Document Training and Research Symfony2

I. Install Symfony2:

1. Download Symfony 2

You can download on https://symfony.com or open your command console and execute the following commands:

```
$ sudo curl -LsS http://symfony.com/installer -o /usr/local/bin/symfony
$ sudo chmod a+x /usr/local/bin/symfony

# use the most recent version in any Symfony branch
$ symfony new my_project_name 2.3
```

```
ex: $ symfony new demo 2.3
```

Then, move the download Symfony2 file to your project's directory
ex: */var/www/html/demo*

2. Configuring a Web Server

You directory to "***/etc/apache2/sites-available***"
Create the applications vhost "**file.conf**"
ex: "***/etc/apache2/sites-available/demo.local.conf***"
Affter open file **demo.local.conf**:

Configuring file:

```
<VirtualHost *:80>
        ServerAdmin webmaster@localhost
        ServerName my_domain.com
        ServerAlias www.my_domain.com
        DocumentRoot "/var/www/html/my_project_name/web/"
</VirtualHost>
ex:
<VirtualHost *:80>
        ServerAdmin webmaster@localhost
        ServerName demo.local
```

```
                ServerAlias www.demo.local
                DocumentRoot "/var/www/html/demo/web/"
            </VirtualHost>
```

Configuring file hosts "/etc/hosts"

```
ex:     #127.0.0.1 localhost
        127.0.0.1 demo.local
```

Affter finally activate the new vhost

```
        $ sudo a2ensite my_application_dev
        ex: $ sudo a2ensite  demo.local.conf
        $ sudo service apache2 restart
```

Affter open browser test : *demo.local*

II. Create Bundle

1. Generating a New Bundle:

Open your command console and execute the following commands:

```
                $ cd /var/www/html/my_project_name
                ex: $ cd /var/www/html/demo
                $ php app/console generate:bundle
                or
                $ php app/console generate:bundle
--namespace=folder_name/folder_bundle_name/name_Bundle –no-interaction
        ex: php app/console generate:bundle --namespace=Acme/Bundle/BlogBundle –no-interaction
```

2. Generating a New Controller

```
        $ php app/console generate:controller --no-interaction –controller=name_Bundle:Post
        ex: $ php app/console generate:controller --no-interaction –
controller=StudentBundle:Student
```

or

$ php app/console generate:controller

3. Creating Routes:

```
        # app/config/config.yml
        framework:
                # ...
                router: { resource: "%kernel.root_dir%/config/routing.yml" }

        # app/config/routing.yml
```

Affter open file “*routing.yml*”

```
#...
name_router:
 resource: "@Name_Bundle/Resources/config/routing.yml"
prefix:   /
ex:
student:
 resource: "@StudentBundle/Resources/config/routing.yml"
prefix:   /
```

Affter directory to “*StudentBundle/Resources/config*” Create file “*routing.yml*”

```
          name_router:
          path:  /student
          defaults: { _controller: Name_Bundle:Contrller:index }
ex:
          student:
          path:  /student
          defaults: { _controller: StudentBundle:Student:index }
```

Affter open browser test : demo.local/app_dev.php/student

III. Entity

1. Databases and Doctrine

a. Configuring the Database

```
# app/config/parameters.yml
parameters:
database_driver:    pdo_mysql
database_host:      localhost
database_name:      test_project
database_user:      root
database_password:  password
# …
```

Now that Doctrine knows about your database, you can have it create the database for you:

```
$ php app/console doctrine:database:drop --force
$ php app/console doctrine:database:create
```

b. Creating an Entity Class

```
$ php app/console doctrine:generate:entity
```

ex:

→ The Entity shortcut name: Student

→ Configuration format (yml, xml, php, or annotation)[annotation]: annotation

→ New field name (press <return> to stop adding fields): …..

Affter create Entity class:

```
namespace StudentBundle\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
* @ORM\Entity(repositoryClass="StudentBundle\Entity\StudentRepository")
* @ORM\Table(name="student")
*/
class Student
{
        /**
        * @ORM\Column(type="integer")
        * @ORM\Id
        * @ORM\GeneratedValue(strategy="AUTO")
        */
        protected $id;

        /**
        * @ORM\Column(type="string", length=100)
        */
        protected $name;

        /**
        * @ORM\Column(type="integer")
        */
        protected $age;
        ……..
        …...............
}
```

#Persisting Objects to the Database

```
/ src/StudentBundle/Controller/DefaultController.php
```

```
// ...
use StudentBundle\Entity;
use Symfony\Component\HttpFoundation\Response;

// ...
public function createAction()
{
        $student = new Student();
        $student->setName('AnLam');
        $student->setAge('19');

        $em = $this->getDoctrine()->getManager();

        $em->persist($student);
        $em->flush();

        return new Response('Created Student id '.$student->getId());
}
```

C. Entity Relationships/Associations

ex:

```
$ php app/console doctrine:generate:entity \
--entity="StudentBundle:School" \
--fields="name:string(255)"
```

1. Relationship Mapping Metadata

```
// src/StudentBundle/Entity/School.php
// ...
use Doctrine\Common\Collections\ArrayCollection;
class  School
{
        // ...

        /**

        * @ORM\OneToMany(targetEntity="Student", mappedBy="school")
        */
        protected $student;

        public function __construct()
        {
        $this-> student = new ArrayCollection();
        }
}
```

Next, since each `Student` class can relate to exactly one `School` object, you'll want to add a `$school` property to the `Student` class:

```
// src/StudentBundle/Entity/Student.php

// ...
class student
{
        // ...

         /**
        * @ORM\ManyToOne(targetEntity="School", inversedBy="student")
        * @ORM\JoinColumn(name="school_id", referencedColumnName="id")
        */
        protected $school;
}
```

Finally, now that you've added a new property to both the `School` and S`tudent` classes, tell Doctrine to generate the missing getter and setter methods for you:

```
$ php app/console doctrine:generate:entities  StudentBundle
```

Before you continue, be sure to tell Doctrine to add the new `school` table, and student`.school_id` column, and new foreign key:

```
$ php app/console doctrine:schema:update --force
```

IV. API

1. Download the Bundle

Open a command console, enter your project directory and execute the following command to download the latest stable version of this bundle:

if project not found file composer.phar

```
$ sudo php composer.phar install
```

Affter:

```
$ php composer.phar require friendsofsymfony/rest-bundle
```

***2.* Enable the Bundle**

Then, enable the bundle by adding the following line in the `app/AppKernel.php` file of your project:

```
// app/AppKernel.php
class AppKernel extends Kernel
{
        public function registerBundles()
```

```
		{
		$bundles = array(
			// ...
			new FOS\RestBundle\FOSRestBundle(),
		);

	// ...
		}
	}
```

3. Enable a Serializer

```
	$ php composer.phar require jms/serializer-bundle
```

Then, enable the bundle by adding the following line in the `app/AppKernel.php` file of your project:

```
	// app/AppKernel.php
	class AppKernel extends Kernel
	{
		public function registerBundles()
		{
		$bundles = array(
			// ...
			JMS\SerializerBundle\JMSSerializerBundle(),
		);

	// ...
		}
	}
```

4. Create Bundle API (RestBundle)

# name Bundle: RESTBundle

```
	$ php app/console generate:bundle
```

a. Router

```
		# app/config/routing.yml
		#...
		name_router:
		 resource: "@Name_Bundle/Resources/config/routing.yml"
		prefix:   /api
		ex:
		rest_student:
		 resource: "@RESTBundle/Resources/config/routing.yml"
		prefix:   /api
```

Affter directory to “*StudentBundle/Resources/config*” Create file “*routing.yml*”

```
            name_router:
            path:  /school
            defaults: { _controller: Name_Bundle:Contrller:index }
ex:
            rest_student:
            path:  /school
            defaults: { _controller: RESTBundle:SchoolRest:index }

            Affter open browser test : demo.local/app_dev.php/student
```

b. **The view layer**

```
<?php

use FOS\RestBundle\Controller\FOSRestController;

class SchoolRestController extends FOSRestController
{
            public function getSchoolsAction()
            {
                        $data = ...; // get data, in this case list of users.
                        $view = $this->view($data, 200)
                                    ->setTemplate("Student:School:index.html.twig")
                                    ->setTemplateVar('users')
                        ;

                        return $this->handleView($view);
            }

            public function viewAction($id)
            {
                        $em = $this->getDoctrine()->getEntityManager();
                        $school = $em->getRepository('StudentBundle:School')->find($id);
                        return  array('school_'.$id=>$school);
            }
}
```

5. API Doc (NelmioApiDocBundle)

a. Installation

```
$ php composer.phar require nelmio/api-doc-bundle
```

b. Register the bundle in `app/AppKernel.php`:

```
// app/AppKernel.php
public function registerBundles()
{
            return array(
                        // ...
```

```
                new Nelmio\ApiDocBundle\NelmioApiDocBundle(),
        );
}
```

c. Import the routing definition in routing.yml:

```
# app/config/routing.yml
NelmioApiDocBundle:
resource: "@NelmioApiDocBundle/Resources/config/routing.yml"
prefix:   /api/doc
```

d. Enable the bundle's configuration in app/config/config.yml:

```
# app/config/config.yml
nelmio_api_doc: ~
```

f. Ex:

```
<?php

use FOS\RestBundle\Controller\FOSRestController;
use Nelmio\ApiDocBundle\Annotation\ApiDoc;
class SchoolRestController extends FOSRestController
{
        /**
        * @ApiDoc(
        * resource=true,
        * description="Get list school",
        * requirements={
        *     {"name"="_format", "dataType"="String", "requirement"="",
"description"="json|xml" }
        *   },
        * statusCodes = {
        *     200 = "Returned when successful",
        *   },
        * )
        * )
        * @return View
        */
        public function indexAction()
         {
                $data = ...; // get data, in this case list of users.
                $view = $this->view($data, 200)
                        ->setTemplate("Student:School:index.html.twig")
                        ->setTemplateVar('users')
                ;

                return $this->handleView($view);
        }
        /**
```

```
                                        * @ApiDoc(
                                        * resource=true,
                                        * description="Get School by id",
                                        * requirements={
                                        *      {"name"="id", "dataType"="integer", "requirement"="ID",
"description"="ID for school" },
                                        *      {"name"="_format", "dataType"="string", "requirement"="xml | json",
"description"="xml | json" }
                                        *   },
                                        * )
                                        * @return View
                                        */
                                        public function viewAction($id)
                                        {
                                                $em = $this->getDoctrine()->getEntityManager();
                                                $school = $em->getRepository('StudentBundle:School')->find($id);
                                                return  array('school_'.$id=>$school);
                                        }
                                }
```

V. Paging

1. Installation and configuration

```
                $ php composer.phar require knplabs/knp-paginator-bundle
```

Add PaginatorBundle to your application kernel

```
                // app/AppKernel.php
                public function registerBundles()
                {
                        return array(
                        // ...
                        new Knp\Bundle\PaginatorBundle\KnpPaginatorBundle(),
                                // ...
                        );
                }
```

2. Usage examples:

<u>Controller</u>

Doctrine\Common\Collection\ArrayCollection - any doctrine relation collection including

```
                        // Acme\StudentBundle\Controller\SchoolController.php

                        public function indexAction(Request $request)
                        {
                                $repository = $this->getDoctrine()
                                                ->getRepository('StudentBundle:School');
```

```
                    $schools = $repository->findAll();

                    $paginator  = $this->get('knp_paginator');
                    $pagination = $paginator->paginate(
                        $schools,
                        $request->query->getInt('page', 1)/*page number*/,
                            10/*limit per page*/
                    );

                    // parameters to template
                    return $this->render('AcmeMainBundle:Article:list.html.twig', array('schools'
=> $pagination));
                }
```

View

```
                //...............
                {% for school in schools %}
                <tr style="border-bottom: solid 1px #D3D3D3;">
                    <td style="padding:3px 5px;">{{school.id}}</td>
                    <td>{{ school.name }}</td>
                    <td>{{ school.phone }}</td>
                    <td>{{ school.address }}</td>
                    <td><a href="{{ app.request.getBaseURL() }}/school/update/
{{school.id}}">Edit</a> | <a href="{{ app.request.getBaseURL() }}/school/remove/
{{school.id}}">Remove</a></td>
                </tr>
                {% endfor %}
                //........
                <div class="navigation">
                    {{ knp_pagination_render(schools) }}
                </div>
                //...............
```

VI. Generating a CRUD Controller Based on a Doctrine Entity

1. Creating/Configuring Services in the Container

The `generate:doctrine:crud` generates a basic controller for a given entity located in a given bundle. This controller allows to perform the five basic operations on a model.

1. Listing all records,
2. Showing one given record identified by its primary key,
3. Creating a new record,
4. Editing an existing record,
5. Deleting an existing record.

By default the command is run in the interactive mode and asks questions to determine the entity name, the route prefix or whether or not to generate write actions:

```
$ php app/console generate:doctrine:crud
```

To deactivate the interactive mode, use the *--no-interaction* option but don't forget to pass all needed options:

```
$ php app/console generate:doctrine:crud --entity=StudentBundle:School
--format=annotation --with-write --no-interaction
```

Read more about CRUD at :

http://symfony.com/doc/current/bundles/SensioGeneratorBundle/commands/generate_doctrine_crud.html

VII. Service Container

1. Creating/Configuring Services in the Container:

```
# app/config/config.yml
// …..
- { resource: "@StudentBundle/Resources/config/services.yml" }

# app/config/config.yml
framework:
      secret:          xxxxxxxxxx
      form:            true
      csrf_protection: true
      router:        { resource: "%kernel.root_dir%/config/routing.yml" }
      # …
```

```
# StudentBundle/Resources/config/services.yml
parameters:
      admin.usermanager: StudentBundle\Manager\UserManager

services:
      bo.admin.user:
            class:     %admin.usermanager%
            arguments: [@parameter('some_param')]
```

ex:

*create file **UserManager.php** in folder Manager*

```
        function createEncodePassword($entity, $password)
        {
                $encoder = $this->container->get('security.encoder_factory')
                        ->getEncoder($entity);
                $password = $encoder->encodePassword($password, '');

                return $password;
        }
```

Controller:

```

        //...............
        $entity = new User();
        $form = $this->createCreateForm($entity);
        $form->handleRequest($request);
        //..........
        $password = $this->get('bo.admin.user')->createEncodePassword($entity, $entity-
>getPassword());
        //..........
```

VIII. Menu Bundle

Step 1: Download the Bundle

```
$ php composer.phar require knplabs/knp-menu-bundle "~2"
```

Step 2: Enable the Bundle

```
// app/AppKernel.php

// ...
class AppKernel extends Kernel
{
        public function registerBundles()
        {
                $bundles = array(
                        // ...

                        new Knp\Bundle\MenuBundle\KnpMenuBundle(),
                );

                // ...
        }

        // ...
}
```

Step 3: (optional) Configure the bundle

```
# app/config/config.yml

knp_menu:
# use "twig: false" to disable the Twig extension and the TwigRenderer
twig:
      template: knp_menu.html.twig
#  if true, enables the helper for PHP templates
templating: false
# the renderer to use, list is also available by default
default_renderer: twig
```

EX: Creating Menus as Services :

```
<?php
#
namespace StudentBundle\Manager;
use Knp\Menu\FactoryInterface;
use Symfony\Component\HttpFoundation\RequestStack;
class MenuBuilder
{
        private $factory;
        /**
        * @param FactoryInterface $factory
         */
         public function __construct(FactoryInterface $factory)
        {
                $this->factory = $factory;
        }
        public function createMainMenu(RequestStack $requestStack)
        {
                $menu = $this->factory->createItem('root');
                $menu->addChild('Home', array('route' => 'admin'));
                 $menu->addChild('List User', array('route' => 'admin_user'));
                /*
                 * add sub menu:
                 */
                //          $menu['List User']->addChild('Add User', array('route' =>
'admin_user_new'));

                $menu->addChild('List Schools', array('route' => 'school'));
                $menu->addChild('List Students', array('route' => 'student'));
                return $menu;
        }
}
```

Config services:

```
parameters:
    //.................
    admin.menumanager: StudentBundle\Manager\MenuBuilder
app.menu_builder:
    class:     %admin.menumanager%
    arguments: ["@knp_menu.factory"]
app.main_menu:
    class: Knp\Menu\MenuItem
    factory: ["@app.menu_builder", createMainMenu]
    arguments: ["@request_stack"]
     tags:
        - { name: knp_menu.menu, alias: main }
```

You can now render the menu directly in a template via the name given in the `alias` key above:

```
{{ knp_menu_render('main') }}
```