

Installation et configuration du framework Symfony

I. Création d'entité

On se propose ici de créer une application de gestion de produits

1. Créez l'entité Produit (la classe Produit à partir de laquelle l'ORM Doctrine va générer la table Mysql Produit)
 - a. Créez une base de données dans Mysql appelée BD_Prod,
 - b. Ouvrez le fichier de paramètres app/config/parameters.yml et indiquez le nom de la base de données Mysql, votre login (root) et votre mot de passe :

```
1 # This file is auto-generated during the composer install
2 parameters:
3     database_host: 127.0.0.1
4     database_port: null
5     database_name: BD_Prod
6     database_user: root
7     database_password: null
8     mailer_transport: smtp
9     mailer_host: 127.0.0.1
10    mailer_user: null
11    mailer_password: null
12    secret: ThisTokenIsNotSoSecretChangeIt
```

Ouvrez une fenêtre invite de commande, puis placez-vous dans

C:\xampp\htdocs\my_project_name

- c. Créer l'entité avec la commande :

C:\xampp\htdocs\my_project_name >php bin/console

doctrine:generate:entity

```

C:\xampp\htdocs\my_project_name>php bin/console doctrine:generate:entity

Welcome to the Doctrine2 entity generator

This command helps you generate Doctrine2 entities.
First, you need to give the entity name you want to generate.
You must use the shortcut notation like AcmeBlogBundle:Post.

The Entity shortcut name: CatalogueBundle:Produit
Determine the format to use for the mapping information.
Configuration format <yml, xml, php, or annotation> [annotation]:
Instead of starting with a blank entity, you can add some fields now.
Note that the primary key will be added automatically <named id>.
Available types: array, simple_array, json_array, object,
boolean, integer, smallint, bigint, string, text, datetime, datetimetz,
date, time, decimal, float, binary, blob, guid.
New field name <press <return> to stop adding fields>: nom
Field type [string]:
Field length [255]: 50
Is nullable [false]:
Unique [false]:
New field name <press <return> to stop adding fields>: prix
Field type [string]: decimal
Precision [10]:
Scale:
Is nullable [false]:
Unique [false]:
New field name <press <return> to stop adding fields>:

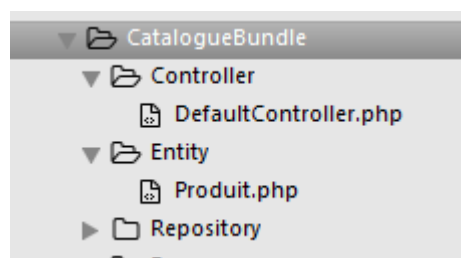
Entity generation

created .\src\CatalogueBundle\Entity/
created .\src\CatalogueBundle\Entity\Produit.php
> Generating entity class C:\xampp\htdocs\my_project_name\src\CatalogueBundle\Entity\Produit.php: OK!
> Generating repository class C:\xampp\htdocs\my_project_name\src\CatalogueBundle\Repository\ProduitRepository.php: OK!

Everything is OK! Now get to work :>.

```

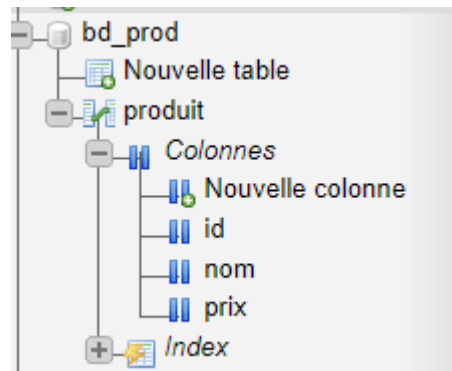
- d. Vérifier la création du fichier Produit.php dans le dossier
src/CatalogueBundle/Entity/



- f. Générer, avec Doctrine, la table Produit à partir de l'entité Produit en tapant

```
C:\xampp\htdocs\my_project_name >php bin/console doctrine:schema:create
```

g. Vérifier la création de la table Produit :



h. Créer un nouveau Controller : `ProduitController.php` dans le dossier : `C:\xampp\htdocs\my_project_name\src\CatalogueBundle\Controller` (Vous pouvez faire du copier/coller du fichier `DefaultController.php`).

Créer la méthode `addProduitAction($nom,$prix)` qui permet d'ajouter un nouveau produit. Le contenu du fichier `ProduitController.php` est le suivant :

```

1  <?php
2  namespace CatalogueBundle\Controller;
3  use CatalogueBundle\Entity\Produit;
4  use Symfony\Bundle\FrameworkBundle\Controller\Controller;
5  use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
6  class ProduitController extends Controller
7  {
8      /**
9       * @Route("/addProduit/{nom}/{prix}")
10      */
11      public function addProduitAction($nom,$prix)  {
12          $p= new Produit();
13          $p->setNom($nom);
14          $p->setPrix($prix);
15          $em=$this->getDoctrine()->getManager();
16          $em->persist($p);
17          $em->flush();
18      return $this->render('CatalogueBundle:Default:addproduit.html.twig',array('produit'=>$p));
19      } }

```

2. Ajouter la vue `addproduit.html.twig` dans le dossier :

`src/CatalogueBundle/Resources/views/Default/` son contenu est le suivant :

```

1
2  {% extends '::base.html.twig' %}
3  {% block body %}
4  ID = {{ produit.id }} <br/>
5  nom = {{ produit.nom }} <br/>
6  prix = {{ produit.prix }}
7  {% endblock %}

```

4. Tester votre travail en tapant l'URL suivante :

http://localhost/my_project_name/web/app_dev.php/addProduit/hp/1200

5. Vérifier l'insertion du produit dans la table Produit, 6. Ajouter au moins 3 produits de votre choix,

Ajouter, au Contrôleur ProduitController.php, la méthode listeProduitAction qui affiche la liste des produits, son code est le suivant :

```

/** * @Route("/listeProduits") */
public function listeProduitAction() {
    $produits = $this->getDoctrine()->getRepository("CatalogueBundle:Produit")->findAll();
    return $this->render('CatalogueBundle:Default:listeproduit.html.twig',array('produits'=>$produits));
}

```

Ajouter la vue listeproduit.html.twig dans le dossier:

src/CatalogueBundle/Resources/views/Default/ son contenu est le suivant :

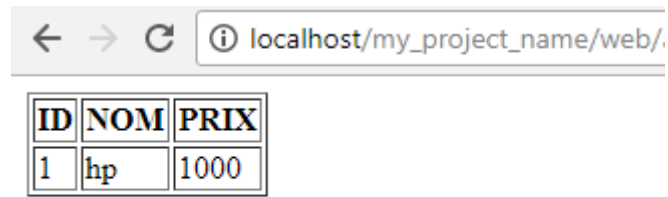
```

1  {% extends '::base.html.twig' %} {% block body %}
2  <table border="1">      <tr>
3      <th>ID </th> <th>NOM </th> <th>PRIX </th>
4      </tr>
5      {% for p in produits %}
6      <tr>
7      <td>{{ p.id }}</td>
8      <td>{{ p.nom }}</td>
9      <td>{{ p.prix }}</td>
10     </tr>
11     {% endfor %}
12     </table>
13     {% endblock %}
14

```

Tester votre travail en tapant l'URL suivante :

http://localhost/my_project_name/web/app_dev.php/listeProduits



The image shows a web browser window with the address bar displaying 'localhost/my_project_name/web/'. Below the browser, a table is displayed with the following content:

ID	NOM	PRIX
1	hp	1000