

---

# Open Orchestra Documentation

*Version 2.0.0*

**Open Orchestra**

juin 08, 2017



---

## Table des matières

---

<b>1</b>	<b>Guide développeur</b>	<b>1</b>
1.1	Guide développeur . . . . .	1
1.2	Contribution à la documentation . . . . .	26



## Guide développeur

### Installation du CMS Open Orchestra

Par défaut, une application Open Orchestra est composée de deux applications Symfony :

- La Back office ([open-orchestra](#))
- Le Front office ([open-orchestra-front-demo](#))

---

**Note :** Open Orchestra permet aussi de regrouper le Back office et le Front office au sein d'une même application.

---

### Pré-requis

Pour simplifier l'installation et la mise en place d'un environnement compatible, Open Orchestra fournit une configuration [Docker](#).

- [Installer Docker](#)
- [Installer Docker compose](#)

**Prudence :** Une version supérieur à 1.6.0 est requise pour Docker compose et supérieur à 1.10 pour Docker Engine.

### Création des applications

**Prudence :** Cette section considère que vous maîtrisez [Composer](#) et qu'il soit installé.

**Astuce :** Pour simplifier l'installation, il est conseillé de mettre les deux applications Symfony (Back office et Front office) et la configuration de l'environnement Docker au sein du même dossier.

---

Création des applications Front et Back office :

```
1 $ cd my_project_name/
2
3 # création des applications
4 $ composer create-project open-orchestra/open-orchestra open-orchestra --ignore-
  ↳platform-reqs --no-scripts
5 $ composer create-project open-orchestra/open-orchestra-front-demo open-orchestra-
  ↳front-demo --ignore-platform-reqs --no-scripts
6
7 # Dossier qui contiendra les medias de la médiathèque
8 $ mkdir uploaded-files
```

Création de l'environnement docker

```
1 $ cd my_project_name/
2
3 # clonage de la configuration pour l'environnement docker
4 $ git clone git@github.com:open-orchestra/open-orchestra-provision-docker.git
5
6 $ cd open-orchestra-provision-docker/
7 # création des conteneurs
8 $ docker-compose up -d
```

Installation des applications

Installation des vendors, des dépendances Javascript et de la configuration.

Back office :

```
1 $ docker exec -it -u www-data oo_apache_php /bin/bash
2 $ cd /var/www/openorchestra/ && composer install #install vendors
3 $ ./bin/node ./node_modules/.bin/grunt # compilation des less et js
4 $ exit
```

**Note :** Lorsque Composer demandera de remplir les différents paramètres, utilisez les valeurs ci-dessous ou la valeur par défaut si le paramètre n'est pas spécifié

```
1 open_orchestra_cms.mongodb.host: oo_mongo
2 fos_http_cache.proxy_client.varnish.servers: [oo_varnish:6081]
3 media_storage_directory: /var/www/uploaded-files
4 host_elastica: oo_elasticsearch
```

Front office :

```
1 $ docker exec -it -u www-data oo_apache_php /bin/bash
2 $ cd /var/www/front-openorchestra/ && composer install
3 $ app/console assets:install
4 $ exit
```

---

**Note :** Lorsque Composer demandera de remplir les différents paramètres, utilisez les valeurs ci-dessous ou la valeur par défaut si le paramètre n'est pas spécifié

```
1 open_orchestra_cms.mongodb.server: 'mongodb://oo_mongo:27017'  
2 fos_http_cache.proxy_client.varnish.servers: [oo_varnish:6081]  
3 host_elastica: oo_elasticsearch
```

## Configuration des hosts

Dans le fichier de configuration des hosts de votre ordinateur (/etc/hosts pour linux) utilisez les dns suivants :

```
1 [IP]    admin.openorchestra.2-0.dev  
2 [IP]    demo.openorchestra.2-0.dev  
3 [IP]    media.openorchestra.2-0.dev
```

---

### Note :

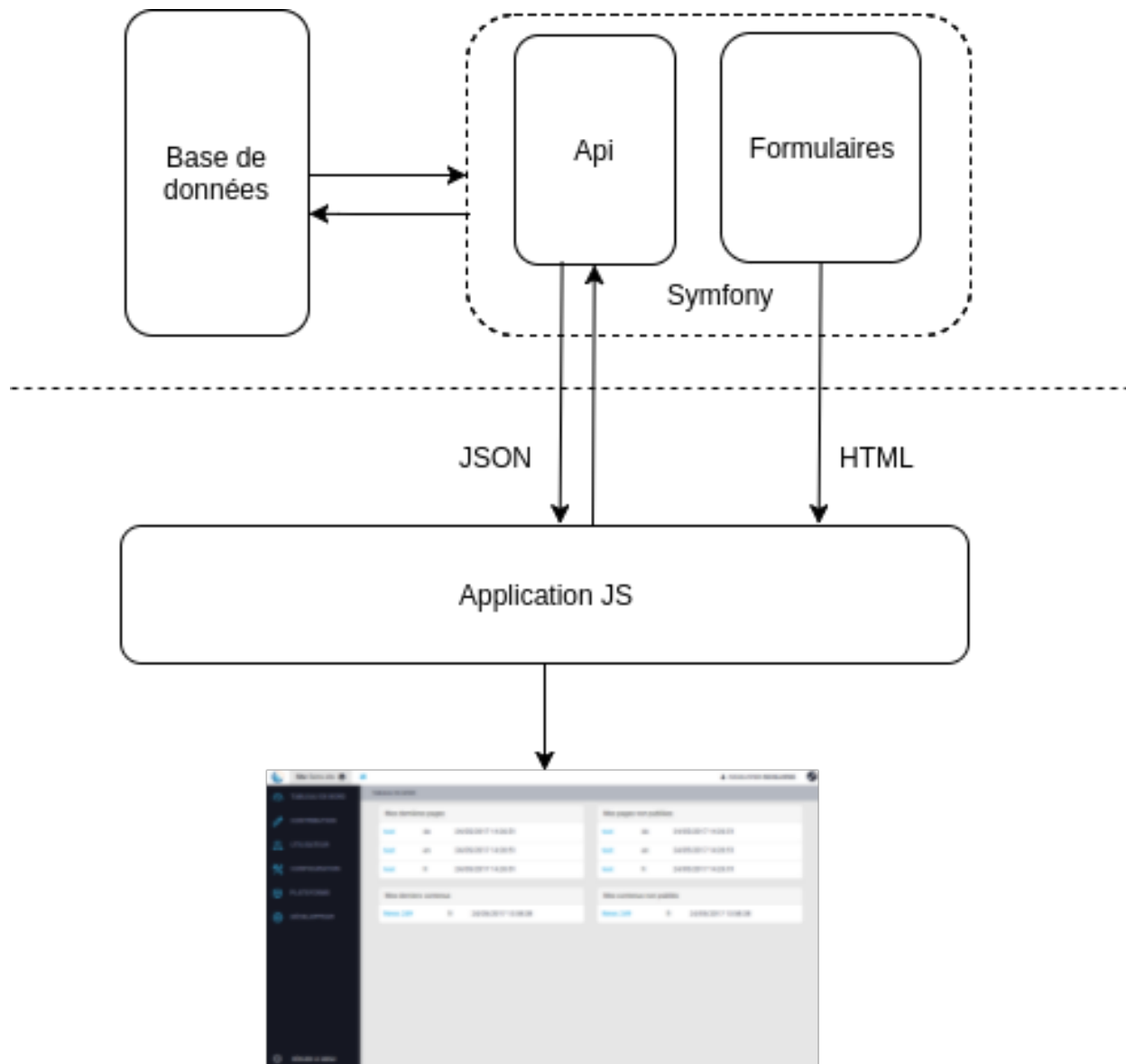
[IP] doit être remplacé par 127.0.0.1 pour Linux.

[IP] doit être remplacé par la valeur fournie par la commande `docker-machine ip default`

---

## Architecture des applications

## Back office



## Base de données

Open Orchestra n'est pas lié à un type de base de données spécifique. Par défaut, il y a une implémentation avec [MongoDB](#) qui est fourni. Toutefois elle peut être remplacée afin d'utiliser une autre base de données (Mysql, ...).

## Api

Open Orchestra fournit une [Api REST](#) réalisée avec [Symfony](#) afin d'accéder aux différentes entités du CMS (pages, contenus, sites, etc)



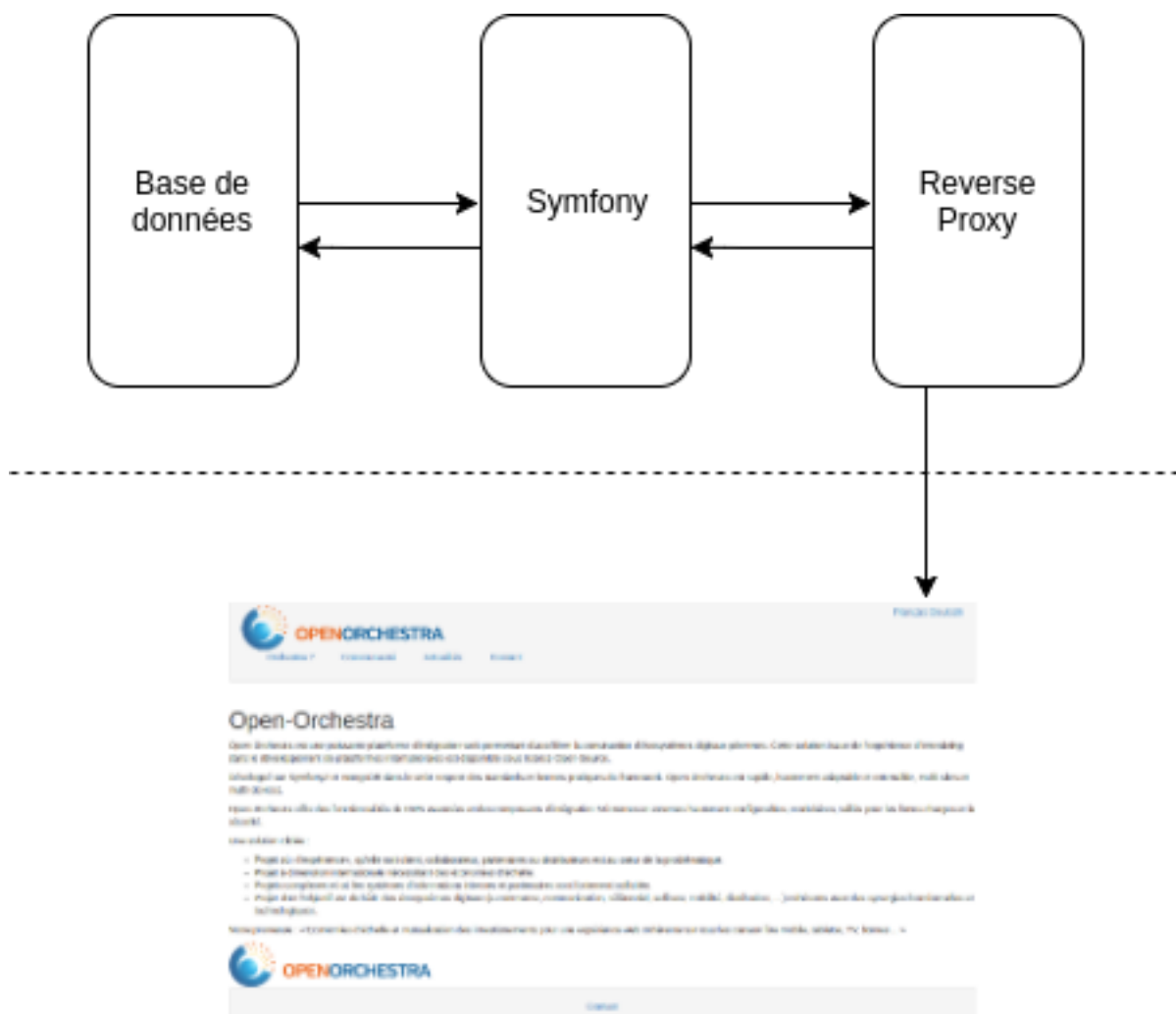
## Form

Sur Open Orchestra les formulaires ne sont pas créés par l'application Javascript mais par Symfony, cela afin de bénéficier des différents avantages du [composant form](#) de Symfony. L'application Javascript récupère les formulaires générés par Symfony en Ajax.

## Application Javascript

Le Back office d'Open Orchestra est réalisé intégralement en JavaScript avec le framework [Backbone.js](#) L'application Javascript récupère les différentes données en Ajax fournies par l'application Symfony.

## Front office



**L'application Front office d'un site réalisée avec Open Orchestra se rapproche plus de l'architecture standard d'une application Symfony.**

La seule particularité est la présence d'un reverse proxy qui permet d'améliorer les performances avec notamment l'utilisation du cache [esi](#) .

**Note :** La *configuration docker* proposée par Open Orchestra utilise [Varnish](#) comme reverse proxy. Il est accessible depuis le port 6081

---

**Note :** Sur Open Orchestra la présence d'un reverse proxy n'est pas obligatoire. Si aucun reverse proxy n'est configuré sur l'application Front office, cette dernière continuera de fonctionner.

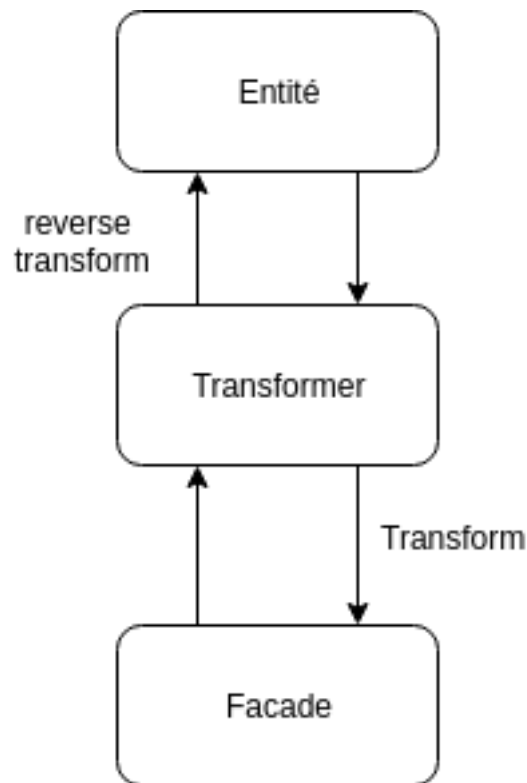
---

## API

Open Orchestra fournit une API REST pour accéder aux différentes entités du CMS (pages, contenus, sites, etc). Afin de découpler le stockage des données et leur exposition dans l'API, Open Orchestra implémente le [design pattern facade](#).

Le design pattern *facade* se découpe en deux parties : Tout d'abord les *facades* qui sont des objets avec différents attributs qui seront ceux exposés par l'API ; puis les *transformer*, qui à partir des données provenant de la base de données remplissent une *facade*.

Les *transformer* permettent aussi de retourner une entité, qui pourra être stockée, depuis les données d'une *facade*.



### Exposer une entité

Supposons que vous construisiez une application *todo list* qui devra afficher différentes tâches. Vous disposez d'une classe *Task* qui représente et stocke les données d'une tâche.

```
1 // src/AppBundle/Entity/Task.php
2 namespace AppBundle\Entity;
```

```

3
4 class Task
5 {
6     protected $title;
7     protected $dueDate;
8
9     // ... getters
10    // ... setters
11 }

```

Dans un premier temps, il faut créer la classe qui représente la *facade*. Celle-ci doit implémenter `OpenOrchestra\BaseApi\Facade\FacadeInterface` afin d'être reconnue par Open Orchestra comme étant une *facade*.

```

1 // src/AppBundle/Facade/TaskFacade.php
2 namespace AppBundle\Facade;
3
4 use OpenOrchestra\BaseApi\Facade\FacadeInterface;
5
6 class TaskFacade implements FacadeInterface
7 {
8     public $title;
9     public $dueDate;
10 }

```

Afin de remplir cette *facade* à partir des données de l'entité `AppBundle\Entity\Task`, il faut créer le *transformer* qui doit étendre la classe abstraite `OpenOrchestra\BaseApi\Transformer\AbstractTransformer`.

```

1 // src/AppBundle/Transformer/TaskTransformer.php
2 namespace AppBundle\Transformer;
3
4 use OpenOrchestra\BaseApi\Transformer\AbstractTransformer;
5
6 class TaskTransformer extends AbstractTransformer
7 {
8     // Rempli les différents attributs de la facade
9     // avec ceux stockés dans la tâche ($task)
10    public function transform($task)
11    {
12        $facade = new TaskFacade();
13        $facade->title = $task->getTitle();
14        $facade->dueDate = $task->getDueDate();
15
16        return $facade;
17    }
18
19    // Créé une tâche à partir des données de la facade
20    public function reverseTransform($taskFacade)
21    {
22        $task = new Task();
23        $task->setTitle($taskFacade->title);
24        $task->setDueDate($taskFacade->dueDate);
25
26        return $task;
27    }
28
29    // Nom du transformer afin de l'identifier lors de son utilisation
30    public function getName()

```

```
31 {
32     return 'task_transformer';
33 }
34 }
```

Afin de limiter les dépendances et faciliter l'utilisation des *transformer* dans le reste de l'application, il faut enregistrer le *transformer* en tant que service *taggé*.

```
1 app_bundle.transformer.task:
2     class: AppBundle\Transformer\TaskTransformer
3     tags:
4         - { name: open_orchestra_api.transformer.strategy }
```

Le *transformer* `TaskTransformer` peut être maintenant appelé en utilisant le `TransformerManager`.

Le `TransformerManager` est un service qui connaît tous les *transformer* de l'application. Cela permet de simplifier les appels à ces derniers.

```
1 // src/AppBundle/Controller/Api/TaskController.php
2 namespace AppBundle\Controller;
3
4 class TaskController extends Controller
5 {
6     public function showAction()
7     {
8         // Création d'un objet tâche
9         // celui-ci pourrait aussi provenir d'une base de données
10        $task = new Task();
11        $task->setTitle('test');
12
13        // Transformation de l'object Task en facade
14        // en utilisant le transformer manager
15        //
16        // task_transformer est le nom du transformer défini par la
17        // méthode getName de AppBundle\Transformer\TaskTransformer
18        $facade = $this
19            ->get('open_orchestra_api.transformer_manager')
20            ->get('task_transformer')
21            ->transformer($task);
22    }
23 }
```

## Sérialisation

Afin de retourner une réponse JSON, la *facade* doit être sérialisée. Pour cela Open Orchestra utilise le bundle `JMSSerializerBundle`.

Afin de sérialiser la *facade*, il faut indiquer à `JMSSerializerBundle` le type des différentes propriétés.

```
1 // src/AppBundle/Facade/TaskFacade.php
2 namespace AppBundle\Facade;
3
4 use OpenOrchestra\BaseApi\Facade\FacadeInterface;
5 use JMS\Serializer\Annotation\Type;
6
7 class TaskFacade implements FacadeInterface
8 {
```

```

9      /**
10     * @Type("string")
11     */
12     public $title;
13
14     /**
15     * @Type("DateTime")
16     */
17     public $dueDate;
18 }

```

**Prudence :** Les annotations sont mises en cache. Il faut donc vider ce dernier après modification des annotations d'une *facade*.

**Astuce :** L'utilisation des annotations n'est pas obligatoire. JMSSerializerBundle supporte aussi la configuration en **YAML** ou **XML**.

Une fois la configuration effectuée, nous pouvons utiliser le service `jms_serializer` afin de sérialiser la *facade* en JSON.

```

1 // src/AppBundle/Controller/Api/TaskController.php
2 namespace AppBundle\Controller;
3
4 class TaskController extends Controller
5 {
6     public function showAction()
7     {
8         $task = new Task();
9         $task->setTitle('test');
10
11         $facade = $this
12             ->get('open_orchestra_api.transformer_manager')
13             ->get('task_transformer')
14             ->transformer($task);
15
16         // appel au service JMSSerializerBundle
17         $serializer = $container->get('jms_serializer');
18
19         // sérialisation de la *facade* en JSON
20         $content = $serializer->serialize($facade, 'json');
21
22         // Création d'une réponse Symfony
23         return new Response(
24             serializer
25             200,
26             array('content-type' => 'application/json')
27         )
28     }
29 }

```

**Astuce :** Open Orchestra propose de créer automatiquement une *Response* JSON à partir d'une *facade* retournée par une action de *Controller* grâce à l'annotation

OpenOrchestra\BaseApiBundle\Controller\Annotation\serialize.

```
1 // src/AppBundle/Controller/Api/TaskController.php
2 namespace AppBundle\Controller;
3
4 use OpenOrchestra\BaseApiBundle\Controller\Annotation as Api;
5
6 class TaskController extends Controller
7 {
8     /**
9      * @Api\Serialize()
10     */
11     public function showAction()
12     {
13         $task = new Task();
14         $task->setTitle('test');
15
16         $facade = $this
17             ->get('open_orchestra_api.transformer_manager')
18             ->get('task_transformer')
19             ->transformer($task);
20
21         return $facade;
22     }
23 }
```

Si l'annotation est placée directement sur la classe alors tous les retours des actions du *Controller* seront sérialisés.

---

### Contexte de sérialisation

Lorsque l'API d'une application Open Orchestra devient conséquente, il peut être intéressant de sérialiser/transformer suivant le contexte de l'action uniquement certains éléments d'une *facade*.

Pour cela, Open Orchestra fournit l'annotation `OpenOrchestra\BaseApiBundle\Controller\Annotation\Groups` qui permet de spécifier un groupe de contexte pour l'action courante.

```
1 // src/AppBundle/Controller/Api/TaskController.php
2 namespace AppBundle\Controller;
3
4 use OpenOrchestra\BaseApiBundle\Controller\Annotation as Api;
5
6 class TaskController extends Controller
7 {
8     /**
9      * @Api\Serialize()
10     *
11     * Indique que l'action "show" a pour contexte le groupe SHOW
12     * @Api\Groups({"show_dueDate"})
13     */
14     public function showAction()
15     {
16         // ...
17     }
18 }
```

---

**Astuce :** Pour simplifier et centraliser les différents contextes de l'API, il est conseillé d'utiliser des constantes.

```

1  /**
2   * @Api\Serialize()
3   *
4   * @Api\Groups({"AppBundle\Context\ApiContext::SHOW_DUE_DATE"})
5   */
6  public function showAction()
7  {
8      // ...
9  }

```

Le contexte d'une action peut être utilisé dans les *transformer* grâce à la méthode `hasGroup($group)`

```

1  // src/AppBundle/Transformer/TaskTransformer.php
2  namespace AppBundle\Transformer;
3
4  use OpenOrchestra\BaseApi\Transformer\AbstractTransformer;
5
6  class TaskTransformer extends AbstractTransformer
7  {
8      // Rempli les différents attributs de la facade
9      // avec ceux stockés dans la tâche ($task)
10     public function transform($task)
11     {
12         $facade = new TaskFacade();
13         $facade->title = $task->getTitle();
14
15         // l'attribut dueDate sera ajouté à la facade uniquement
16         // si l'action qui demande la transformation appartient au groupe
17         // show_dueDate sinon il ne sera pas ajouté à la facade et donc ne
18         // sera pas sérialisé
19         if ($this->hashGroup("show_dueDate")) {
20             $facade->dueDate = $task->getDueDate();
21         }
22
23         return $facade;
24     }
25
26     // ...
27 }

```

## Désérialisation

L'API permet aussi d'effectuer des modifications sur les entités à partir de données JSON fournies par l'application Javascript.

Pour cela, il faut utiliser la méthode `deserialize` du service `jms_serializer`, afin de remplir une *facade* à partir des données d'une requête.

Ensuite, afin d'obtenir une entité à partir de la *facade* nous pouvons utiliser la méthode `reverseTransform` des *transformers*.

```

1  // src/AppBundle/Controller/Api/TaskController.php
2  namespace AppBundle\Transformer;
3
4  class TaskController extends Controller
5  {

```

```

6   public function editAction(Request $request)
7   {
8       // Désérialisation du contenu de la requête
9       // dans la facade TaskFacade
10      $facade = $this
11          ->get('jms_serializer')
12          ->deserialize(
13              $request->getContent(),
14              'AppBundle\Facade\TaskFacade',
15              $request->get('_format', 'json')
16          );
17
18      // Utilisation du transformer manager pour récupérer
19      // le transformer task
20      // @see AppBundle\Transformer\TaskTransformer
21      $task = $this
22          ->get('open_orchestra_api.transformer_manager')
23          ->get('task_transformer')
24          ->reverseTransform($facade);
25
26      // ...
27      // Validation de l'entité task
28      // Sauvegarde de l'entité
29  }
30 }

```

## Client Javascript

Le Back office d'Open Orchestra est réalisé intégralement en JavaScript, plus précisément en [ECMAScript 6](#), avec le framework [Backbone.js](#)

### Application

Open Orchestra est composé d'une application principale et de sous-application qui apportent des fonctionnalités supplémentaires comme par exemple la gestion de la médiathèque.

### Application principale

L'application principale est le point d'entrée du Back office, elle est représentée par l'objet `Application` se trouvant dans `BackofficeBundle/Resources/public/ecmascript/OpenOrchestra/Application/Application.js`.

Afin de démarrer l'application, il suffit de faire appel à la méthode `run()` de cet objet qui va initialiser les différents éléments nécessaires.

```

1   // BackofficeBundle/Resources/public/ecmascript/OpenOrchestra/Application/Application.
2   ↪ js
3   class Application
4   {
5       // ...
6
7       /**
8        * Run Application

```



```

9      */
10     run() {
11         // Initialisation des paramètres de configuration
12         // de l'application
13         this._initConfiguration();
14
15         // ...
16
17         // Initialisation des différents routeurs Backbone
18         this._initRouter();
19
20         // Événement lancé avant le démarrage de l'application
21         Backbone.Events.trigger('application:before:start');
22
23         // Initialisation des layouts de l'application (menu, header, etc)
24         this._initLayoutView();
25
26         // Démarrage de l'application
27         Backbone.history.start();
28
29         // Événement lancé après le démarrage de l'application
30         Backbone.Events.trigger('application:after:start');
31     }
32
33     // ...
34 }

```

## Sous-application

Open Orchestre est découpé en différents bundles Symfony qui apportent chacun leur lot de fonctionnalités. Ainsi, chaque bundle peut posséder une sous-application Javascript pour ajouter des fonctionnalités à l'application principale.

Pour instancier une sous-application, il suffit d'utiliser l'événement `application:before:start` qui est lancé juste avant le démarrage de l'application principale.

```

1 // src/AppBundle/Resources/public/main.js
2
3 // Objet représentant la sous-application
4 import AppSubApplication from './Application/AppSubApplication'
5
6 $((() => {
7     Backbone.Events.on('application:before:start', () => {
8         // Démarrage de la sous-application (router, ajout de configuration, etc)
9         AppSubApplication.run();
10     });
11 });

```

## Structure

L'application principale et les sous-applications d'Open Orchestra présentes dans le dossier `Ressources\public` des bundles utilisent la même structure.

```

1 - Ressources/public
2   |

```

```
3  - config # Fichiers json de configuration de l'application (menu, etc)
4  |
5  - ecmaScript # Classe Javascript es6 de l'application
6  | |
7  |   - OpenOrchestra # "Namespace"
8  |   |
9  |     - Application
10 |     |   - Collection # Collections backbone
11 |     |   - Model      # Models backbone
12 |     |   - Router     # Routeurs backbone
13 |     |   - Views      # Vue Backbones
14 |     |
15 |     - Service # Classes génériques,
16 |     |         # utilisées par différents éléments de l'application
17 |     |
18 |     - main.js # Démarrage de l'application ou sous-application
19 |
20 - template # template underscore
```

### Backbone.js

**Prudence :** Cette section considère que vous maîtrisez [Backbone.js](#)

### ECMAScript 6

En ECMAScript 6, il n'est pas possible de définir des propriétés de classe en dehors des méthodes. Or les différents composants de Backbone.js (View, Model) doivent définir divers propriétés de classe (tagName, className).

Ainsi, les différents composants de Backbone.js ont été étendus afin d'ajouter une méthode `preinitialize` pour définir ces propriétés.

```
1  class CustomView extends Backbone.View {
2
3      preinitialize() {
4          this.tagName = "li";
5          this.className = "custom-class";
6      }
7
8      initialize() {
9          //...
10     }
11
12     render() {
13         //...
14     }
15 }
```

---

**Note :** L'ajout de la méthode `preinitialize` est une fonctionnalité qui sera ajoutée dans la version 1.4 de Backbone.js

---

## Composants Backbone.js

Les différents composants de Backbone.js ont été étendus afin d'ajouter des comportements spécifiques à Open Orchestra (rendu de template dans les vues, gestion des erreurs lors des appels API, ...).

Ainsi lorsque vous désirez créer un *model*, *router*, *collection*, *view*, il est préférable d'étendre les composants Open Orchestra.

```

1 // src/AppBundle/Resources/public/MyApp/Application/View/AppView.js
2 import OrchestraView from '../../../../../../OpenOrchestra/Application/View/OrchestraView'
3 class AppView extends OrchestraView {}
4
5 // src/AppBundle/Resources/public/MyApp/Application/Router/AppRouter.js
6 import OrchestraView from '../../../../../../OpenOrchestra/Application/Router/
  ↳OrchestraRouter'
7 class AppRouter extends OrchestraRouter {}
8
9 // src/AppBundle/Resources/public/MyApp/Application/Collection/AppCollection.js
10 import OrchestraView from '../../../../../../OpenOrchestra/Application/Collection/
  ↳OrchestraCollection'
11 class AppCollection extends OrchestraCollection {}
12
13 // src/AppBundle/Resources/public/MyApp/Application/Model/AppModel.js
14 import OrchestraView from '../../../../../../OpenOrchestra/Application/Model/OrchestraModel'
15 class AppModel extends OrchestraModel {}

```

## Routes Symfony

Pour faciliter l'utilisation de l'API, au sein de l'application JavaScript, vous pouvez accéder aux routes Symfony grâce au bundle [FOSJsRoutingBundle](#)

---

**Note :** La génération des routes en JSON est géré par Open Orchestra grâce à une tâche Grunt.

---

**Prudence :** Afin que le bundle prenne en compte les routes des contrôleurs celles-ci doivent posséder l'option `expose = true`, plus d'informations dans la [documentation](#).

## Traductions Symfony

Pour faciliter, la gestion des traductions dans l'application JavaScript, vous pouvez utiliser le [composant de traduction de Symfony](#).

Afin d'accéder aux traductions en Javascript Open Orchestra utilise le bundle [JsTranslationBundle](#)

---

**Note :** Le domaine de traduction qui est exposé par défaut sur Open Orchestra est `interface`.

---



---

**Note :** La génération des traductions en Javascript est géré par Open Orchestra grâce à une tâche Grunt.

---

### Template

Au sein des vues, Open Orchestra utilise les [templates Underscore](#) .

Pour simplifier le chargement et l'utilisation des différents templates, Open Orchestra met en place le service `TemplateManager` qui permet de récupérer un template underscore à partir de son nom.

---

**Note :** Les différents templates sont automatiquement compilés par une tâche Grunt dans la variable `Orchestra.Template`.

Pour que vos template soient compilés par Grunt, il faut que celui-ci se trouve dans le dossier template des ressources publiques de votre bundle (exemple : `src/AppBundle/Resources/public/template`)

---

Le `TemplateManager` est directement accessible dans les vues Backbone.Js, si celles-ci étendent bien `OrchestraView` grâce à la méthode `_renderTemplate(templateName, parameters)`.

```
1 // src/AppBundle/Resources/public/MyApp/Application/View/AppView.js
2 import OrchestraView from '../../../../../../OpenOrchestra/Application/View/OrchestraView'
3
4 class AppView extends OrchestraView {
5
6     // ...
7
8     render() {
9         let template = this._renderTemplate('helloView', {
10             name: 'Foo'
11         });
12         this.$el.append(template);
13     }
14 }
```

```
1 <!-- src/AppBundle/Resources/public/template/helloView._tpl.html -->
2
3 <p> Hello <%- name %> </p>
```

---

**Note :** Lors du rendu d'un template la méthode `_renderTemplate` injecte automatiquement le paramètre `renderTemplate` qui permet d'injecter un template à l'intérieur d'un autre template.

```
1 <!-- src/AppBundle/Resources/public/template/helloView._tpl.html -->
2
3 <p> Hello <%- renderTemplate('otherTemplate') %> </p>
```

---

### Grunt

Afin de gérer les différentes ressources (JavaScript, CSS) Open Orchestra utilise [Grunt](#).

Il y a deux tâches Grunt importantes :

La tâche `css` qui s'occupe de compiler et concaténer les fichiers `less` des différents bundles se trouvant dans `/Resources/public/less`.

Puis la tâche `javascript` qui s'occupe de gérer tous les éléments nécessaires pour l'application JavaScript (compilation des fichiers js, exposition des traductions et des routes, compilation des templates underscores)

Ces deux tâches ne s'appliquent pas sur tous les bundles/vendors Symfony, il faut spécifier à Grunt les différents bundles qui doivent être parcourus. Pour cela, il faut les indiquer dans le fichier de configuration application.config.js.

```

1 // grunt/targets/application.config.js
2
3 module.exports = {
4   application : {
5     // Listes des différents bundles qui seront parcourus par les tâches css et_
6     ↪ javascript
7     bundles: [
8       'openorchestrabackoffice',
9       'openorchestrauseradmin',
10      'openorchestragroup',
11      'openorchestralog',
12      'openorchestraworkflowadmin',
13      'openorchestramediaadmin'
14    ],
15    dest: {
16      template : 'web/built/', //web/build/template/template.js
17      menu : 'web/built/', //web/build/menu/menu.js
18      javascript : 'web/built/openorchestra/' // emplacement ou sera compiler_
19      ↪ les différents javascript
20    }
21  }
22 };

```

Ainsi lorsque vous ajoutez une sous-application JavaScript, il faut bien penser à ajouter le bundle Symfony qui contient votre sous-application à la configuration de Grunt.

## Menu de navigation

Avec Open Orchestra, les différents éléments du menu sont gérés grâce à une configuration JSON.

Cette configuration JSON doit se trouver dans le fichier Ressources/public/config/menu.json de votre bundle. Les fichiers menu.json des différents bundles sont rassemblés en un seul fichier (par défaut web/build/menu/menu.js) grâce à une tâche grunt.

Par exemple, voici la configuration pour ajouter une sous-entrée dans le menu contribution :

```

1 // src/AppBundle/Resources/public/config/menu.js
2 {
3   "contribution": { // Nom du menu parent de niveau 1 (contribution, configuration,
4   ↪ platform, developer)
5     "monMenu": {
6       "template": "Menu/Contribution/monMenu", // Template underscore utilisé pour_
7       ↪ le rendu
8       "rank": 0 // rang du menu par rapport au autre entrée
9     },
10  }
11 }

```

```

1 <!-- src/AppBundle/Resources/public/template/Menu/contribution/monMenu._tpl.html -->
2
3 <li>
4   <a href="#"&#x2013; Backbone.history.generateUrl('monMenu') %>" id="navigation-mon_menu
5   ↪ ">

```

```
5     Mon menu
6     </a>
7 </li>
```

### Surcharges

Pour des besoins spécifiques à un projet, il peut être nécessaire de surcharger une classe (Model, View, Router, etc) JavaScript définis par Open Orchestra.

Afin de surcharger une classe JavaScript sur Open Orchestra, il faut bien comprendre comment sont compilés et concaténés les différents fichiers de l'application et des sous-applications JavaScript par la tâche Grunt.

Avant de concaténer les différents fichiers la tâche Grunt les copies tous dans un même dossier (par défaut web/built/openorchestra/js, cf la configuration Grunt).

Par exemple, si l'on prend deux fichiers de deux sous-applications JavaScript différentes

```
BackofficeBundle/Resources/public/ecmascript/OpenOrchestra/Application/View/
AreaView.js` `
```

et

```
MediaAdminBundle/Resources/public/ecmascript/OpenOrchestra/Application/View/
MediasView.js
```

lors de l'exécution de la tâche Grunt, ils seront tous les deux déplacés dans le même dossier

```
web/built/openorchestra/js/OpenOrchestra/Application/View/
```

---

**Note :** L'ordre dans lequel les fichiers des applications sont copiés est défini par l'ordre de chargement des bundles fourni dans la configuration de Grunt (grunt/targets/application.config.js)

---

Ainsi, si il y a besoin de surcharger un fichier javascript, il suffit de mettre le nouveau fichier dans la même structure de dossier dans la sous-application et de modifier la configuration Grunt (grunt/targets/application.config.js) pour charger son bundle après celui que l'on désire surcharger.

Par exemple si l'on veut surcharger BackofficeBundle/Resources/public/ecmascript/OpenOrchestra/Application/View/AreaView.js, il suffit de créer un fichier AreaView.js dans la même structure de dossier dans votre sous-application JavaScript, c'est à dire AppBundle/Resources/public/ecmascript/OpenOrchestra/Application/View/AreaView.js.

## Contenu, Type de contenu et champ personnalisé

### Présentation des concepts

OpenOrchestra prend en charge un certain nombre de fonctionnalités communes à la gestion de contenus : processus de validation, versions, suppression réversible, droits d'accès, intégrité... Un soin tout particulier doit donc être apporté lors de la modélisation technique d'un projet afin d'optimiser cette prise en charge et donc l'utilisation d'OpenOrchestra. Cette notion de contenu peut et doit dépasser la notion habituelle de contenu éditorial. La possibilité de déconnecter les fonctionnalités de processus de validation et de versions permet d'utiliser ces contenus comme des données de référence, pour une liste de départements par exemple.

La description d'un contenu est portée par l'objet type de contenu. Celui-ci bénéficie également d'un certain nombre de fonctionnalités natives : processus de validation, versions, suppression réversible, droits d'accès... Il est chargé de décrire les différents champs permettant la contribution des contenus héritant de cet objet. Par exemple, le type

de contenu voiture spécifiera que les contenus en dérivant sont constitués d'un nom, champ de type texte, d'une description, champ de type tinyMce, d'une marque, champ de type sélecteur... Ci-dessous, les écrans du backoffice correspondant à la création de ce type de contenu.

The screenshot shows the 'PROPRIÉTÉS' (Properties) configuration screen for a content type. The interface includes a dark sidebar with navigation icons and a main content area with the following sections:

- PROPRIÉTÉS \* CHAMPS**: Tabbed interface with 'PROPRIÉTÉS' selected.
- Propriétés**:
  - Libellé back-office \*: Input field with value 'Voiture' and language selectors for EN and FR.
  - Id \*: Input field with value 'car'.
  - Activer le workflow: Toggle switch set to 'Oui'.
  - Autoriser les versions: Toggle switch set to 'Oui'.
- Personnalisation**:
  - Template personnalisé de formulaire d'édition: Empty input field.
- Partage des contenus**:
  - Partagé sur les autres sites: Toggle switch set to 'Oui'.
  - Toujours (ou à la demande): Toggle switch set to 'Non'.
- Éléments à afficher dans la liste back-office**:
  - Nom: Toggle switch set to 'Oui'.
  - Date de création: Toggle switch set to 'Non'.
  - Date de mise à jour: Toggle switch set to 'Oui'.
  - Etat workflow: Toggle switch set to 'Oui'.
  - Partagé sur les autres sites: Toggle switch set to 'Non'.
  - Auteur: Toggle switch set to 'Oui'.
  - Auteur de la mise à jour: Toggle switch set to 'Non'.

The screenshot shows the 'CHAMPS' (Fields) configuration screen for a content type named 'CAR'. The interface includes a dark sidebar and a main content area with the following elements:

- DÉVELOPPEUR > TYPE DE CONTENU**: Breadcrumb navigation.
- Retour à la liste**: Button to return to the list.
- CAR**: Content type name.
- PROPRIÉTÉS \* CHAMPS**: Tabbed interface with 'CHAMPS' selected.
- Buttons: Fermer la liste, Déployer la liste, and Ajouter un champ.
- Table of Fields**:
 

Libellé	Id du champ	Type	Champ obligatoire		
Nom	name	Ligne de texte	Oui		
Description	description	Texte riche	Non		
Marque	brand	Choix	Oui		
- \* Champ obligatoire**: Legend for the mandatory field indicator.
- Buttons: Sauvegarder (Save) and Supprimer (Delete).

OpenOrchestra vient avec un certain nombre de types de champs pré-définis :

- Choix (champ select avec liste des options à renseigner)
- Choix d'un contenu (champ select avec une liste filtrée de contenus)

- Date (champ date avec calendrier ou trois champs texte numériques ...)
- Adresse email (champ texte avec vérification du formatage email)
- Champ caché
- Entier (champ texte numérique)
- Monnaie (champ texte numérique avec 2 décimales)
- Ligne de texte (champ de type texte)
- Zone de texte (champ de type textarea)
- Texte riche (tinyMce)
- Média (champ permettant de sélectionner un élément de la médiathèque)

L'une des tâches récurrentes pour l'intégrateur sera d'ajouter à cette liste de nouveaux types de champ en fonction des spécificités de son projet.

### Ajout d'un type de champ



Voici le détail pour ajouter un type de champ case à cocher à notre précédent exemple pour pouvoir spécifier si une voiture est en ligne ou non.

```
1 #app/config.yml
2 open_orchestra_backoffice:
3     field_types:
4         online_checkbox:
5             # le label utilisé dans le sélecteur de type dans le type de contenu
6             label: Case à cocher
7             # le FormType utilisé dans le formulaire du contenu (namespace ou alias)
8             type: Symfony\Component\Form\Extension\Core\Type\CheckboxType
9             # le type du format de désérialisation (voir ↪
10 ↪JMS\Serializer\GraphNavigator)
11             deserialize_type: string
12             # la stratégie de recherche dans le tableau des contenus
13             search: online_checkbox
14             options:
15                 required:
16                     # la valeur par défaut de l'option
17                     default_value: false
18                 value:
19                     # la valeur par défaut de l'option
20                     default_value: online
21             default_value:
22                 # le FormType utilisé pour contribuer la valeur par défaut du champ
23                 type: Symfony\Component\Form\Extension\Core\Type\CheckboxType
24                 options:
25                     # le label utilisé pour la valeur par défaut
26                     label: Valeur par défaut
27                     # l'option obligatoire ou non pour la valeur par défaut
28                     required: false
29             options:
30                 value:
31                     # le FormType utilisé pour contribuer la valeur de l'option
32                     type: Symfony\Component\Form\Extension\Core\Type\TextType
33                     # le label de l'option
34                     label: Valeur associée
35                     # le caractère obligatoire ou non de cette option
36                     required: true
```


On obtient ainsi dans le formulaire de type de contenu :




**Propriétés**

Libellé  EN  FR

online

Id du champ  online

Type Case à cocher 

**Paramètres**

Champ obligatoire Non  Oui

Valeur associée online

Valeur par défaut Non  Oui

Et dans le formulaire d'un contenu voiture :

online Non  Oui

On remarque dans l'exemple la définition d'une nouvelle option "value" contribué à l'aide d'un TextType, ayant pour valeur par défaut "online" et qui sera passer à l'OptionResolver de notre checkbox. OpenOrchestra vient avec un certain nombre d'options pré-définies :

- max\_length
- required
- grouping
- rounding\_mode
- multiple
- expanded
- choices
- currency
- precision
- format
- widget
- input
- content\_search (pour `OpenOrchestra\Backoffice\Form\Type\Component\ContentChoiceType`)

## Tableau de consultation

Lors de la visualisation de ces contenus sous formes de tableau, il est nécessaire de mettre en place la brique permettant de transformer les différentes propriétés du contenu sous forme de chaîne de caractères.

Cela se fait par la mise en place d'une stratégie de transformation de la propriété implémentant l'interface `OpenOrchestra\Backoffice\ValueTransformer\ValueTransformerInterface` et gérée par `OpenOrchestra\Backoffice\ValueTransformer\ValueTransformerManager`.

L'inscription se fait automatiquement lors de la passe de compilation en définissant la stratégie comme un service taggué `open_orchestra_backoffice.value_transformer.strategy`.

Cette représentation de la propriété sous forme de chaîne est générée à la création ou à la modification du contenu et pas à la volée lors de sa consultation.

Voici le code du transformer :

```
1 // src/AcmeBundle/ValueTransformer/OnlineCheckboxToHtmlStringTransformer.php
2 namespace AcmeBundle\ValueTransformer\Strategies;
3
4 use OpenOrchestra\Backoffice\ValueTransformer\ValueTransformerInterface;
5
6 /**
7  * Class OnlineCheckboxToHtmlStringTransformer
8  */
9 class OnlineCheckboxToHtmlStringTransformer implements ValueTransformerInterface
10 {
11     /**
12      * @param array $data
13      *
14      * @return string
15      */
16     public function transform($data)
17     {
18         return ($data) ?
19             '<i aria-hidden="true" class="fa fa-check text-success"></i>' :
20             '<i aria-hidden="true" class="fa fa-close text-danger"></i>';
21     }
22
23     /**
24      * @param string $fieldType
25      * @param mixed $value
26      *
27      * @return bool
28      */
29     public function support($fieldType, $value)
30     {
31         return gettype($value) == 'boolean' && ($fieldType == 'online_checkbox');
32     }
33
34     /**
35      * @return string
36      */
37     public function getName()
38     {
39         return 'online_checkbox';
40     }
41 }
```

et le paramétrage permettant de l'activer :

```
1 # app/config/services.yml
2 services:
3     acme_bundle.value_transformer.online_checkbox:
4         class: _
5         ↪AcmeBundle\ValueTransformer\Strategies\OnlineCheckboxToHtmlStringTransformer
6         tags:
7             - { name: open_orchestra_backoffice.value_transformer.strategy }
```

On obtient ainsi la liste de consultation suivante :

<input type="checkbox"/>	Nom	<input type="checkbox"/> Restreint au site	<input type="checkbox"/> Date de création	<input type="checkbox"/> Auteur	<input type="checkbox"/> Etat workflow	<input type="checkbox"/> Online	<input type="checkbox"/> Dupliquer
<input type="checkbox"/>	DS 3 fr	<input checked="" type="checkbox"/>	02/06/2017 11:18:15	admin	Brouillon	<input checked="" type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	206 3 portes fr	<input checked="" type="checkbox"/>	02/06/2017 11:18:15	admin	Brouillon	<input checked="" type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	R5 3 portes fr	<input checked="" type="checkbox"/>	02/06/2017 11:18:15	admin	Brouillon	<input checked="" type="checkbox"/>	<input type="checkbox"/>

## Moteur de filtres

Dans le YAML permettant d'ajouter le type de champ case à cocher, le paramètre `open_orchestra_backoffice.field_types.online_checkbox.search` sert à gérer entre autres l'affichage dans le moteur de filtres. La première étape est de créer la classe js permettant de générer l'affichage.

```

1 // src/AcmeBundle/Ressources/public/ecmascript/Acme/Service/SearchFormGroup/
  ↳OnlineCheckboxForm.js
2 import TemplateManager      from '../../../../../../OpenOrchestra/Service/TemplateManager
  ↳'
3 import AbstractSearchFormGroup from '../../../../../../OpenOrchestra/Service/
  ↳SearchFormGroup/AbstractSearchFormGroup'
4
5 /**
6  * @class OnlineCheckboxForm
7  */
8 class OnlineCheckboxForm extends AbstractSearchFormGroup
9 {
10     /**
11      * test if field is supported
12      *
13      * @param {Object} field
14      */
15     support(field) {
16         // check on the value setted in the yml
17         return field.search == 'online_checkbox';
18     }
19
20     /**
21      * render the field
22      *
23      * @param {Object} field
24      */
25     render(field) {
26         return TemplateManager.get('SearchFormGroup/onlineCheckboxForm') ({
27             field: field
28         });
29     }
30 }
31
32 // unique instance of OnlineCheckboxForm
33 export default (new OnlineCheckboxForm);

```

Ensuite il faut enregistrer cette classe auprès du manager responsable de son exploitation (pour plus de détail, voir la partie client js).

```

1 // src/AcmeBundle/Ressources/public/ecmascript/Acme/Application/AcmeSubApplication.js
2 import SearchFormGroupManager from '../../../../../../OpenOrchestra/Service/
  ↳SearchFormGroup/Manager'

```

```

3 import CheckboxSearchFormGroup from '../../../../../../OpenOrchestra/Service/
  ↳ SearchFormGroup/OnlineCheckboxForm'
4
5 /**
6  * @class AcmeSubApplication
7  */
8 class AcmeSubApplication
9 {
10     /**
11     * Run sub Application
12     */
13     run() {
14         this._initSearchFormGroupManager;
15     }
16
17     /**
18     * Initialize field search library
19     * @private
20     */
21     _initSearchFormGroupManager() {
22         SearchFormGroupManager.add(CheckboxSearchFormGroup);
23     }
24 }

```

Puis il faut créer le template d'affichage SearchFormGroup/onlineCheckboxForm.

```

1 <!-- src/AcmeBundle/Ressources/public/template/SearchFormGroup/onlineCheckboxForm._
  ↳ tpl.html -->
2 <label for="attributes.online" class="control-label col-md-4">
3     Online
4 </label>
5 <div class="switch-button">
6     <span>Non</span>
7     <label class="switch">
8         <input id="attributes.online" name="attributes.online" value="1" type=
  ↳ "checkbox">
9         <div class="slider"></div>
10     </label>
11     <span>Oui</span>
12 </div>

```

On obtient le moteur de recherche suivant.

The screenshot shows a search interface with the following elements:

- Input fields for "Nom" (Name), "Auteur" (Author), "Date de mise à jour" (Update date), and "Description".
- A dropdown menu for "Etat workflow" (Workflow status) with a downward arrow icon.
- A toggle switch for "Online" with "Non" (No) and "Oui" (Yes) options.
- A blue button labeled "Filtrer les résultats" (Filter results).

## Requête de filtres

Enfin, les données du moteur de recherche vont être, à la soumission, envoyées à l'API pour retourner les contenus correspondants. L'API va donc créer la requête permettant de filtrer les contenus. Cela se fait au niveau de la requête de repository `findForPaginateFilterByContentTypeSiteAndLanguage` de votre `ContentRepository`.

**Note :** Si vous utilisez les bundle mongo, alors une mécanique a été mise en place pour pouvoir enrichir facilement la recherche.

Création du trait de filtrage :

```

1 // src/AcmeBundle/Pagination/MongoTrait/FilterTypeStrategy/Strategies/
  ↳OnlineCheckboxFilterStrategy.php
2 namespace AcmeBundle\Pagination\MongoTrait\FilterTypeStrategy\Strategies;
3
4 use OpenOrchestra\Pagination\FilterType\FilterTypeInterface;
5
6 /**
7  * Class OnlineCheckboxFilterStrategy
8  */
9 class OnlineCheckboxFilterStrategy implements FilterTypeInterface
10 {
11     const FILTER_TYPE = 'online_checkbox';
12
13     /**
14      * @param string $type
15      *
16      * @return bool
17      */
18     public function support($type)
19     {
20         return $type === self::FILTER_TYPE;
21     }
22
23     /**
24      * @param string $name
25      * @param string $value
26      * @param string $documentName
27      * @param string $format
28      *
29      * @return array
30      */
31     public function generateFilter($name, $value, $documentName='', $format='')
32     {
33         if ($value === 'true' || $value === '1') {
34             return array($name => true);
35         } elseif ($value === 'false' || $value === '0') {
36             return array($name => false);
37         }
38
39         return null;
40     }
41
42     /**
43      * @return string
44      */

```

```
45 public function getName ()
46 {
47     return 'online_checkbox_filter';
48 }
49 }
```

Enregistrement du trait auprès du manager qui construit la requête dans le repository à l'aide d'un service taggué.

```
1 # app/config/services.yml
2 services:
3     acme_bundle.value_transformer.online_checkbox:
4         class: ◻
5         ↪AcmeBundle\Pagination\MongoTrait\FilterTypeStrategy\Strategies\OnlineCheckboxFilterStrategy
6         tags:
7             - { name: open_orchestra_pagination.filter_type.strategy }
```

## Contribution à la documentation

### note ::

---

**Note :** Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras ac arcu ligula. Nulla molestie neque eget justo blandit, ac laoreet tellus tristique. Sed libero nunc, tincidunt id accumsan sed, porttitor eu mauris. In blandit leo id mauris egestas laoreet. Aenean nisi ex, viverra at tempor quis, placerat at nisi. Suspendisse potenti. Mauris urna eros, pretium id sodales non, lobortis a est.

```
1 # for example, if WAMP is used ...
2 c:\> move symfony c:\wamp\bin\php
3 # ... then, execute the command as:
4 c:\> symfony
```

### tip ::

---

**Astuce :** Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras ac arcu ligula. Nulla molestie neque eget justo blandit, ac laoreet tellus tristique. Sed libero nunc, tincidunt id accumsan sed, porttitor eu mauris. In blandit leo id mauris egestas laoreet. Aenean nisi ex, viverra at tempor quis, placerat at nisi. Suspendisse potenti. Mauris urna eros, pretium id sodales non, lobortis a est.

---

### caution ::

**Prudence :** then *three* services have been created (the automatic service + your two services) and the automatically loaded service will be passed - by default - when you type-hint `SiteUpdateManager`. That's why creating the alias is a good idea.

**code-block : : ini**

```

1 ; Linux and macOS systems
2 curl.cainfo = "/path/to/cacert.pem"
3
4 ; Windows systems
5 curl.cainfo = "C:\path\to\cacert.pem"

```

**code-block : : text**

```

1 http://localhost:8000/config.php

```

**code-block : : terminal**

```

1 # Linux and macOS systems
2 $ sudo mkdir -p /usr/local/bin
3 $ sudo curl -Ls https://symfony.com/installer -o /usr/local/bin/symfony
4 $ sudo chmod a+x /usr/local/bin/symfony

```

**code-block : : yaml**

```

1 # config.yml
2
3 # FOSUserBundle
4 fos_user:
5     db_driver: mongodb
6     firewall_name: main
7     user_class: OpenOrchestra\UserBundle\Document\User
8     group:
9         group_class: OpenOrchestra\GroupBundle\Document\Group

```

**code-block : : javascript**

```

1 module.exports = function(grunt) {
2     var appConfig = require('./grunt/app_config.js');
3     var GruntConfigBuilder = require(appConfig.GruntConfigBuilder);
4
5     GruntConfigBuilder.init(grunt, appConfig);
6 };

```

**code-block : : bash**

```

1 ./bin/grunt

```

## code-block :: php

```
1 class AppKernel extends Kernel
2 {
3     // ...
4
5     public function registerBundles()
6     {
7         $bundles = array(
8             // others bundles
9             new Doctrine\Bundle\MongoDBBundle\DoctrineMongoDBBundle(),
10            new FOS\HttpCacheBundle\FOSHttpCacheBundle(),
11        );
12
13        // ...
14    }
15 }
```