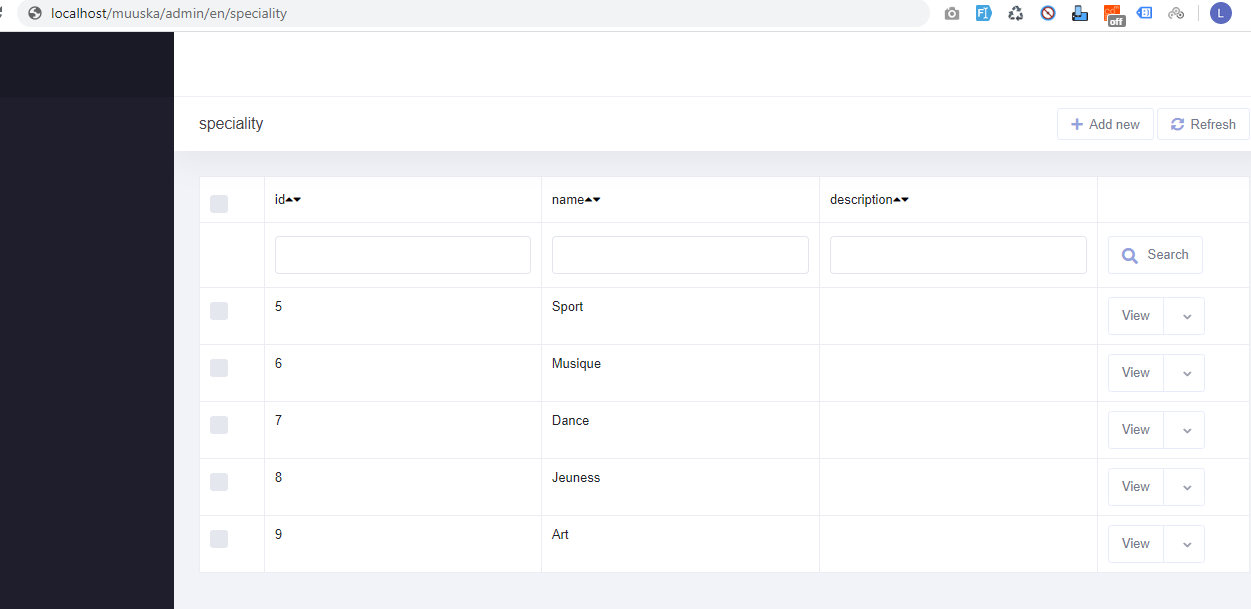
# CRUD

La classe **\muuska\controller\CrudController** implémente les opérations (Create, Read, update, delete).

Code de base

Pour créer un contrôleur CRUD, créer un contrôleur qui hérite de la classe **\muuska\controller\CrudController** et redéfinissez la méthode « **onCreate** » en spécifiant la définition du model.





## List

Pour customiser la liste, vous devez redéfinir la méthode **createListHelper** :

### Gestion des champs externes

Pour définir un champ externe, il faut utiliser la méthode **addExternalFieldDefinition** qui prend en paramètre le nom du champ sur lequel l’association est définit et un tableau associatif permettant de définir les options d’affichages. Ce tableau associatif possède les options suivantes :

#### useToString

Indique si l’on veut appeler la méthode « **\_\_toString** » sur l’objet associé. Valeur par défaut : « **true** ».

#### otherFields

C’est un tableau associatif permettant de spécifier les champs associé au Model que l’on souhaite afficher. La clé représente le nom du champ dans le modèle associé et la valeur est un autre tableau associatif avec les paramètres suivant :

##### label

Le nom qui sera affiché sur l’entête de la colonne.

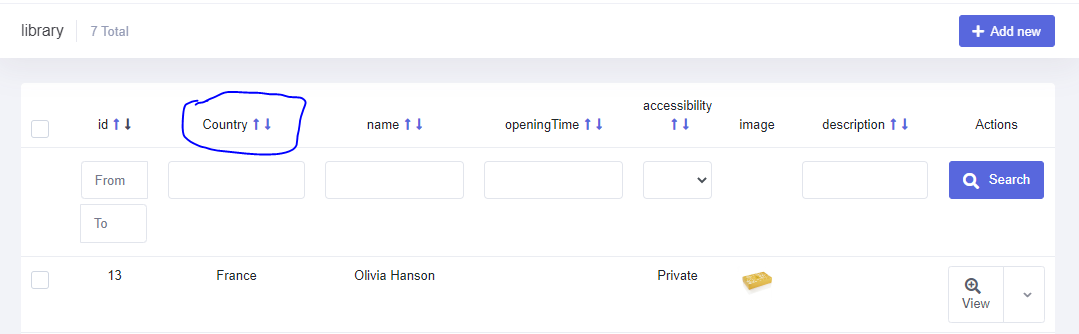
#### hidden

A utiliser si l’on ne veut pas afficher le champ sur lequel l’association est définit ne s’affiche pas dans la liste. Valeur par défaut : « **false** ».

#### externalFieldsDefinition

C’est un tableau associatif permettant de spécifier les options d’affichages des champs externes présent dans l’option « **otherFields** ». La clé de ce tableau représente le nom du champ et la valeur est un tableau associatif permettant de définir les options d’affichages.

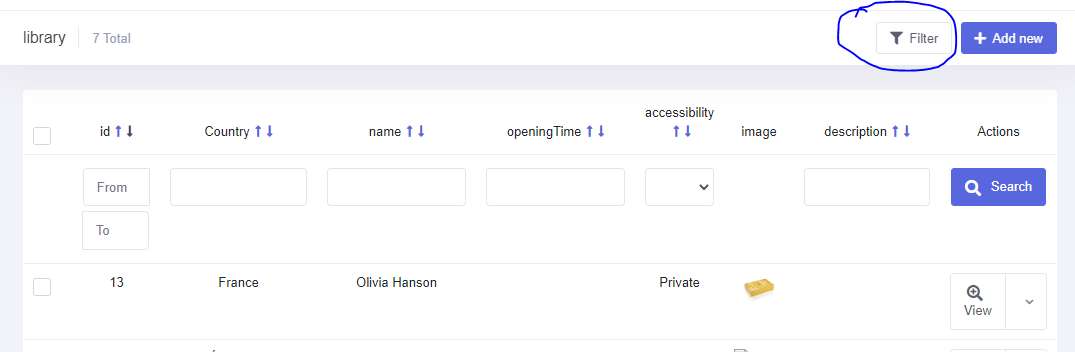




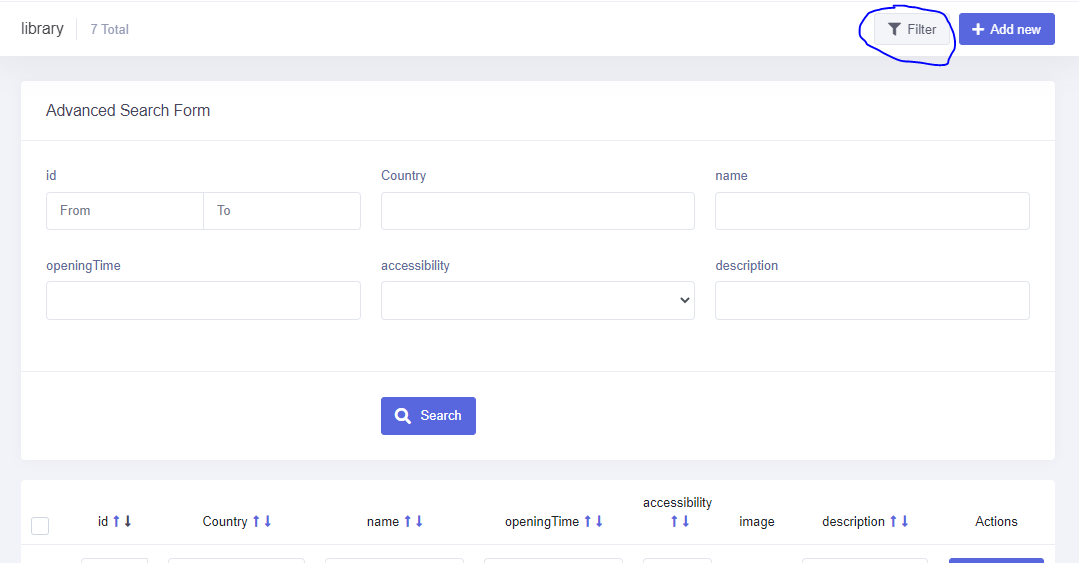
### Activation de la recherche spécifique

Pour le faire, il faut utiliser la méthode **setSpecificSearchEnabled** :





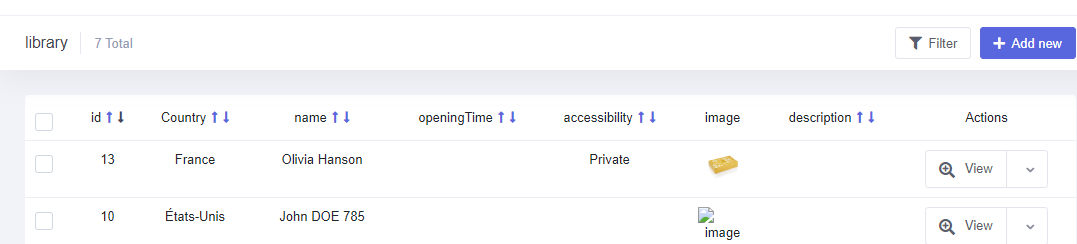
Pour afficher le formulaire de recherche, cliquez sur le bouton « **Filter** ».



### Désactivation de la recherche interne

Pour le faire, il faut utiliser la méthode **setInnerSearchEnabled** :

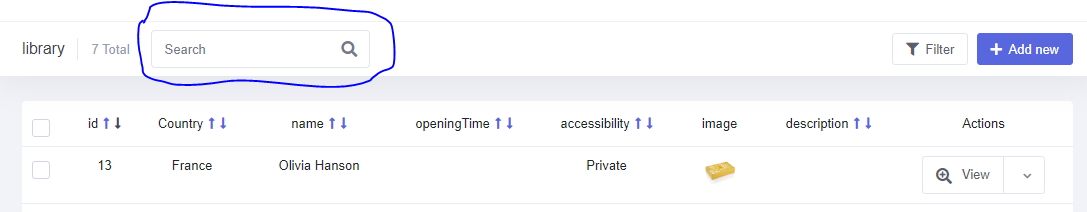




### Activation de la recherche rapide

Pour le faire, il faut utiliser la méthode **setQuickSearchEnabled** :

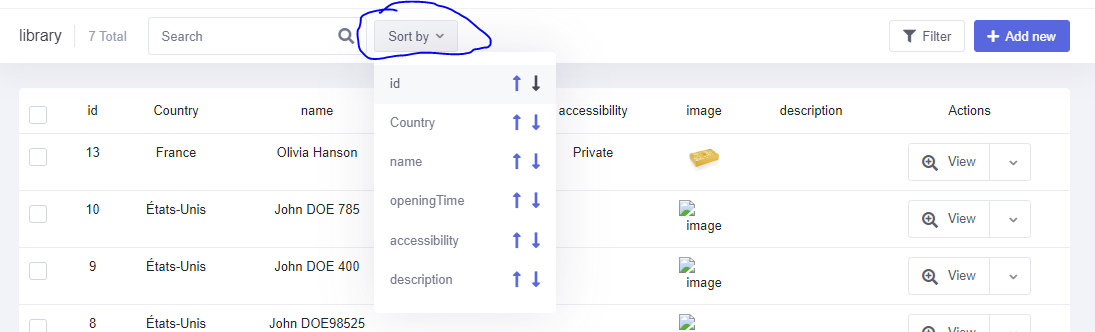




### Activation du trie spécifique

Pour le faire, il faut utiliser la méthode **setSpecificSortEnabled** :



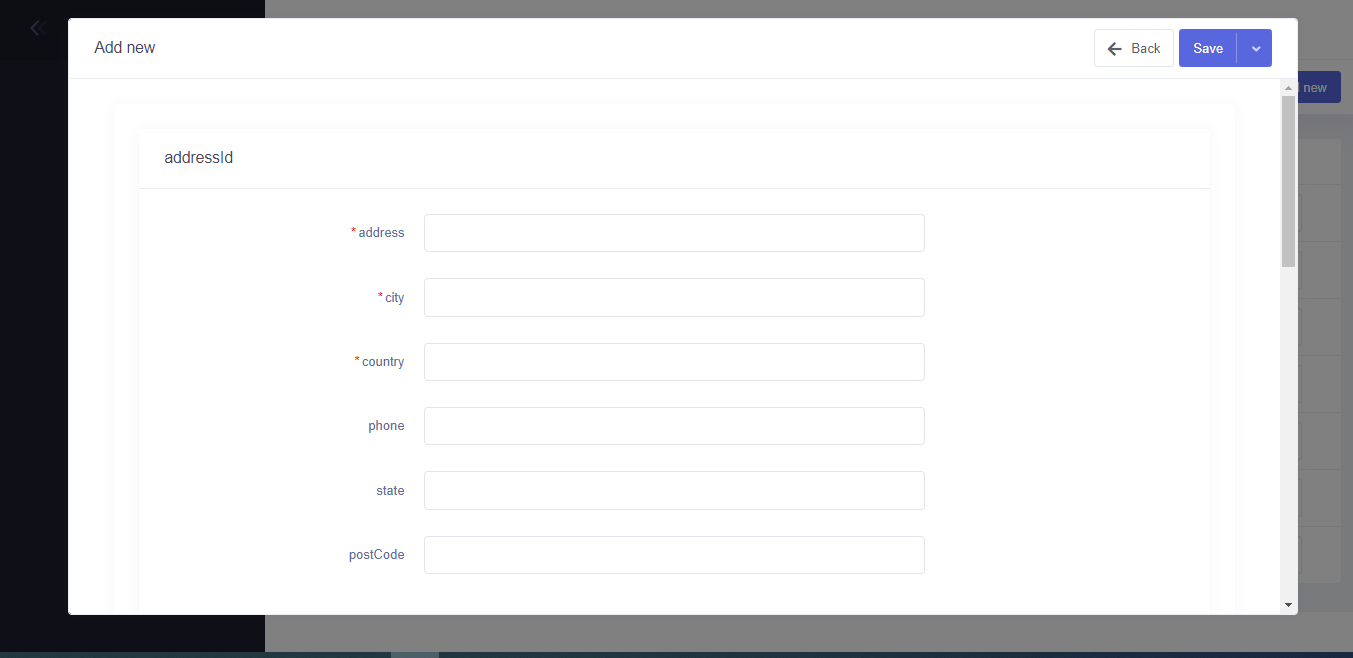


### Activation de l’AJAX

Pour le faire, il faut utiliser la méthode **setAjaxEnabled** :



Si vous cliquez sur n’importe quel bouton de la liste le traitement se fera par AJAX.



## Form

Pour customiser la liste, vous devez redéfinir la méthode **createFormHelper** :

### Gestion des champs externes

Pour définir un champ externe, il faut utiliser la méthode **addExternalFieldDefinition** qui possède les paramètres suivants :

#### field

Représente le nom du champ sur lequel l’association est définit.

#### editionType

Permet de spécifier comment le champ sera édité. La classe **\muuska\constants\ExternalFieldEditionType** contient la liste des valeurs possibles :

##### SELECT

La liste de toutes les valeurs possibles pour l’attribut sont proposées sous forme de liste sélectionnable (Avec un « **Select** » HTML). Par défaut c’est cette option qui est utilisée.

##### All\_FIELDS

Permet d’éditer directement tous les champs du Model associé dans le formulaire courant.

Par exemple : pour enregistrer une bibliothèque nous ne souhaitons pas enregistrer l’adresse en premier avant de venir la sélectionné par la suite ; nous voulons enregistrer toutes les informations au même moment.

##### TREE

La liste de toutes les valeurs possibles pour l’attribut sont proposées sous forme d’arbre. Par défaut c’est cette option qui est utilisée lorsque le model associé à une structure en arbre (Exemple : les catégories).

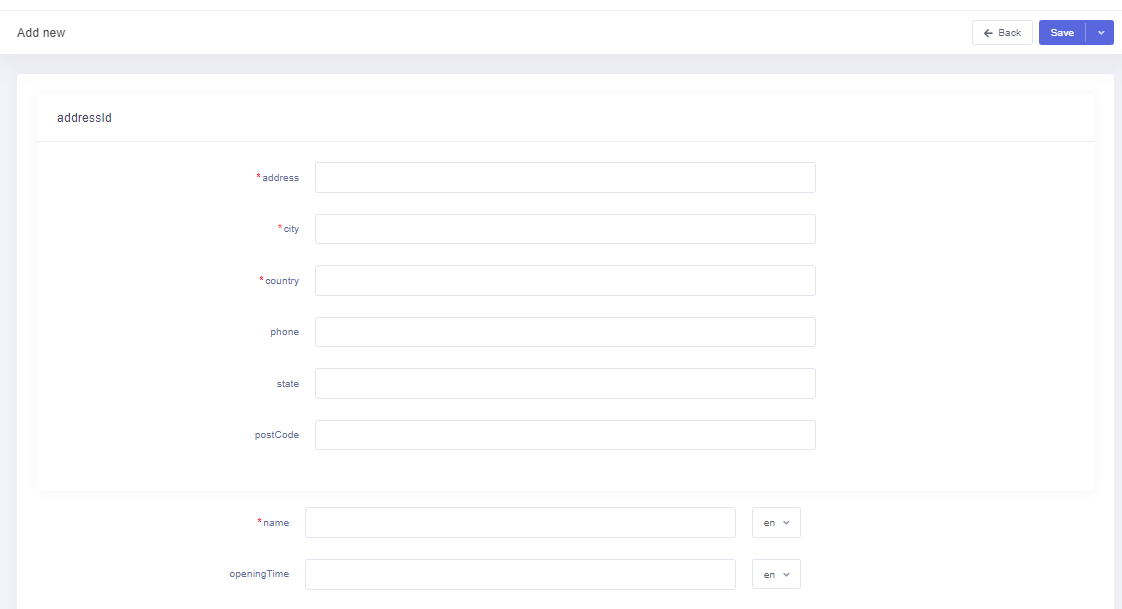
#### excludedFields

Représente la liste des champs à exclure.

#### subExternalFieldsDefinition

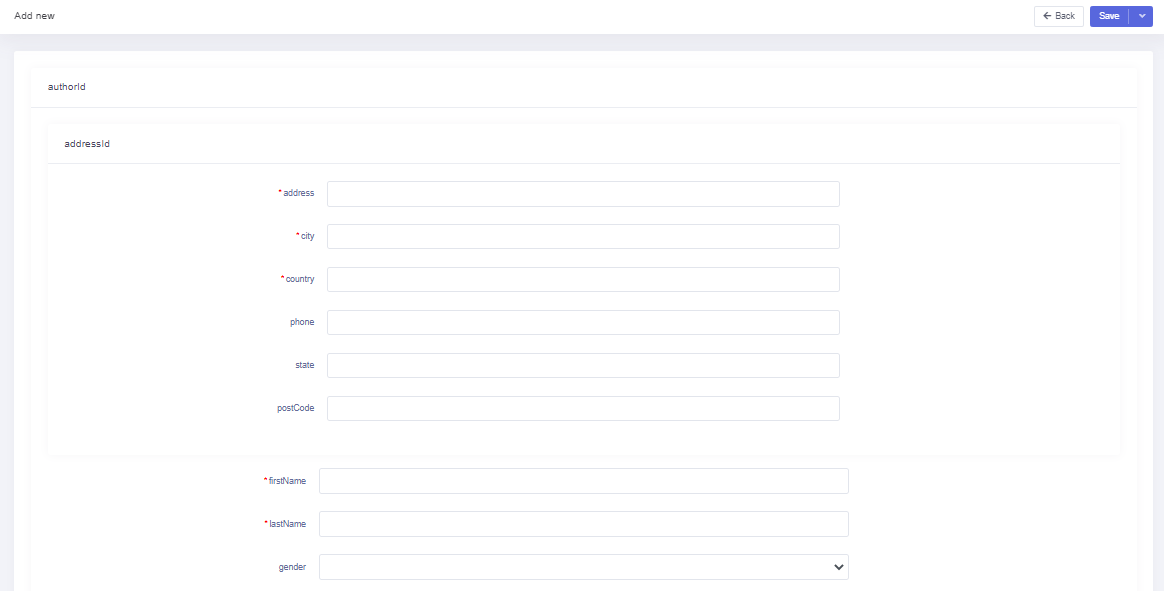
C’est un tableau associatif permettant de spécifier les options d’éditions des champs externes au cas où le paramètre « **editionType** » possède la valeur « **ALL\_FIELDS** ». La clé de ce tableau représente le nom du champ et la valeur est un tableau associatif permettant de définir les options d’édition.





Exemple avec double imbrication :

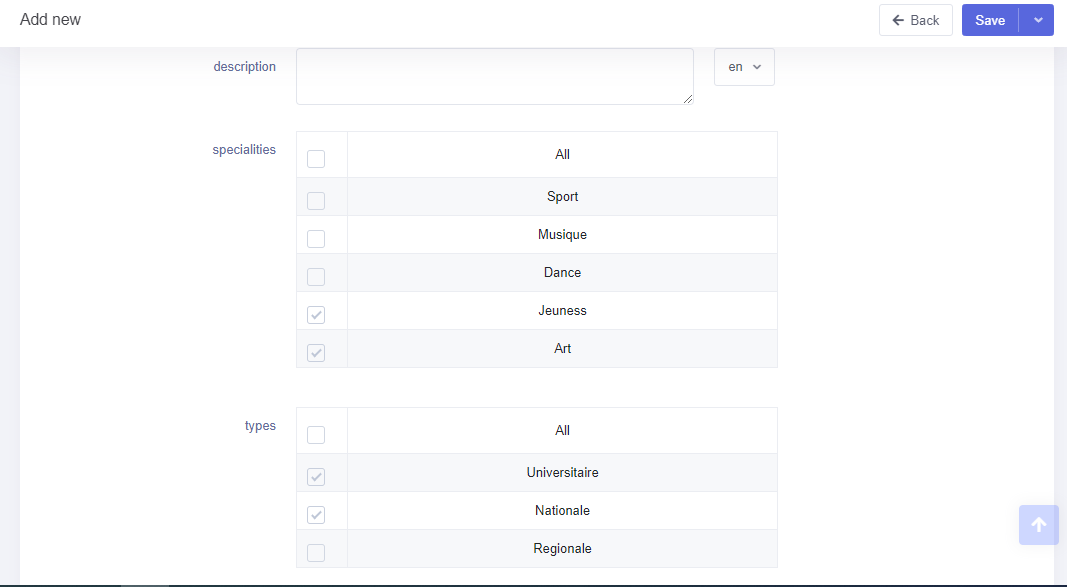




### Gestion des associations multiples

Pour définir une association multiple, il faut utiliser la méthode **addMultipleAssociation** qui prend en paramètre le nom de l’association (Identique au nom indiqué dans la définition du model) et le nom du second champ.





## View

Pour customiser la liste, vous devez redéfinir la méthode **createViewHelper** :

### Gestion des champs externes

Pour définir un champ externe, il faut utiliser la méthode **addExternalFieldDefinition** qui possède les paramètres suivants :

#### field

Représente le nom du champ sur lequel l’association est définit.

#### viewType

Permet de spécifier comment le champ sera affiché. La classe **\muuska\constants\ExternalFieldViewType** contient la liste des valeurs possibles :

##### TO\_STRING

La méthode « **\_\_toString** » sera appelé sur l’objet associé. Par défaut c’est cette option qui est utilisée.

##### All\_FIELDS

Permet d’afficher directement tous les champs du Model associé.

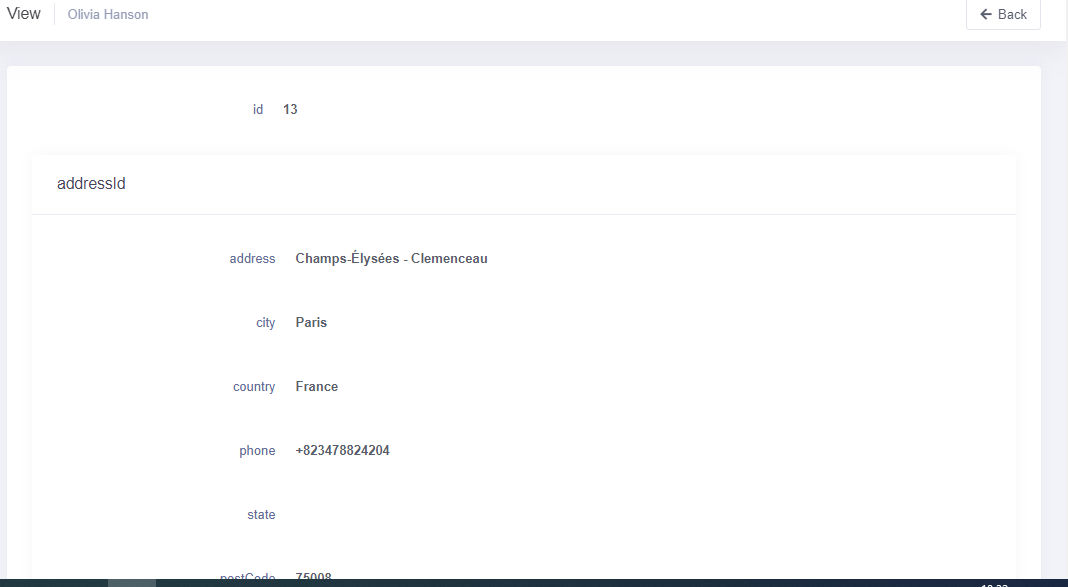
#### excludedFields

Représente la liste des champs à exclure.

#### subExternalFieldsDefinition

C’est un tableau associatif permettant de spécifier les options d’affichages des champs externes au cas où le paramètre « **viewType** » possède la valeur « **ALL\_FIELDS** ». La clé de ce tableau représente le nom du champ et la valeur est un tableau associatif permettant de définir les options d’affichage.

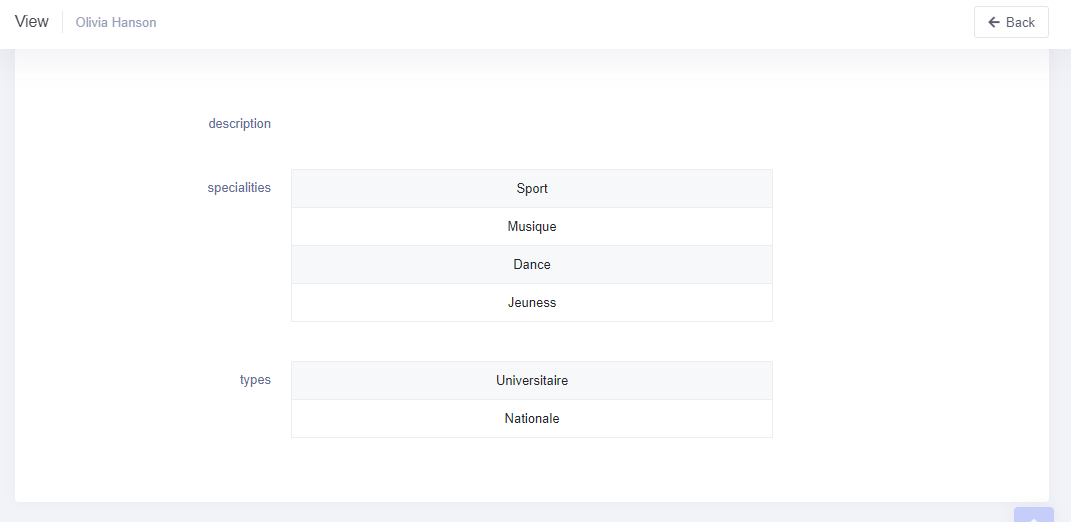




### Gestion des associations multiples

Pour définir une association multiple, il faut utiliser la méthode **addMultipleAssociation** qui prend en paramètre le nom de l’association (Identique au nom indiqué dans la définition du model) et le nom du second champ.





### Utilisation de la navigation interne

La navigation interne vous permet d’afficher d’autre contrôleur dans le contrôleur courant. Par exemple l’affichage des livres liés à une bibliothèque.

#### Activation

Pour activer la navigation interne, vous devez utiliser la méthode **setInnerNavigationEnabled :**



#### Ajout d’un contrôleur

Pour ajouter un contrôleur, vous devez utiliser la méthode **addControllerInnerNavigation** qui prend en paramètre le nom du contrôleur, le label de la navigation et le nom du paramètre qui sera ajouté dans l’URL avec l’identifiant de l’objet courant.

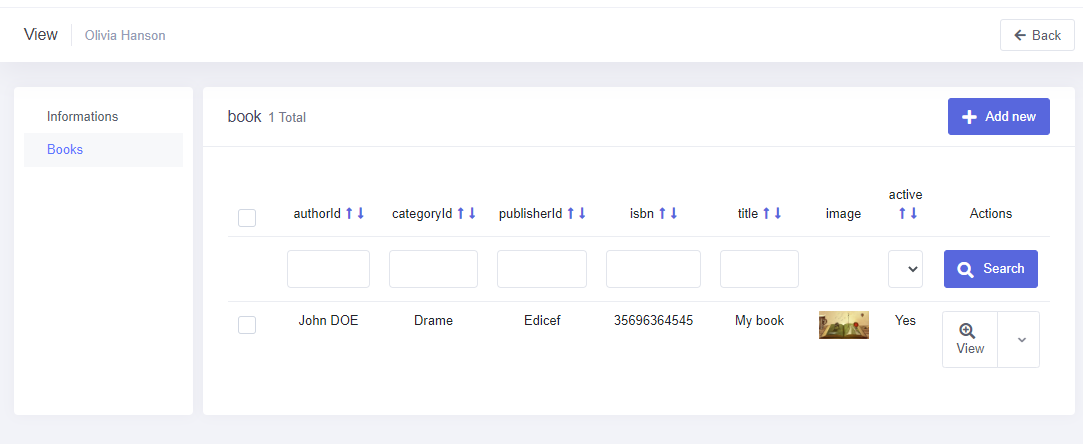


#### Modification du contrôleur externe

Dans le contrôleur externe, vous devez redéfinir la méthode **initParamResolver** afin d’ajouter le nouveau paramètre que vous avez définit dans le contrôleur précèdent :



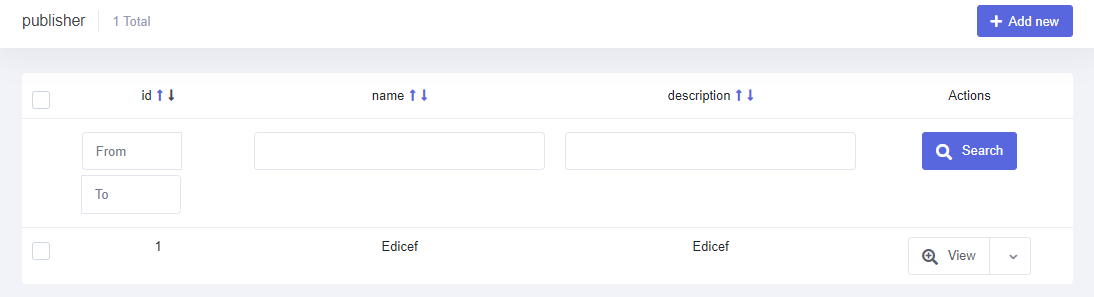
**Résultat**



## Contrôleur virtuel

Pour créer un contrôleur virtuel vous devez utiliser la classe **\muuska\controller\VirtualCrudController**.





# Composant HTML

Ce sont les éléments qui permettent de gérer la vue.

La classe de base des composants est **\muuska\html\HtmlElement**.

## HtmlElement

Cette classe possède plusieurs méthodes pour la manipulation de composant :

### addClass

Ajoute une nouvelle classe.

### addAttribute

Ajoute un nouvel attribut.

### addStyleAttribute

Ajoute un nouvel attribut à d’attribut « **style** ».

### setVisible

Indique si le composant doit visible ou pas. La valeur par défaut est « **true** ».

### setId

L’identifiant du composant.

### setOnlyContentEnabled

Indique si on veut que seul le contenu interne du composant soit affiché. C’est-à-dire que la balise qui enveloppe le composant ne sera pas prise en compte lors de la création du html final (si vous l’utilisez par exemple sur un formulaire, la balise « **form** » ne sera pas dans le HTML finale).

### setName

Le nom du composant.

### setRenderer

Modifier l’objet qui retournera le html finale du composant. L’objet doit être de type **\muuska\renderer\HtmlContentRenderer**.

### addDisabledArea

Spécifie une surface du composant qu’on ne souhaite pas afficher. Par exemple le « **footer** » d’une page, le bouton « **submit** » d’un formulaire.

## HtmlCustomElement

Hérite directement de la classe **HtmlElement**. C’est un composant pouvant contenir plusieurs autres composants. La disposition finale des composants sera prise en compte pas le **renderer** du composant. Pour ajouter des composants vous devez utiliser un objet de type **\muuska\html\areacreator\DefaultAreaCreator**.

## HtmlPage

Composant représentant une page principale. Il hérite directement de la classe **HtmlCustomElement**. Voici la liste des méthodes utiles :

### setTitle

Le titre de la page.

### setLangIso

Le code iso de la langue.

### addBodyClass

Ajoute une classe au body.

### addBodyAttribute

Ajoute un attribut au body.

### addBodyStyleAttribute

Ajoute un nouvel attribut à d’attribut « **style** » du body.

### setBodyVisible

Indique si le body doit visible ou pas. La valeur par défaut est « **true** ».

### setBodyId

L’identifiant du body.

## Icon

**\muuska\html\icon\ClassIcon**.





## Image

**\muuska\html\HtmlImage**.





## Relative Image





## Commande

La classe mère de commande est la **\muuska\html\command\HtmlCommand**.

### Bouton

**\muuska\html\command\Button**.

#### Standard





#### Avec icon





#### Avec icon à droite





#### Avec style





### Lien

**\muuska\html\command\HtmlLink**.

#### Standard





#### Sous forme de bouton





## Input

### Default input

**\muuska\html\input\HtmlInput**.





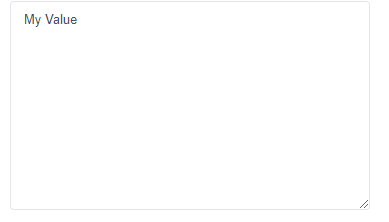
### Input hidden

**\muuska\html\input\InputHidden**.

### Textarea

**\muuska\html\input\Textarea**.





### Select

**\muuska\html\input\Select**.





### Checkbox

**\muuska\html\input\Checkbox**.





### Radio

**\muuska\html\input\Radio**.

#### Standard





#### Inline





### Radio Switch

**\muuska\html\input\RadioSwitch**.

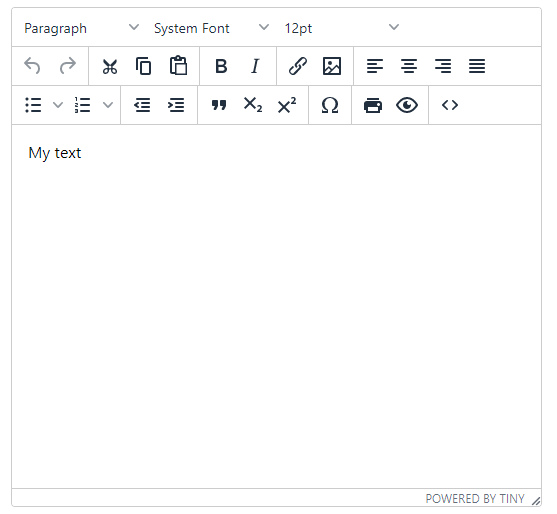




### Rich text editor

**\muuska\html\input\RichTextEditor**.

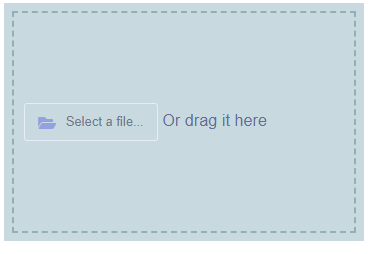




### File upload

**\muuska\html\input\FileUpload**.





#### Add preview

**\muuska\html\input\UploadPreview**.





### Input icon

**\muuska\html\input\InputIcon.**





### Input group

#### Standard

**\muuska\html\input\InputGroup.**





#### Avec Icon





#### Avec bouton





### Custom input group

**\muuska\html\input\CustomInputGroup.**

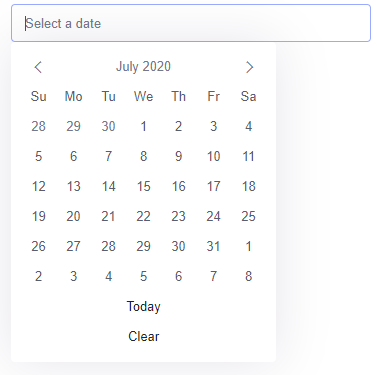
****



### Datepicker

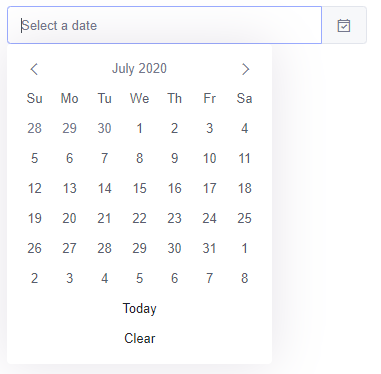
#### Standard





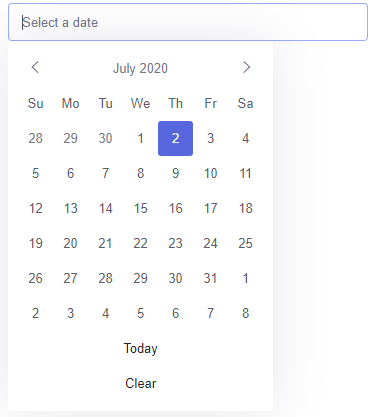
#### Input group





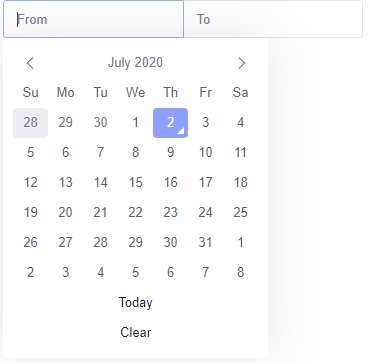
### Datetimepicker





### Date range





Alert

## Formulaire

### Form

### Field

### Grid field

### Translatable Field

### Grid Translatable Field

### Quick search form

## Dropdown

### Default

### Split

### Custom

## Panel

### Default

### List panel

## Navigation

### Nav

### Nav Item

### Full navigation

## Liste

### Definition

### Item

### Field

### Tree

## Children Container

## Banner

## Field Value

## Grid Field Value

# Renderer

## HtmlContentRenderer

C’est un objet permettant de créer le html finale d’un composant.

### Template

#### Concept

#### PHP Template

#### Création d’un nouveau moteur de template

### Callback renderer

## ValueRenderer

C’est un objet permettant de créer le html d’une valeur.

### Model File

### Option Label

### Simple Value renderer

### Image renderer

### Class Icon renderer

# Asset

## Absolute

## Relative

## Group

## Contenu JS

## Variable JS

## Contenu CSS

## Conteneur

## Scope

## Meta

## Link

# JS

Methode utiles

## Ajax

JS

## Controlleur

# Getter

## Concept

## Classes exist ante

## Création

# Checker

## Concept

## Classes existante

## Création

# Validation

# Gestion des options

## Concept

## Création d’un fournisseur d’option

# Gestion du cache

Concept

# Configuration

Une configuration est une instance de l’interface **\muuska\config\Configuration**.

## L’interface \muuska\config\Configuration

## Récupération de la configuration principale

## Création d’une configuration

### JSON

### DAO

## Ajout d’une configuration à l’application

## Obtention d’une Configuration enregistré à l’application

# Mail

## Concept

## Envoi de mail

# URL

## Controller URL creator

## Object URL

### Model URL

### Array URL

## Pagination URL

## List Limiter URL

# Gestion des utilisateurs

## Current User

## Person resolver

# Gestion des Evènements

## Concept

## Ecouter un évènement

# Helpers

# HTTP

## Request

## Response

## Cookie

## Visitor info recorder

## Session

## Router

## Evenement

### Route loading event

### Request parsing event

# Utilitaires

## Tools

## StringTools

## FileTools

## ArrayTools

# Développement de thème

## Concept

## Création d’un thème

## Utilisation d’un thème

# Surcharge

## Concept

## Procedure