Mục tiêu của document này: Sử dụng **SonataAdminBundle** để quản lý một số đối tượng

* Config
* List
* Filter
* Form

[Overview](#h.2dnloam8wrf9)

[Step 1: Define SonataAdminBundle routes](#h.nxoomnrmbe3)

[Step 2: Setup the persistence service (ORM, ODM, ...)](#h.f2mifeg3xg3i)

[Step 3: Create admin class](#h.65pdfyipek2j)

[Step 4: Create admin service](#h.ql30gzz7adhz)

[Step 5: Configuration](#h.jgqlb0xxa6pq)

[Step 6: Security](#h.vmuz4xb9b8fp)

[Advanced Admin class](#h.oq3isxsibiy)

[Listing](#h.6q7k23wdib6k)

[Form](#h.qi2oveirg8vr)

# Overview

Chúng ta sẽ xây dựng ứng dụng nhỏ quản lý bài viết, có quan hệ như ở dưới. Tuy nhiên chúng ta sẽ chỉ đi xây dựng ví vụ một vài đối tượng cần quản lý (sẽ có hướng dẫn tạo từng cái ở dưới)

* NewsCategory
  + id (integer - AI)
  + name (string 255)
  + parent (NewsCategory)
  + lft (integer)
  + rgt (integer)
  + articles (manytomany Article)
* Article
  + id (integer - AI)
  + title (string 255)
  + alias (string 255)
  + image (string 255)
  + content (text)
  + categories (manytomany NewsCategory)
* Comment
  + id (integer - AI)
  + article (ManyToOne Article)
  + user\_id (integer - allow null) (may use ManyToOne User - User is the default of system)
  + content (text)

Nội dung sau được tham khảo tại trang <http://sonata-project.org/bundles/admin/master/doc/reference/getting_started.html>

# Step 1: Define SonataAdminBundle routes

Thêm vào file routing (hướng dẫn này sử dụng file yml):

|  |
| --- |
| *# app/config/routing.yml* admin:  resource: '@SonataAdminBundle/Resources/config/routing/sonata\_admin.xml'  prefix: /admin  \_sonata\_admin:  resource: .  type: sonata\_admin  prefix: /admin |

# Step 2: Setup the persistence service (ORM, ODM, ...)

Cái này mặc định rồi, khỏi quan tâm.

# Step 3: Create admin class

Lớp admin dùng để mapping các đối tượng vào giao diện quản lý (list, forms, show). Cách đơn giản nhất là kế thừa lại lớp Admin của SonataAdminBundle.

Code dưới đây là ví dụ về quản trị cho Article.

* Vị trí: thường đặt trong thư mục Admin\Entity của Bundle đang dùng
* Tên thường dùng: TênLớpAdmin

|  |
| --- |
| <?php  namespace MS\Bundle\NewsBundle\Admin\Entity;  use Sonata\AdminBundle\Admin\Admin;  use Sonata\AdminBundle\Form\FormMapper;  use Sonata\AdminBundle\Datagrid\DatagridMapper;  use Sonata\AdminBundle\Datagrid\ListMapper;  use Sonata\DoctrineORMAdminBundle\Datagrid\ProxyQuery;  use Symfony\Component\HttpFoundation\RedirectResponse;  class ArticleAdmin extends Admin  {  protected function configureListFields(ListMapper $listMapper)  {  $listMapper  ->add('title', null, array('label'=>'ms\_lb\_news\_title'))  ->add('alias', null, array('label'=>'ms\_lb\_news\_alias'))  ->add('image', null, array('label'=>'ms\_lb\_news\_image'))  ->add('content', null, array('label'=>'ms\_lb\_news\_content'))  ->add('categories', null, array('label'=>'ms\_lb\_news\_categories'))  ->add('\_action', 'actions', array(  'actions' => array(  'edit' => array(),  'delete' => array(),  )));  parent::configureListFields($listMapper);  }  protected function configureDatagridFilters(DatagridMapper $datagridMapper)  {  $datagridMapper  ->add('title')  ->add('content')  ->add('categories')  ;  }  protected function configureFormFields(FormMapper $formMapper)  {  $formMapper  ->add('title')  ->add('alias')  ->add('image')  ->add('content')  ->add('categories')  ;  }  } |

# Step 4: Create admin service

Để giao diện administrator nhận biết được lớp quản lý mới này, chúng ta cần khai báo trong dịch vụ.

|  |
| --- |
| #MS\Bundle\NewsBundle\Resources\config\admin\_orm.yml  ms.news.article:  class: MS\Bundle\NewsBundle\Admin\Entity\ArticleAdmin  tags:  - { name: sonata.admin, manager\_type: orm, group: News, label: Article }  arguments: [null, MS\Bundle\NewsBundle\Entity\Article, null]  calls:  - [setTranslationDomain, [news.admin.translation\_domain]] |

|  |  |
| --- | --- |
| A description... | Nếu sử dụng file admin\_orm.yml, bạn cần kiểm tra việc load file của Symfony, tham khảo các dòng lệnh sau:  #MS\Bundle\NewsBundle\DependencyInjection\MSNewsExtension.php  public function load(array $configs, ContainerBuilder $container)  {  //...  $loader = new Loader\YamlFileLoader($container, new FileLocator ( \_\_DIR\_\_ . '/../Resources/config' ));  //...  $loader->load('admin\_orm.yml');  } |

# Step 5: Configuration

Tới đây, chúng ta đã cơ bản có được mô hình trang quản trị. Để nhanh chóng trong việc cấu hinh trang administration, chúng ta tạo một số cấu hình như sau:

|  |
| --- |
| *# app/config/config.yml* sonata\_admin:  title: MS Project  title\_logo: /bundles/sonataadmin/logo\_title.png  templates:  *# default global templates*  layout: SonataAdminBundle::standard\_layout.html.twig  ajax: SonataAdminBundle::ajax\_layout.html.twig   *# default actions templates, should extend a global templates*  list: SonataAdminBundle:CRUD:list.html.twig  show: SonataAdminBundle:CRUD:show.html.twig  edit: SonataAdminBundle:CRUD:edit.html.twig  dashboard:  blocks:  *# display a dashboard block*  - { position: left, type: sonata.admin.block.admin\_list } |

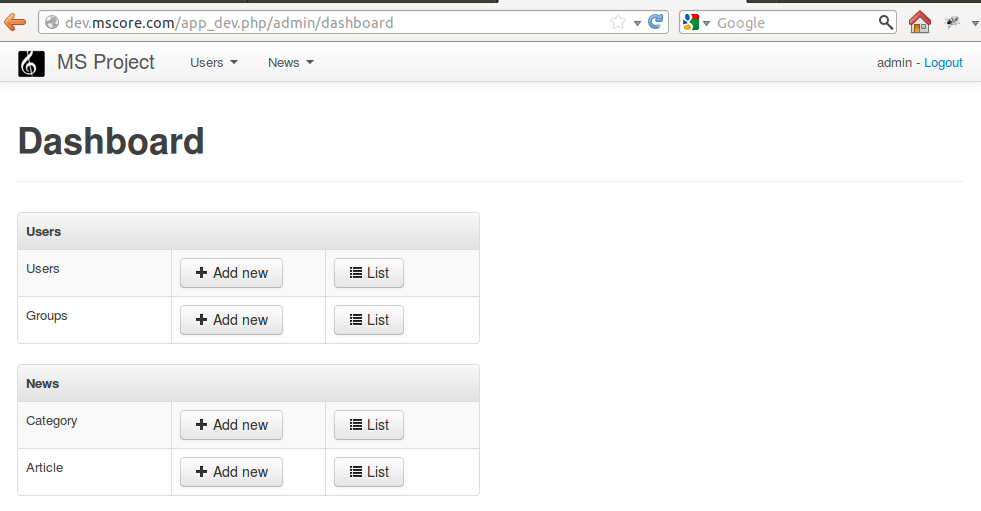
Một số thiết lập hữu ích khác bạn có thể tra tại tài liệu hướng dẫn của SonataAdminBundle trong chương Configuration.

# Step 6: Security

Bước cuối cùng và vô cùng quan trọng đó là bảo mật. Mặc định, SonataAdminBundle không cung cấp quản lý User để đảm bảo tính mềm dẻo. Tuy nhiên đây là tính năng mà hệ thống chúng ta cần cung cấp. Dự án của Sonate có cung cấp SonataUserBundle tích hợp bundle rất phổ biến FOSUserBundle. Vì vậy hãy tham khảo tài liệu phần bảo mật của Bundle này.

Ví dụ:

|  |
| --- |
| *# app/config/config.yml*  sonata\_admin:  security:  handler: sonata.admin.security.handler.role  #...  sonata\_block:  default\_contexts: [cms]  blocks:  sonata.admin.block.admin\_list:  contexts: [admin] |



Như vậy chúng ta đã có trang quản trị cơ bản. Bây giờ chúng ta sẽ đi sâu hơn vào Lớp Admin để có thể hiểu rõ hơn và tùy biến nhiều hơn cho phần quản trị.

# Advanced Admin class

## Listing

Mục tiêu cho ví vụ:

* List các mục sau cho bài viết: title, thumb image in content, content (about 50 word without image), categories, action buttons (edit, delete)

Đầu tiên để đơn giản chúng ta list title, content, categories và action buttons.

Xem trong function configureListFields

|  |
| --- |
| #src\MS\Bundle\NewsBundle\Admin\Entity\ArticleAdmin.php  //...  class ArticleAdmin extends Admin  {  protected function configureListFields(ListMapper $listMapper)  {  $listMapper  ->add('title', null, array('label'=>'ms\_lb\_news\_title'))  ->add('content', null, array('label'=>'ms\_lb\_news\_content'))  ->add('categories', null, array('label'=>'ms\_lb\_news\_categories'))  ->add('\_action', 'actions', array(  'actions' => array(  'edit' => array(),  'delete' => array(),  )));  parent::configureListFields($listMapper);  }  //...  } |

Ta thấy biến $listMapper sẽ thêm các trường trong đối tượng Article đã có, sử dụng hàm **add()**:

* Tham số thứ nhất: tên trường!! (hoặc tên hàm trả về dữ liệu đã rút gọn. Ví dụ ta viết thêm cho Article hàm **getImageThumbPath**() trả về đường dẫn của hình nhỏ trong bài viết. thì ở đây ta add tên trường là **'image\_thumb\_path'**) - vì vậy ở đây tôi gọi là "tên trường" cho dễ hiểu.
* Tham số thứ 2: là kiểu dữ liệu hiển thị, mặc định là null. Các kiểu có thể tham khảo trong phần [Types available](http://sonata-project.org/bundles/doctrine-orm-admin/master/doc/reference/list_field_definition.html" \l "types-available) của SonataAdminBundle. Chút xíu nữa thôi chúng ta sẽ sử dụng các field type tự định nghĩa cho việc list ra hình thumbnail chứa trong content.
* Tham số thứ 3 là mảng các options tương ứng với kiểu dữ liệu ta chọn trong tham số thứ 2. Một số options mặc định hay dùng:
  + label: tên của cột dữ liệu
  + template: tên của template ta sẽ sử dụng cho trường hợp cần thay đổi về hiển thị đối với một số trường hợp (trong ví dụ ta sẽ làm hiển thị hình ảnh)

Ngoài hàm add(), ta có thể sử dụng hàm **addIdentifier()** với các tham số tương tự. Khi dùng addIdentifier(), hiển trường dữ liệu đó sẽ là link để edit đối tượng đó.

Như vậy chúng ta đã có được list đơn giản cho Article (thiếu thumb image, content chưa rút gọn xuống, loại bỏ các thẻ html)

**Advanced listing - Custom listing**

Chúng ta bắt đầu với content. Muốn thay đổi theo ý muốn, hãy để ý tham số thứ 3 của hàm **add()** đó là option template để thay đổi hiển thị. Chúng ta sẽ làm như sau:

|  |
| --- |
| $listMapper  //...  ->add('content', 'string', array('label'=>'ms\_lb\_news\_content', 'template'=>'MSNewsBundle:Admin:list\_content.html.twig'))  //..  ; |

Lú này, hiển thị của ô chứa content sẽ biểu diễn theo cấu trúc của file trong Resources\views\Admin\list\_content.html.twig.

|  |
| --- |
| {# MS\Bundle\NewsBundle\Resources\views\Admin\list\_content.html.twig #}  {% extends 'SonataAdminBundle:CRUD:base\_list\_field.html.twig' %}  {% block field %}  <div>  {{ object.content|striptags }}  </div>  {% endblock %} |

Trong file twig ở trên, object chính là entity chúng ta đang sửa hoặc thêm mới trong form. Vì vậy nó có các phương thức của entiy Article: getContent(). Vì vậy nội dung ở đây chúng ta có thể thêm, chỉnh sửa hoặc gắn nhiều nội dung các trường khác vào. (Vì vậy, theo mục tiêu ban đầu, ta sẽ xử lý để cho nội dung content sẽ ngắn lại 50 ký tự. Ở đây bạn hãy tự nghĩ cách, vì có thể tùy biến rất đa dạng)

## Form

Mục tiêu cho ví dụ:

* Tạo ra form nhập liệu cho Article
* Tùy biến các yêu cầu như required
* Sử dụng các loại form\_type của Sonata cung cấp
* Biến nội dung nhập content sang dạng WYSIWYG (sử dụng tinymce)

Đầu tiên, tạo form nhập liệu đơn giản

|  |
| --- |
| #src\MS\Bundle\NewsBundle\Admin\Entity\ArticleAdmin.php  //...  class ArticleAdmin extends Admin  {  //...  protected function configureFormFields(FormMapper $formMapper)  {  $formMapper  ->add('title')  ->add('alias')  ->add('image')  ->add('content')  ->add('categories')  ;  }  //...  } |

Cũng tương tự như Listing, trong form chúng ta có biến FormMapper $formMapper sử dụng phương thức add() để gắn các trường dữ liệu vào form:

* Tham số thứ nhất: trên trường (tương tự như trên ListMapper)
* Tham số thứ hai: tên kiểu dữ liệu (để null thì nó sẽ tự động chọn tương ứng với kiểu của trường đó). Các kiểu dữ liệu hỗ trợ sẵn có thể tham khảo tại mục [Types available](http://sonata-project.org/bundles/doctrine-orm-admin/master/doc/reference/form_field_definition.html" \l "types-available) của tài liệu SonataAdminBundle. Chúng ta cũng có thể tự định nghĩa thêm kiểu này (sẽ có hướng dẫn ở dưới)
* Tham số thứ 3 tương tự ListMapper là mảng các options tương ứng với kiểu dữ liệu ta chọn trong tham số thứ 2. Các tham số thường dùng:
  + required: boolean -> trường này có bắt buộc hay không. Nếu không quy định option này, mặc định sẽ lấy theo sự bắt buộc mà entity đó quy định (nullable)
  + label: string -> nhãn của trường.
  + attr: array -> mảng bổ sung các thuộc tính cho control của trường (ví dụ: 'attr'=>aray('class'=>'tinymce', 'data-theme'=>'medium'))

Đôi khi chúng ta muốn nhóm một số trường liên quan và cần thiết nhất lên trên, và một số trường đặc thù, hoặc có thể rỗng xuống dưới. Sử dụng with() và end() để có các nhóm.

|  |
| --- |
| $formMapper  ->with('General')  ->add('title')  ->add('alias')  ->add('content')  ->add('categories')  ->end()  ->with('Image')  ->add('image')  ->end()  ; |

**Advanced - Custom form**

Chúng ta xem lại giao diện form của Article: alias chúng ta sẽ dùng script để tự động sinh ra dựa trên title. (chúng ta bỏ qua việc alias giống nhau nhé - cái này sẽ tùy cách xử lý của mỗi người). Bây giờ chúng ta sẽ phải tìm giải pháp cho việc tự sinh alias.

Như đã thấy, việc thêm vào input nhập cho các trường đơn giản dùng hàm add(). Nhưng sẽ là khó cho việc thêm các script để xử lý riêng cho mỗi input hoặc bổ sung các nội dung khác. Vì vậy chúng ta sẽ không sử dụng input mặc định nữa mà sẽ thay đổi type của trường này bằng type mới. Cách thức như các bước dưới đây.

|  |  |
| --- | --- |
| Step 1 | Tạo một Field Type, ở ví dụ này ta đặt là AliasType, kế thừa AbstractType:  <?php  namespace MS\Bundle\NewsBundle\Form\Type;  use Symfony\Component\Form\AbstractType;  use Symfony\Component\Form\FormBuilderInterface;  use Symfony\Component\Form\FormView;  use Symfony\Component\Form\FormInterface;  class AliasType extends AbstractType  {  public function getDefaultOptions(array $options)  {  $options = parent::getDefaultOptions($options);  $options['based\_input'] = '\_title';  return $options;  }  /\*\*  \* {@inheritDoc}  \*/  public function buildView(FormView $view, FormInterface $form, array $options)  {  $view->set('based\_input', $form->getAttribute('based\_input'));  }  /\*\*  \* {@inheritDoc}  \*/  public function buildForm(FormBuilderInterface $builder, array $options)  {  $builder  ->setAttribute('based\_input', $options['based\_input'])  ;  }  public function getParent()  {  return 'text';  }  public function getName()  {  return 'alias';  }  } |
| Step 2 | Creating a Template for the Field  {# src/MS/Bundle/NewsBundle/Resources/views/Form/fields.html.twig #}  {% block alias\_widget %}  {% spaceless %}  {{ form\_widget(form) }}  {% endspaceless %}  {% endblock %}  Ở trên, nội dung {{ form\_widget(form) }} sẽ trả lại nội dung của input giống như mặc định. Ta sẽ bổ sung script vào phía sau. Sau hướng dẫn các bước này chúng ta sẽ thay đổi lại nội dung của alias\_widget để tự động điền dữ liệu vào theo nội dung title. |
| Step 3 | Config  Đảm bảo config cho hệ thống hiểu template của form type mới trong NewsBundle.   |  | | --- | | *# app/config/config.yml* twig:  form:  resources:  - 'MSNewsBundle:Form:fields.html.twig' | |
| Step 4 | Creating your Field Type as a Service  # src/AMS/Bundle/NewsBundle/Resources/config/services.yml  services:  form.type.alias:  class: MS\Bundle\NewsBundle\Form\Type\AliasType  tags:  - { name: form.type, alias: alias } |
| Step 5 | Enjoy: trong hàm configureFormFields của phần Admin, ta sử dụng kiểu alias  $formMapper  ->add('alias', 'alias', array('based\_input'=>'\_title')) |

Như vậy là chúng ta đã tự tạo ra một Filed Type. Ở step 2, ta mới chỉ tạo template hiển thị như cũ. Bây giờ ta sẽ bổ sung thêm script cho nó để tự động lấy dữ liệu từ title phía trên

|  |
| --- |
| {# src/MS/Bundle/NewsBundle/Resources/views/Form/fields.html.twig #}  {% block alias\_widget %}  {% spaceless %}    {{ form\_widget(form) }}  <script src="{{ asset('bundles/msnews/js/common.js') }}" ></script>  <script lang="javascript">  $('input[id$="{{based\_input}}"]').change(function(){  $('input[id$="\_{{form.name}}"]').val(getAliasString($(this).val()));  });  </script>  {% endspaceless %}  {% endblock %} |

Chú ý: ở trong nội dung của alias\_widget trên ta thấy có 1 biến {{**based\_input**}}. Biến based\_input đó được truyền vào thông qua các options khi sử dụng form type này.