# Open Orchestra Documentation

*Version 2.0.0*

**Open Orchestra**

**juil. 10, 2017**

# Table des matières

**1 Guide développeur** **1**
1.1 Guide développeur . . . . . 1
1.2 Contribution à la documentation . . . . . 29

CHAPITRE 1

# Guide développeur

# 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'environement Docker au sein du même dossier.

Création des applications Front et Back office :

```
$ cd my_project_name/

# création des applications
$ composer create-project open-orchestra/open-orchestra open-orchestra --ignore-
↪platform-reqs --no-scripts
$ composer create-project open-orchestra/open-orchestra-front-demo open-orchestra-
↪front-demo --ignore-platform-reqs --no-scripts

# Dossier qui contiendra les medias de la médiathèque
$ mkdir uploaded-files
```

### Création de l'environnement docker

```
$ cd my_project_name/

# clonage de la configuration pour l'environnement docker
$ git clone git@github.com:open-orchestra/open-orchestra-provision-docker.git

$ cd open-orchestra-provision-docker/
# création des conteneurs
$ docker-compose up -d
```

### Installation des applications

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

Back office :

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

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

Front office :

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

```
open_orchestra_cms.mongodb.server: 'mongodb://oo_mongo:27017'
fos_http_cache.proxy_client.varnish.servers: [oo_varnish:6081]
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 :

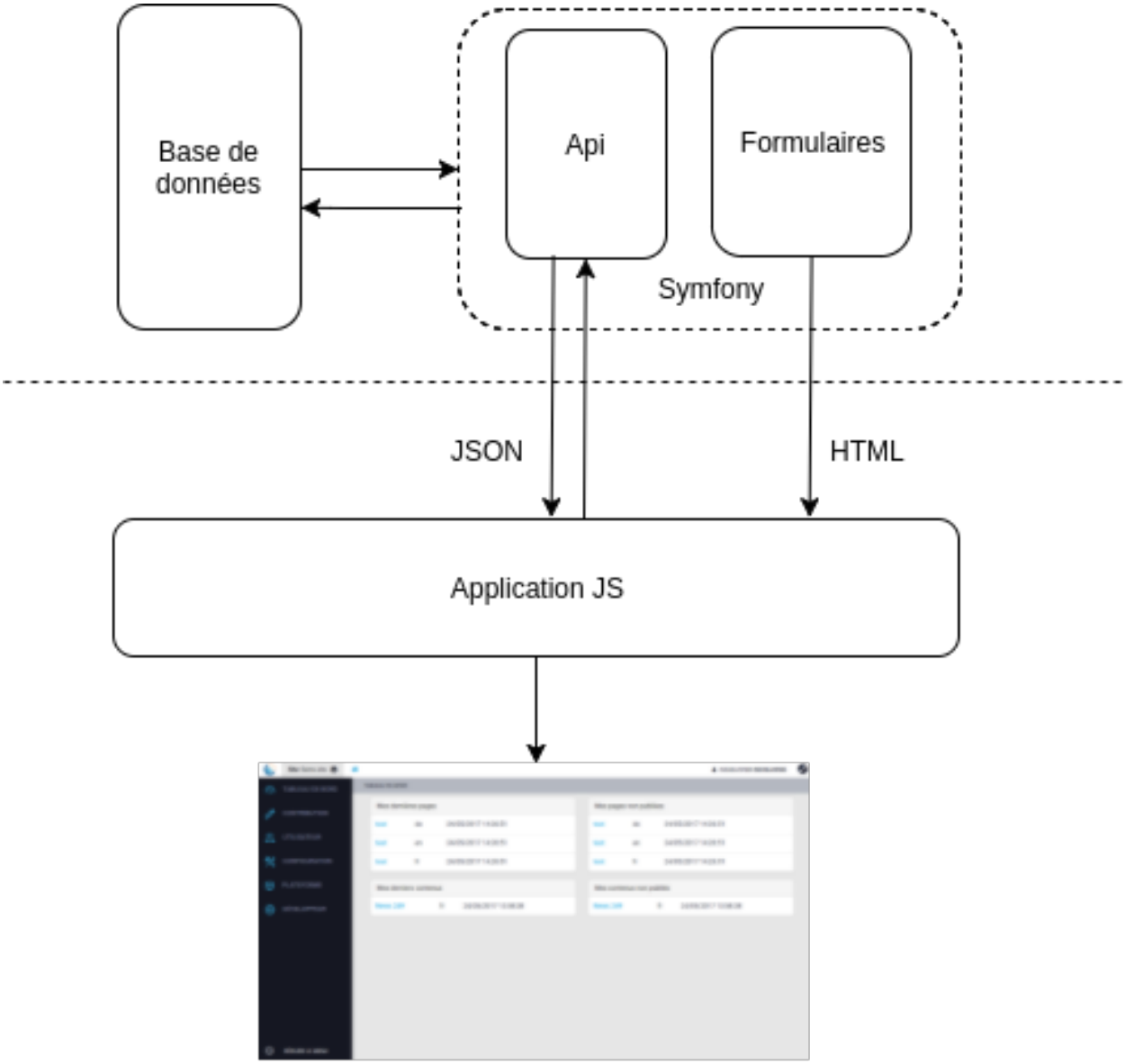
```
[IP]    admin.openorchestra.2-0.dev
[IP]    demo.openorchestra.2-0.dev
[IP]    media.openorchestra.2-0.dev
```

**Note :**

[IP] doit être remplacé par 127.0.0.1 pour Linux.
[IP] doit être remplacé par la valeur fournit 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)

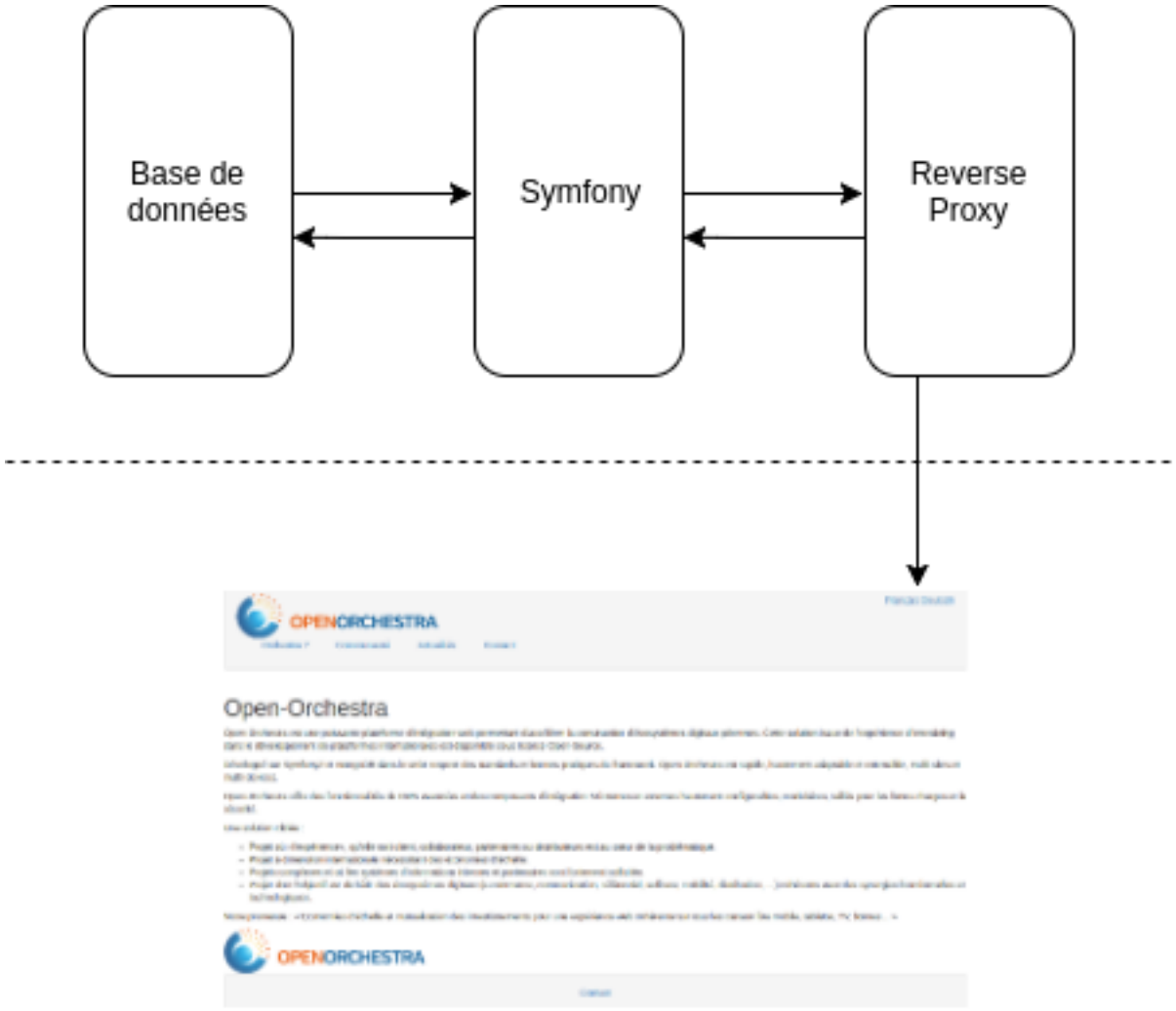
### Form

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`

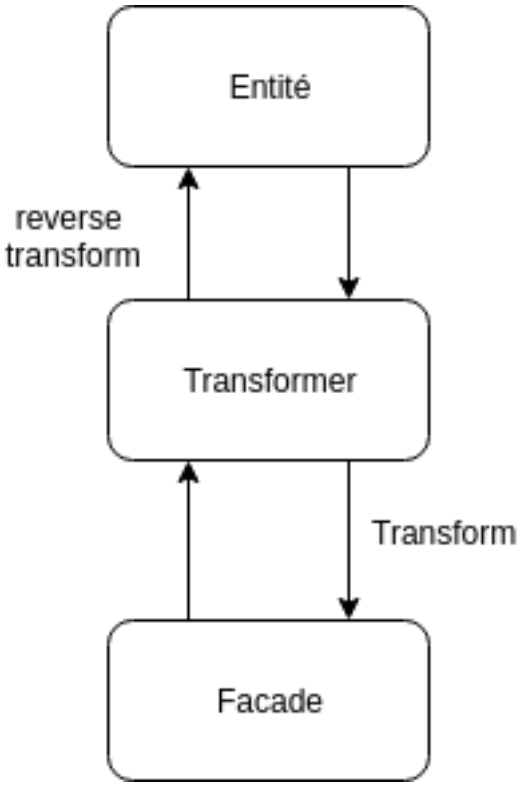
**Note :** Sur Open Orchestra la présence d'un reverse proxy n'est pas obligatoire. Si aucun reverse proxy n'est configuré sur l'application Front office, cette dernière continuera de fonctionner.

## 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 différent 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;
```

```

class Task
{
    protected $title;
    protected $dueDate;

    // ... getters
    // ... setters
}
```

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*.

```
// src/AppBundle/Facade/TaskFacade.php
namespace AppBundle\Facade;

use OpenOrchestra\BaseApi\Facade\FacadeInterface;

class TaskFacade implements FacadeInterface
{
    public $title;
    public $dueDate;
}
```

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`.

```
// src/AppBundle/Transformer/TaskTransformer.php
namespace AppBundle\Transformer;

use OpenOrchestra\BaseApi\Transformer\AbstractTransformer;

class TaskTransformer extends AbstractTransformer
{
    // Rempli les différents attributs de la facade
    // avec ceux stockés dans la tâche ($task)
    public function transform($task)
    {
        $facade = new TaskFacade();
        $facade->title = $task->getTitle();
        $facade->dueDate = $task->getDueDate();

        return $facade;
    }

    // Créé une tâche à partir des données de la facade
    public function reverseTransform($taskFacade)
    {
        $task = new Task();
        $task->setTitle($taskFacade->title);
        $task->setDueDate($taskFacade->dueDate);

        return $task;
    }

    // Nom du transformer afin de l'identifier lors de son utilisation
    public function getName()
```

```
    {
        return 'task_transformer';
    }
}
```

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*.

```
app_bundle.transformer.task:
    class: AppBundle\Transformer\TaskTransformer
    tags:
        - { 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.

```
// src/AppBundle/Controller/Api/TaskController.php
namespace AppBundle\Controller;

class TaskController extends Controller
{
    public function showAction()
    {
        // Création d'un object tâche
        // celui-ci pourrais aussi provenir d'une base de données
        $task = new Task();
        $task->setTitle('test');

        // Transformation de l'object Task en facade
        // en utilisant le transformer manager
        //
        // task_transformer est le nom du transformer défini par la
        // méthode getName de AppBundle\Transformer\TaskTransformer
        $facade = $this
            ->get('open_orchestra_api.transformer_manager')
            ->get('task_transformer')
            ->transformer($task);
    }
}
```

### 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.

```
// src/AppBundle/Facade/TaskFacade.php
namespace AppBundle\Facade;

use OpenOrchestra\BaseApi\Facade\FacadeInterface;
use JMS\Serializer\Annotation\Type;

class TaskFacade implements FacadeInterface
{
```

```
    /**
     * @Type("string")
     */
    public $title;

    /**
     * @Type("DateTime")
     */
    public $dueDate;
}
```

**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.

```
// src/AppBundle/Controller/Api/TaskController.php
namespace AppBundle\Controller;

class TaskController extends Controller
{
    public function showAction()
    {
        $task = new Task();
        $task->setTitle('test');

        $facade = $this
            ->get('open_orchestra_api.transformer_manager')
            ->get('task_transformer')
            ->transformer($task);

        // appel au service JMSSerializerBundle
        $serializer = $container->get('jms_serializer');

        // sérialisation de la *facade* en JSON
        $content = $serializer->serialize($facade, 'json');

        // Création d'une réponse Symfony
        return  new Response(
            serializer
            200,
            array('content-type' => 'application/json')
        )
    }
}
```

**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`.

```php
// src/AppBundle/Controller/Api/TaskController.php
namespace AppBundle\Controller;

use OpenOrchestra\BaseApiBundle\Controller\Annotation as Api;

class TaskController extends Controller
{
    /**
     * @Api\Serialize()
     */
    public function showAction()
    {
        $task = new Task();
        $task->setTitle('test');

        $facade = $this
            ->get('open_orchestra_api.transformer_manager')
            ->get('task_transformer')
            ->transformer($task);

        return $facade;
    }
}
```

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.

```php
// src/AppBundle/Controller/Api/TaskController.php
namespace AppBundle\Controller;

use OpenOrchestra\BaseApiBundle\Controller\Annotation as Api;

class TaskController extends Controller
{
    /**
     * @Api\Serialize()
     *
     * Indique que l'action "show" a pour contexte le groupe SHOW
     * @Api\Groups({"show_dueDate"})
     */
    public function showAction()
    {
        // ...
    }
}
```

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

```php
/**
 * @Api\Serialize()
 *
 * @Api\Groups({"AppBundle\Context\ApiContext::SHOW_DUE_DATE"})
 */
public function showAction()
{
    // ...
}
```

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

```php
// src/AppBundle/Transformer/TaskTransformer.php
namespace AppBundle\Transformer;

use OpenOrchestra\BaseApi\Transformer\AbstractTransformer;

class TaskTransformer extends AbstractTransformer
{
    // Rempli les différents attributs de la facade
    // avec ceux stockés dans la tâche ($task)
    public function transform($task)
    {
        $facade = new TaskFacade();
        $facade->title = $task->getTitle();

        // l'attribut dueDate sera ajouté à la facade uniquement
        // si l'action qui demande la transformation appartient au groupe
        // show_dueDate sinon il ne sera pas ajouté à la facade et donc ne
        // sera pas sérialisé
        if ($this->hashGroup("show_dueDate")) {
            $facade->dueDate = $task->getDueDate();
        }

        return $facade;
    }

    // ...
}
```

### 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*.

```php
// src/AppBundle/Controller/Api/TaskController.php
namespace AppBundle\Transformer;

class TaskController extends Controller
{
```

```
    public function editAction(Request $request)
    {
        // Désérialisation du contenu de la requête
        // dans la facade TaskFacade
        $facade = $this
            ->get('jms_serializer')
            ->deserialize(
                $request->getContent(),
                'AppBundle\Facade\TaskFacade',
                $request->get('_format', 'json')
            );

        // Utilisation du transformer manager pour récupérer
        // le transformer task
        // @see AppBundle\Transformer\TaskTransfomer
        $task = $this
            ->get('open_orchestra_api.transformer_manager')
            ->get('task_transformer')
            ->reverseTransform($facade);

        // ...
        // Validation de l'entité task
        // Sauvegarde de l'entité
    }
}
```

## 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.

```
// BackofficeBundle/Resources/public/ecmascript/OpenOrchestra/Application/Application.
↪js

class Application
{
    // ...

    /**
     * Run Application
```

```
     */
    run() {
        // Initialisation des paramètres de configuration
        // de l'application
        this._initConfiguration();

        // ...

        // Initialisation des différents routeurs Backbone
        this._initRouter();

        // Événement lancé avant le démarrage de l'application
        Backbone.Events.trigger('application:before:start');

        // Initialisation des layouts de l'application (menu, header, etc)
        this._initLayoutView();

        // Démarrage de l'application
        Backbone.history.start();

        // Événement lancé après le démarrage de l'application
        Backbone.Events.trigger('application:after:start');
    }

    // ...
}
```

### 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.

```
// src/AppBundle/Resources/public/main.js

// Objet représentant la sous-application
import AppSubApplication from './Application/AppSubApplication'

$(() => {
     Backbone.Events.on('application:before:start', () => {
        // Démarrage de la sous-application (router, ajout de configuration, etc)
        AppSubApplication.run();
     });
});
```

### Structure

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

```
- Ressources/public
    |
```

```
    - config  # Fichiers json de configuration de l'application (menu, etc)
    |
    - ecmascript # Classe Javascript es6 de l'application
    |   |
    |   - OpenOrchestra # "Namespace"
    |       |
    |       - Application
    |       |    - Collection # Collections backbone
    |       |    - Model      # Models backbone
    |       |    - Router     # Routeurs backbone
    |       |    - Views      # Vue Backbones
    |       |
    |       - Service # Classes génériques,
    |       |           # utilisées par différents éléments de l'application
    |       |
    |       - main.js # Démarrage de l'application ou sous-application
    |
    - 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 on été étendus afin d'ajouter une méthode `preinitialize` pour définir ces propriétés.

```
class CustomView extends Backbone.View {

    preinitialize() {
        this.tagName = "li";
        this.className = "custom-class";
    }

    initialize() {
        //...
    }

    render() {
        //...
    }
}
```

**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 on 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.

```
// src/AppBundle/Resources/public/MyApp/Application/View/AppView.js
import OrchestraView from '../../../../OpenOrchestra/Application/View/OrchestraView'
class AppView extends OrchestraView {}

// src/AppBundle/Resources/public/MyApp/Application/Router/AppRouter.js
import OrchestraView from '../../../../OpenOrchestra/Application/Router/
↪OrchestraRouter'
class AppRouter extends OrchestraRouter {}

// src/AppBundle/Resources/public/MyApp/Application/Collection/AppCollection.js
import OrchestraView from '../../../../OpenOrchestra/Application/Collection/
↪OrchestraCollection'
class AppCollection extends OrchestraCollection {}

// src/AppBundle/Resources/public/MyApp/Application/Model/AppModel.js
import OrchestraView from '../../../../OpenOrchestra/Application/Model/OrchestraModel'
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 JavasSript, 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)`.

```
// src/AppBundle/Resources/public/MyApp/Application/View/AppView.js
import OrchestraView from '../../../../OpenOrchestra/Application/View/OrchestraView'

class AppView extends OrchestraView {

    // ...

    render() {
        let template = this._renderTemplate('helloView', {
            name: 'Foo'
        });
        this.$el.append(template);
    }
}
```

```
<!-- src/AppBundle/Resources/public/template/helloView._tpl.html -->

<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'interieur d'un autre template.

```
<!-- src/AppBundle/Resources/public/template/helloView._tpl.html -->

<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 `/Ressources/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`.

```
// grunt/targets/application.config.js

module.exports = {
    application : {
        // Listes des différents bundles qui seront parcourus par les tâches css et javascript
        bundles: [
            'openorchestrabackoffice',
            'openorchestrauseradmin',
            'openorchestragroup',
            'openorchestralog',
            'openorchestraworkflowadmin',
            'openorchestramediaadmin'
        ],
        dest: {
            template : 'web/built/', //web/build/template/template.js
            menu : 'web/built/', //web/build/menu/menu.js
            javascript : 'web/built/openorchestra/' // emplacement ou sera compiler les différents javascript
        }
    }
};
```

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 fichier `menu.json` des différents bundles sont rassemblés en un seul fichier (par défaut `web/build/menu/menu.js`)grâce a une tâche grunt.

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

```
// src/AppBundle/Resources/public/config/menu.js
{
  "contribution": { // Nom du menu parent de niveau 1 (contribution, configuration, platform, developer)
    "monMenu": {
      "template": "Menu/Contribution/monMenu",  // Template underscore utilisé pour le rendu
      "rank": 0 // rang du menu par rapport au autre entrée
    },
  }
}
```

```
<!-- src/AppBundle/Resources/public/template/Menu/contribution/monMenu._tpl.html -->

<li>
    <a href="#<%- Backbone.history.generateUrl('monMenu') %>" id="navigation-mon_menu">
```

```
        Mon menu
    </a>
</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 JavasScript 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`.

## Pages, Templates

Les pages sont composées de zones et de blocs qui seront visibles par les utilisateurs en Front office. Une page est versionable, tradusible et soumis au workflow définis dans le Back office.

**Sur Open Orchestra, les templates disponibles utilisés par une page sont regroupés dans un conteneur,** appelé *templateSet*.

---

**Note :** Un *template set* est sélectionné lors de la création d'un site. Ainsi toutes les pages de cd site peuvent utiliser les templates du *template set* sélectionné.

---

La création d'un nouveau *template set* se découpe en deux parties : Back office et Front office.

### Configuration Back office

### Configuration d'un template set

La configuration d'un *template set* ŝ'effectue simplement avec de la configuration YAML.

```yaml
# app/config.yml

# ...

open_orchestra_backoffice:
   template_set:
        default_template_set: # identifiant du template set
            label: open_orchestra_backoffice.template_set.default.label # clé de traduction du label
            styles: [] # Liste des styles qui peuvent être utilisés dans les blocs
            templates: [] # Liste des templates appartenant au template set
```

### Création d'un template

```yaml
# app/config.yml

# ...

open_orchestra_backoffice:
   template_set:
        default_template_set: # Identifiant du template set
            label: app.template_set.default.label # clé de traduction du label
            styles: [] # Liste des styles qui peuvent être utilisés dans les blocs
            templates:
                my_custom_template: # Identifiant du template
                    areas: # Liste des aires du template contribuable
                        - main
                    label: app.template_set.default.template_name.my_custom_template # clé de traduction du label
                    # Chemin du fichier qui représente le template utilisé lors de la contribution d'un node
                    path: /bundles/app/templateSet/default_template_set/my_custom_template.html
```

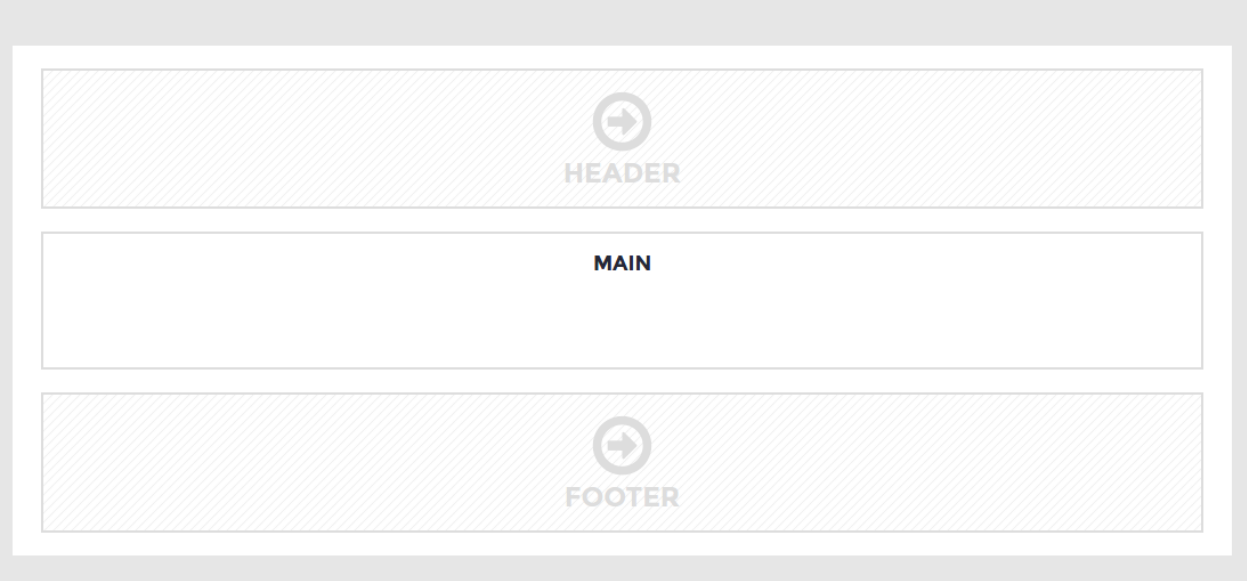
Le fichier fournis par le `path` est simplement un fichier HTML qui est utilisé afin d'avoir un rendu visuelle du template lors de la contribution des pages utilisant ce template. Par exemple voici le fichier utilisé pour **my_custom_template** qui est un template avec **trois zones** (header, main et footer) où main est une zone contribuable.

```html
<!-- src/AppBundle/Resources/public/templateSet/default_template_set/my_custom_template.html -->

<div class="row">
    <div class="col-lg-12">
        <!-- Ajout de la classe disabled pour désactiver la contribution de l'aire -->
        <div class="area header disabled">
            <span>Header</span>
            <div class="block-container"></div>
        </div>
    </div>
</div>
```

```
<div class="row">
    <div class="col-lg-12">
        <!-- data-aire-id obligatoire pour les zones contribuables -->
        <div class="area " data-area-id="main">
            <span>Main</span>
            <div class="block-container"></div>
        </div>
    </div>
</div>
<div class="row">
    <div class="col-lg-12">
        <div class="area footer disabled">
            <span>Footer</span>
            <div class="block-container"></div>
        </div>
    </div>
</div>
```

**Note :** Au sein de ce fichier, vous pouvez utiliser la représentation que vous désirez. Afin d'être reconnus les conteneurs des aires contribuables doivent posséder l'attribut `data-area-id` avec le nom de l'aire comme valeur.

Voici le rendu Back office lorsque le template `my_custom_template` est utilisé par une page :




### Configuration Front office

Une fois la configuration Back office effectuée il faut réaliser le template qui sera utilisé pour le rendu front de la page.

Dans un premier temps, il faut créer un template twig ou d'un autre type suivant le moteur de template utilisé pour votre application Symfony.

```
{# src/AppBundle/Resources/views/Template/Default/my_custom_template.html.twig #}

{% extends 'OpenOrchestraFrontBundle:Node:base_node.html.twig' %}

```

```
{% block body %}
    <div class="container">
        <header class="header">
            Mon header
        </header>

        <main class="row">
            {{ render_area('main', node, {'parameters': parameters }) }}
        </main>

       <footer class="footer">
            Mon footer
       </footer>
    </div>
{% endblock %}
```

**Note :** Il est recommandé d'étendre le template `base_node.html.twig` qui permet de définir les différentes balises contenues dans `<head>` (title, meta, etc) en fonction de la page affichée.

Le template possède une zone contribuable en Back office, la zone `main`. Afin d'afficher cette zone et ces différents blocs contribués en Back Office, il existe la fonction twig `render_area`.

Pour finir, il faut indiquer dans la configuration que le template `my_custom_template` doit utiliser le fichier `my_custom_template.html.twig` pour effectuer son rendu.

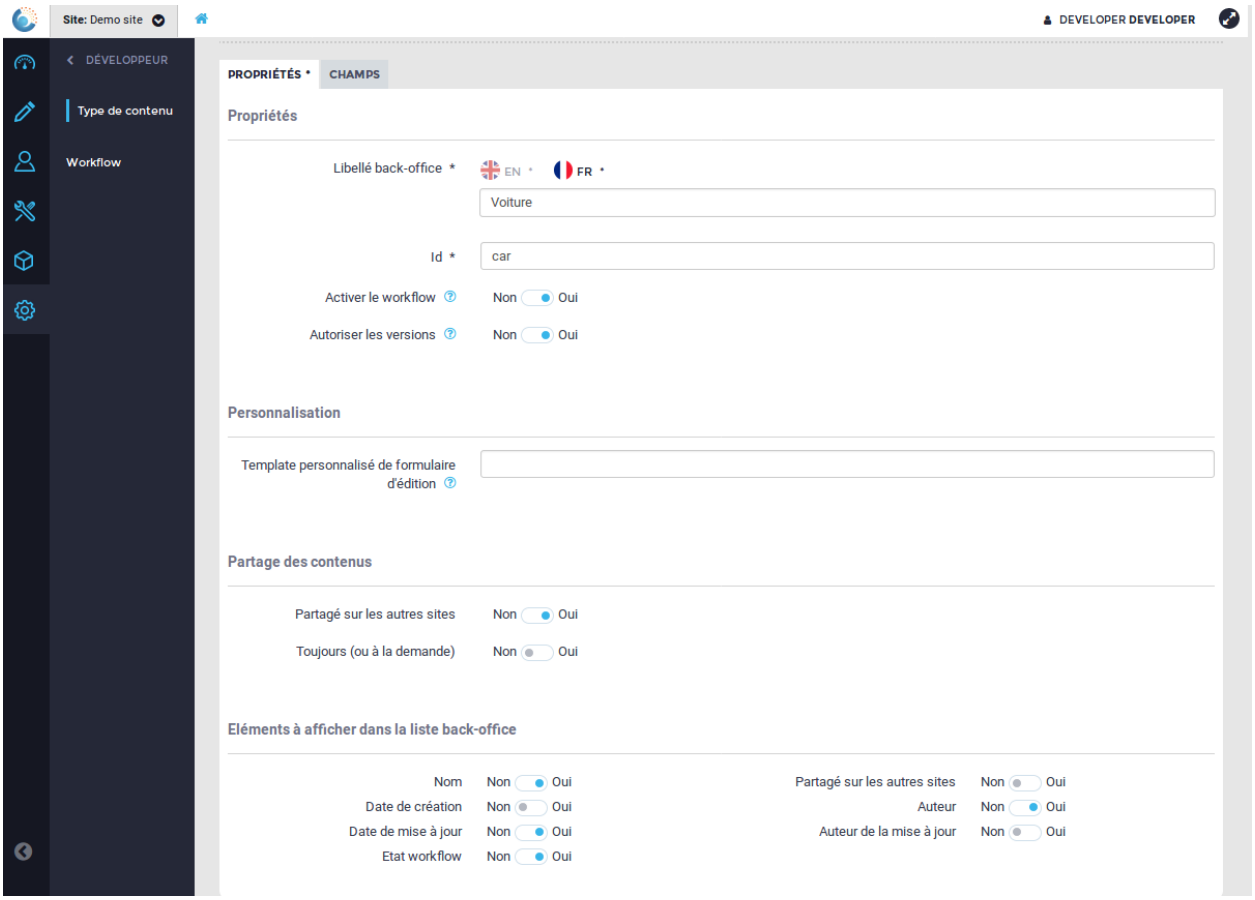
```
# app/config.yml

# ...

open_orchestra_front:
   template_set:
        default_template_set: # Identifiant du template set
            templates:
                my_custom_template: 'AppBundle:Template/Default:my_custom_template.
↪html.twig'
```

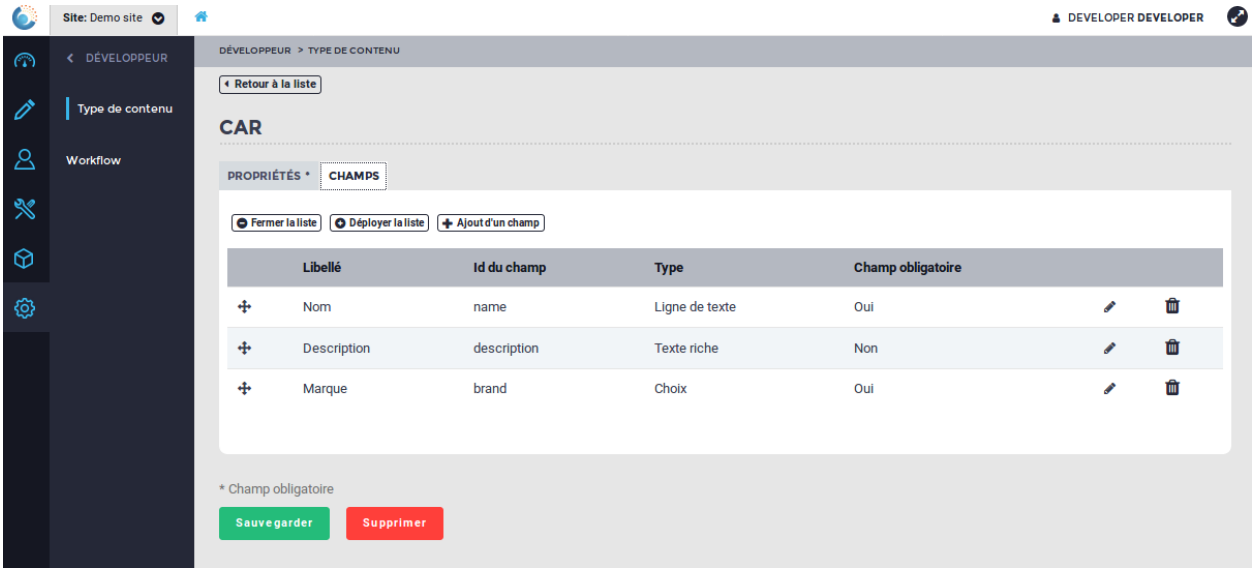
## Contenu, Type de contenu et champ personnalisé

### 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.




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)

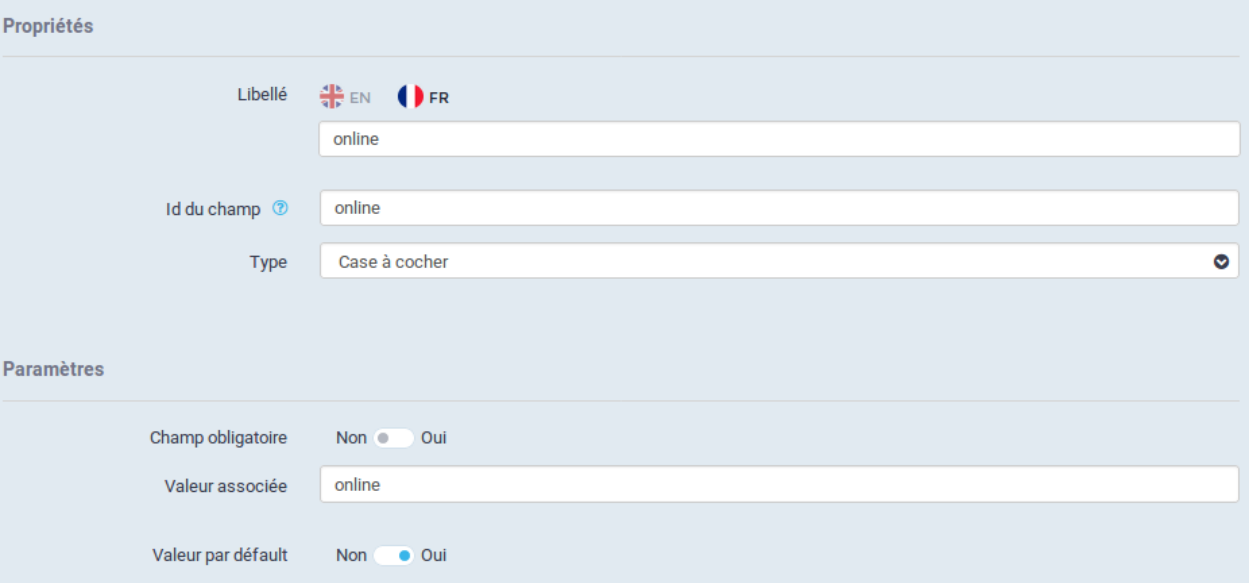
L'une des tâches récurrentes pour l'intégrateur sera d'ajouter à cette liste de nouveaux types de champ en fonction des spécificités de son projet.

### 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.

```yaml
#app/config.yml
open_orchestra_backoffice:
    field_types:
        online_checkbox:
            # le label utilisé dans le sélecteur de type dans le type de contenu
            label: Case à cocher
            # le FormType utilisé dans le formulaire du contenu (namespace ou alias)
            type: Symfony\Component\Form\Extension\Core\Type\CheckboxType
            # le type du format de désérialisation (voir␣
↪JMS\Serializer\GraphNavigator)
            deserialize_type: string
            # la stratégie de recherche dans le tableau des contenus
            search: online_checkbox
            options:
                required:
                    # la valeur par défaut de l'option
                    default_value: false
                value:
                    # la valeur par défaut de l'option
                    default_value: online
            default_value:
                # le FormType utilisé pour contribuer la valeur par défaut du champ
                type: Symfony\Component\Form\Extension\Core\Type\CheckboxType
                options:
                    # le label utilisé pour la valeur par défaut
                    label: Valeur par default
                    # l'option obligatoire ou non pour la valeur par défaut
                    required: false
    options:
        value:
            # le FormType utilisé pour contribuer la valeur de l'option
            type: Symfony\Component\Form\Extension\Core\Type\TextType
            # le label de l'option
            label: Valeur associée
            # le caractère obligatoire ou non de cette option
            required: true
```

On obtient ainsi dans le formulaire de type de contenu :




Et dans le formulaire d'un contenu voiture :


On remarque dans l'exemple la définition d'une nouvelle option "value" contribuée 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.

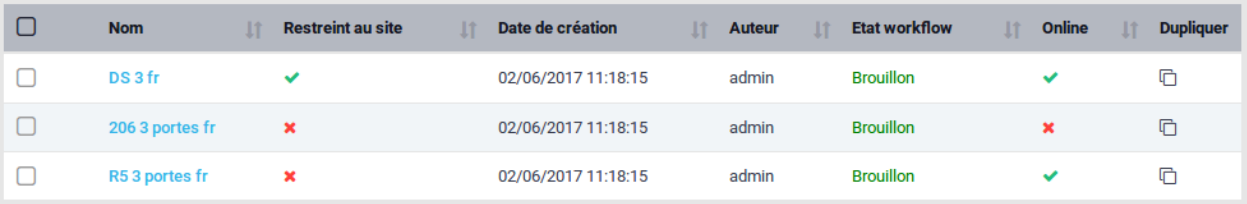
Voici le code du transformer :

```php
// src/AcmeBundle/ValueTransformer/OnlineCheckboxToHtmlStringTransformer.php
namespace AcmeBundle\ValueTransformer\Strategies;

use OpenOrchestra\Backoffice\ValueTransformer\ValueTransformerInterface;

/**
 * Class OnlineCheckboxToHtmlStringTransformer
 */
class OnlineCheckboxToHtmlStringTransformer implements ValueTransformerInterface
{
    /**
     * @param array $data
     *
     * @return string
     */
    public function transform($data)
    {
        return ($data) ?
            '<i aria-hidden="true" class="fa fa-check text-success"></i>' :
            '<i aria-hidden="true" class="fa fa-close text-danger"></i>';
    }

    /**
     * @param string $fieldType
     * @param mixed  $value
     *
     * @return bool
     */
    public function support($fieldType, $value)
    {
        return gettype($value) == 'boolean' && ($fieldType == 'online_checkbox');
    }

    /**
     * @return string
     */
    public function getName()
    {
        return 'online_checkbox';
    }
}
```

et le paramétrage permettant de l'activer :

```yaml
# app/config/services.yml
services:
    acme_bundle.value_transformer.online_checkbox:
        class:␣
↪AcmeBundle\ValueTransformer\Strategies\OnlineCheckboxToHtmlStringTransformer
        tags:
            - { name: open_orchestra_backoffice.value_transformer.strategy }
```

On obtient ainsi la liste de consultation suivante :



| | Nom | Restreint au site | Date de création | Auteur | Etat workflow | Online | Dupliquer |
|---|---|---|---|---|---|---|---|
| ☐ | DS 3 fr | ✔ | 02/06/2017 11:18:15 | admin | Brouillon | ✔ | |
| ☐ | 206 3 portes fr | ✖ | 02/06/2017 11:18:15 | admin | Brouillon | ✖ | |
| ☐ | R5 3 portes fr | ✖ | 02/06/2017 11:18:15 | admin | Brouillon | ✔ | |

### 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.

```
// src/AcmeBundle/Ressources/public/ecmascript/Acme/Service/SearchFormGroup/
↪OnlineCheckboxForm.js
import TemplateManager          from '../../../../OpenOrchestra/Service/TemplateManager
↪'
import AbstractSearchFormGroup from '../../../../OpenOrchestra/Service/
↪SearchFormGroup/AbstractSearchFormGroup'

/**
 * @class OnlineCheckboxForm
 */
class OnlineCheckboxForm extends AbstractSearchFormGroup
{
    /**
     * test if field is supported
     *
     * @param {Object} field
     */
    support(field) {
        // check on the value setted in the yml
        return field.search == 'online_checkbox';
    }

    /**
     * render the field
     *
     * @param {Object} field
     */
    render(field) {
        return TemplateManager.get('SearchFormGroup/onlineCheckboxForm')({
            field: field
        });
    }
}

// unique instance of OnlineCheckboxForm
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).

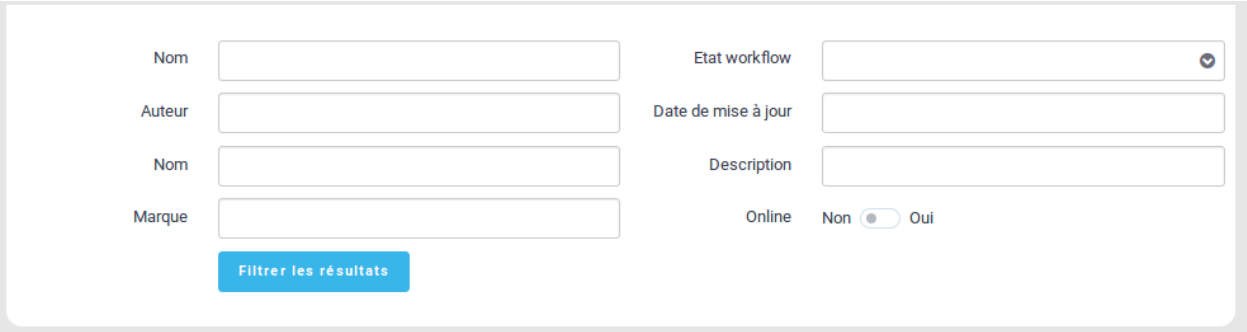
```
// src/AcmeBundle/Ressources/public/ecmascript/Acme/Application/AcmeSubApplication.js
import SearchFormGroupManager  from '../../../../OpenOrchestra/Service/
↪SearchFormGroup/Manager'
```

```
import CheckboxSearchFormGroup from '../../../../OpenOrchestra/Service/
↪SearchFormGroup/OnlineCheckboxForm'

/**
 * @class AcmeSubApplication
 */
class AcmeSubApplication
{
    /**
     * Run sub Application
     */
    run() {
        this._initSearchFormGroupManager;
    }

    /**
     * Initialize field search library
     * @private
     */
    _initSearchFormGroupManager() {
        SearchFormGroupManager.add(CheckboxSearchFormGroup);
    }
}
```

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

```
<!-- src/AcmeBundle/Ressources/public/template/SearchFormGroup/onlineCheckboxForm._
↪tpl.html -->
<label for="attributes.online" class="control-label col-md-4">
    Online
</label>
<div class="switch-button">
    <span>Non</span>
    <label class="switch">
        <input id="attributes.online" name="attributes.online" value="1" type=
↪"checkbox">
        <div class="slider"></div>
    </label>
    <span>Oui</span>
</div>
```

On obtient le moteur de recherche suivant.



### 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 :

```php
// src/AcmeBundle/Pagination/MongoTrait/FilterTypeStrategy/Strategies/
↪OnlineCheckboxFilterStrategy.php
namespace AcmeBundle\Pagination\MongoTrait\FilterTypeStrategy\Strategies;

use OpenOrchestra\Pagination\FilterType\FilterTypeInterface;

/**
 * Class OnlineCheckboxFilterStrategy
 */
class OnlineCheckboxFilterStrategy implements FilterTypeInterface
{
    const FILTER_TYPE =  'online_checkbox';

    /**
     * @param string $type
     *
     * @return bool
     */
    public function support($type)
    {
        return $type === self::FILTER_TYPE;
    }

    /**
     * @param string $name
     * @param string $value
     * @param string $documentName
     * @param string $format
     *
     * @return array
     */
    public function generateFilter($name, $value, $documentName='', $format='')
    {
        if ($value === 'true' || $value === '1') {
            return array($name => true);
        } elseif ($value === 'false' || $value === '0') {
            return array($name => false);
        }

        return null;
    }

    /**
     * @return string
     */
```

```
    public function getName()
    {
        return 'online_checkbox_filter';
    }
}
```

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

```
# app/config/services.yml
services:
    acme_bundle.value_transformer.online_checkbox:
        class: AcmeBundle\Pagination\MongoTrait\FilterTypeStrategy\Strategies\OnlineCheckboxFilterStrategy
        tags:
            - { 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.

```
# for example, if WAMP is used ...
c:\> move symfony c:\wamp\bin\php
# ... then, execute the command as:
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 defaut - when you type-hint `SiteUpdateManager`. That's why creating the alias is a good idea.

### code-block : : ini

```ini
; Linux and macOS systems
curl.cainfo = "/path/to/cacert.pem"

; Windows systems
curl.cainfo = "C:\path\to\cacert.pem"
```

### code-block : : text

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

### code-block : : terminal

```
# Linux and macOS systems
$ sudo mkdir -p /usr/local/bin
$ sudo curl -LsS https://symfony.com/installer -o /usr/local/bin/symfony
$ sudo chmod a+x /usr/local/bin/symfony
```

### code-block : : yaml

```yaml
# config.yml

# FOSUserBundle
fos_user:
    db_driver: mongodb
    firewall_name: main
    user_class: OpenOrchestra\UserBundle\Document\User
    group:
        group_class: OpenOrchestra\GroupBundle\Document\Group
```

### code-block : : javascript

```javascript
module.exports = function(grunt) {
  var appConfig = require('./grunt/app_config.js');
  var GruntConfigBuilder = require(appConfig.GruntConfigBuilder);

  GruntConfigBuilder.init(grunt, appConfig);
};
```

### code-block : : bash

```bash
./bin/grunt
```

### code-block : : php

```php
class AppKernel extends Kernel
{
    // ...

    public function registerBundles()
    {
        $bundles = array(
            // others bundles
            new Doctrine\Bundle\MongoDBBundle\DoctrineMongoDBBundle(),
            new FOS\HttpCacheBundle\FOSHttpCacheBundle(),
        );

        // ...
    }
}
```